Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Raziskovanje
Ustvarjanje novih svetov
Izzivi
FRI
Študij

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

University of Ljubljana
Faculty of Computer and Information Science

Digitalna forenzika

Digital forensics

Zbornik

Proceedings

Seminarske naloge, 2019/2020

Ljubljana, 2020

Zbornik / Proceedings

Digitalna forenzika / Digital forensics, Seminarske naloge 2019/2020

Uredniki / Editors: Andrej Brodnik, Nejc Povšič, Nejc Rebernik, študenti

Ljubljana : Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, 2020.

Kazalo / Contents

1 Uvod / Introduction

Digitalna forenzika je veja forenzične znanosti, ki zajema obnovo in preiskavo gradiva najdenega v digitalnih napravah in je pogosto v povezavi z računalniškim kriminalom. Izraz digitalne forenzike je bil prvotno uporabljen kot sopomenka računalniške forenzike, vendar se je razširil, da bi zajel preiskavo vseh naprav, ki lahko shranjujejo digitalne podatke. Korenine lahko zasledimo v osebni računalniški revoluciji v poznih sedemdesetih in zgodnjih osemdesetih letih. V devetdesetih je razvoj discipline potekal brez prave organizacije, dokler se niso v 21. stoletju pojavile nacionalne smernice.

Digitalne forenzične preiskave imajo različne naloge. Najpogostejše so podpirati ali ovreči hipotezo pred kazenskimi ali civilnimi sodišči. Kriminalni primeri vključujejo domnevno kršitev zakonov, ki jih določa zakonodaja, kot so na primer umori, kraje in napadi na osebo. Civilni primeri pa se ukvarjajo z zaščito pravic in lastnine posameznikov (pogosto povezani z družinskimi spori), lahko pa se tudi ukvarjajo s pogodbenimi spori med gospodarskimi subjekti, pri katerih se vključi oblika digitalne forenzike, ki se imenuje elektronsko odkritje.

V zborniku so zbrane seminarske naloge študentov magistrskega študija na Fakulteti za računalništvo in informatiko, Univerze v Ljubljani 2019/2020. V okviru predmeta Digitalna forenzika je vsaka skupina študentov izbrala en članek, ki je služil kot izhodišče za seminarsko delo. Članki so bili izbrani iz treh raziskovalnih področij: forenzika datotečnega sistema, forenzika pomnilnika in forenzika slik, na koncu pa sledi pet člankov, ki ne spadajo v nobeno od prej omenjenih kategorij.

Na področju forenzike datotečnega sistema se seminarske naloge ukvarjajo s klesanjem datotek, manipulacijo časovnih žigov in detekcijo le-te, restavriranjem podatkov iz podatkovnih baz in z algoritmi za izračun podobnosti med datotekami.

Forenzika pomnilnika postaja vedno bolj aktualna, saj se pojavlja vedno več metod, ki so zmožne pridobiti podatke iz računalniškega spomina. Seminarski nalogi v tem sklopu zajemata analizo ostankov uporabe Tor brskalnika in uporabo strojnega učenja za zaznavanje zlonamernih zank aplikacijskih programskih vmesnikov za operacijski sistem Windows.

Forenzika slik je dandanes zelo pomembna, saj postaja manipulacija s slikovnimi datoteki vedno bolj preprosta in s tem tudi vedno bolj razširjena. Seminarske naloge se v tem delu ukvarjajo z iskanjem informacij o izvoru neke slike in zaznavanjem Deepfake videev, predvsem pa se osredotočajo na zaznavanje raznih manipulacij in ponarejanjem slik.

V sklopu razno pa so seminarske naloge, ki ne spadajo pod noben omenjen sklop. Te so se ukvarjale z napadi preko stranskih kanalov, analizo mobilnih naprav, ki so bile poškodovane z vodo, forenziko pametnih asistentov, rokovanjem z dokazi, ko so bili ti že zavrženi in prepoznavanjem spletnega nasilja preko analize besedila.

Ta zbornik združuje vse končne seminarske naloge, ki so bile izdelane v študijskem letu 2019/2020. Namenjen je vsem, ki jih zanima področje digitalne forenzike ali pa zgolj eno ali več predstavljenih področij.

Ljubljana, 2020 Nejc Povšič, Nejc Rebernik

2 Povzetki / Summaries

2.1 Sintaktično klesanje PNG slik in avtomatsko generiranje testnih primerov

Pridobivanje datotek iz medija brez uporabe datotečnega sistema se imenuje klesanje datotek. Če je datoteka na mediju shranjena zvezno je klesanje dokaj enostavno, čim pa je datoteka shranjena fragmentirano postane klesanje težavno in kompleksno opravilo. Tudi najbolj osnovni scenariji fragmentacij povzročajo težave dostopnim programom za klesanje datotek. V tem delu predstavimo algoritem za sintaktično klesanje datotek, ki za ta namen izkorišča specifičen format PNG datoteke. Algoritem uporabimo na PNG datotekah in pokažemo kako izkoristiti dani format, da čim bolj omejimo preiskovalno območje. Opišemo implementacijo takšne metode in za namen evalvacije predstavimo avtomatsko generiranje testnih podatkov.

Ključne besede / Keywords: Sintaktično klesanje datotek, PNG, generiranje testnih množic

2.2 Artefakti za detekcijo manipulacije časovnih žigov v datotečnem sistemu NTFS v operacijskem sistemu Windows in njihova zanesljivost

V seminarski nalogi raziščemo ugotovitve iz krovnega članka. Ponovimo vse poskuse manipulacije s časovnimi žigi in jih skušamo na načine opisane v članku tudi odkriti oziroma dokazati, da so se zgodili. Vse poskuse manipulacije in postopke odkrivanja le-te opišemo. Zapišemo tudi razne probleme, s katerimi smo se srečali pri ponovitvi poskusa in skušamo obrazložiti njihove vzroke. V nekaj primerih je prišlo do neskladja z rezultati članka – kar smo tudi zapisali.

Ključne besede / Keywords: časovni žigi, ponarejanje dokazov, manipulacija časovnih žigov, ntfs, dnevniki, digitalna forenzika

2.3 Restavriranje podatkov iz podatkovne baze SQLite

Seminarska naloga predstavlja pristope in koncepte restavriranja podatkovne baze SQLite. Za zgled smo uporabili članek, ki prikazuje osnovne in napredne metode restavriranja podatkov. V seminarski nalogi smo predstavili delovanje podatkovne baze SQLite, opisali metodologijo, ki jo uporablja članek in hkrati opisali možne izboljšave. Opisali in raziskali smo tudi delovanje orodja bring2lite in ga predstavili v praksi.

Ključne besede / Keywords: SQLite, forenzika, podatki

2.4 FbHash: shema za izračun podobnosti datotek v digitalni forenziki

Algoritmi za detekcijo podobnih datotek pomagajo digitalnim forenzikom pri obdelavi velikih količin podatkov. V delu predstavimo algoritem za detekcijo podobnih datotek FbHash in nekaj njegovih predhodnikov. Predstavimo in implementiramo tudi svojo različico algoritma FbHash. Implementacijo testiramo na istih množicah datotek kot avtorji članka ter predstavimo naše ugotovitve. Rezultati eksperimentov se ujemajo z rezultati v članku, saj naša implementacija algoritma doseže F-score 94%.

Ključne besede / Keywords: Prstni odtis datoteke, Podobnostni izvleček, Zabrisano zgoščevanje, TF-IDF, Kosinusna podobnost

2.5 Live and Static Analysis on Leftovers from Tor Browser Usage

The Tor Browser bundle is often used for anonymous browsing. Apart from encrypting the messages and covering user's identity in the network, Tor Browser also aims to enable the user to remove the browser without leaving any traces. In this paper, this ability was tested with two approaches; live analysis (analysis of a RAM dump remnants) and static analysis (analysis of traces left on the hard drive). Data leakage is, among other factors, dependant on the medium, where Tor Browser executable has ran from. This paper presents the results of using the browser installed on a USB drive. Nevertheless, remnants of Tor Browser usage are found with both live and static analysis.

Ključne besede / Keywords: Forensics, Tor, Tor Artifacts, live analysis, Volatility, staticanalysis, Autopsy

2.6 HookTracer: Sistem za avtomatsko analizo zank aplikacijskih programskih vmesnikov

Zlonamerna programska oprema obstaja že od skoraj začetka računalništva in se skupaj z njim razvija. Odločili smo se, da bomo prebrali in poskusili implementirati rešitve članka, ki se ukvarja s programsko opremo, ki cilja t. i. API zanke (ang. "API hooks") in v nekaterih primerih teče v samem spominu. Avtorji so v članku predstavili uporabo forenzike računalniškega spomina za zaznavo in analizo zlonamerne programske opreme ter predstavili način, kako bi lahko hitro in učinkovito ločili zlonamerne API zanke od legitimnih. Ustvarili so vtičnik hooktracer za orodje volatility, ki vse to omogoča, hkrati pa je hitrejši in zmogljivejši od že obstoječih rešitev. Vtičnik smo želeli preizkusiti tudi sami, a na žalost ni še na voljo za uporabo.

Ključne besede / Keywords: API Hooks, Memory Forensics, Volatility

2.7 Iskanje izvora JPEG slik na podlagi informacij v glavi

Informacija o napravi oz. tipu naprave s katero je bila zajeta ali ustvarjena neka slika je lahko pomemben podatek v forenzični preiskavi. V članku se ukvarjamo z možnimi načini za pridobitev tega podatka za slike formata JPEG na podlagi podatkov, zapisanih v glavi slike. Na podlagi eksperimentov, ki so jih izvedli P. Mullan in sodelavci, pokažemo s kakšno natančnostjo je možno določiti znamko naprave za nek predhodno neznan model. Eksperimenti so bili izvedeni tudi na slikah, ki so bile predhodno obdelane z neko programsko opremo. Pričakovano se izkaže, da je natančnost v tem primeru precej slabša.

Ključne besede / Keywords: forenzika slik, JPEG format, JPEG meta-podatki, glava slike

2.8 Copy-move Forgery Detection in Digital Images

Copy-move forgery is a method of concealing or manipulating information of an image, done by copying and pasting an area of the same image. Commercialization and ease of use of the tools that enable such processes has made image tempering more widespread, whilst making detecting such forgeries harder for the human eye. In this paper, we describe the methods that are used in detecting copy-move forgery and provide a more detailed overview of the DCT, SVD, SURF based methods.

Ključne besede / Keywords: Copy-move forgery detection, Image forgery detection, Image forensics, DCT, SURF, SVD

2.9 Detecting Copy-Move Image Forgery

This paper approaches a deep neural architecture for image copy-move forgery detection (CMFD), a deep neural network solution to localize potential image manipulation via visual artifacts, copy-move regions and visual similarities. Outperforming the state-of-the-art copy-move detection algorithms on several data sets, BusterNet can be used against some of the most common crimes today, such as fake news, manipulation of surveillance images, among many others. There are different 'weapons' that are used to discover a forged image and deep convolution learning algorithms are one of them.

Ključne besede / Keywords: Copy-Move, Image Forgery Detection, Deep Learning

2.10 Stvaritev in zaznavanje Deepfake videoposnetkov z globokim učenjem

Razvoj metod globokega učenja nam je omogočil reševanje kompleksnih, večdimenzionalnih problemov na področju digitalnih medijev in računalniškega vida. Hkrati pa je privedel do nastanka tehnologij, ki predstavljajo grožnjo za zasebnost, varnost in zaupnost. Deepfake tehnologija je ena od bolj izpostavljenih; lahko ustvari ponarejene slike in posnetke ljudi, ki so zelo težko ločljivi od pristnih. Potreba po zanesljivih metodah preverjanja pristnosti takih medijev narašča, zato bomo v tem prispevku predstavili izzive, s katerimi se digitalni forenziki soočajo pri tem, ter naredili pregled predlaganih modernih metod za samodejno prepoznavanje ponaredkov.

Ključne besede / Keywords: deep learning, deepfakes, computer vision, forensics, digital forensics, audiovideo manipulation

2.11 Learning Rich Features for Image Manipulation Detection review

This document describes a novel method for image manipu- lation detection proposed by Zhou et al.. The described method uses a faster region-based convolutional neural network (R-CNN) for learning rich features from images that have clues of image manipulation. To achieve better performance it uses two streams. The first is an RGB stream and the second is a noise stream, which is derived from the RGB stream with the help of steganalysis rich model (SRM) filters. The RGB stream is used to find strong contrast differences, unnatural boundaries and so on. At the end of these streams, bilinear pooling is used for joining the two streams together. After presenting and describing this novel method, the paper describes experiments made by the authors of the novel method. These experiments show that using both streams gives better results than just using an individual stream.

Ključne besede / Keywords: image forensics, image manipulation detection, faster R-CNN, deep learning

2.12 Leveraging Electromagnetic Side-Channel Analysis for the Investigation of IoT Devices

The area of IoT (Internet of Things) devices is growing by day and they are becoming a vital part of our lives. Consequently, we are more likely to find them at a crime scene, and analysing them could provide insight into the events that unfolded. The article proposes a novel method of analysing such devices and extracting relevant data without destroying the device. The authors suggest that these methods might be part of existing digital investigation flows.

Ključne besede / Keywords: IoT, Side-Channel, Electromagnetic Analysis

2.13 Forensic analysis of water damaged mobile devices

Our lives are becoming more and more entangled with mobile electronic devices. Increased usage can lead to various types of physical damages that can occur. These can happen due to the negligence of their owners or in an effort of destroying crime evidence. In any case, we are interested in retrieving relevant data from such a device. Naturally, this is an important part of any digital forensic investigation. In an effort of destroying crime evidence, electronic devices are often destroyed by submerging in a liquid. To conduct a successful restoration of a device to an operating state it is important to understand electrochemical reactions that happen when a device is exposed to water. In this paper, we analyze the effects of moisture in combination with device electronics, discuss common repair methods and related works in this field.

Ključne besede / Keywords: digital forensics, water damage, metal corrosion, mobile devices

2.14 Digitalne forenzične prakse in metodologije za pametne asistente

Pametni asistenti so lahko pomemben vir podatkov pri digitalni preiskavi, saj so običajno ves čas vklopljeni in čakajo na naš ukaz. Gre za naprave, ki so del tako oblačnih storitev, kot tudi interneta stvari. Pri njihovi preiskavi uporabimo pet metod. V vsaki izmed njih uporabljamo specifični pristop in orodja, kot rezultat pa dobimo nekatere podatke. Pri analiza paketov pametnih asistentov analiziramo HTTP pakete med asistentom ter oblakom, med tem ko pri analiza paketov Android mobilnih aplikacij analiziramo HTTPS pakete med Android mobilno aplikacijo ter oblakom. Ko analiziramo datotečni imenik, analiziramo podatke aplikacije, ki se običajno nahajajo v "/data/data". Ena izmed metod je tudi dekompilacija aplikacijskih paketov, kjer se ukvarjamo z analizo programske kode. Analiza odrezkov poteka na internem spominskem čipu asistenta. Pri analizi uporabimo več različnih programov, pri pametnem asistentu NAVER Clova pa si lahko pomagamo tudi z aplikacijo, ki so jo razvili avtorji originalnega članka.

Ključne besede / Keywords: digitalna forenzika, pametni asistent, internet stvari (IoT), oblačne storitve, analiza

2.15 Nedovoljeni digitalni dokazi in zaščita zasebnosti v sistemih verige blokov

Članek, ki sva ga prebrala se osredotoča na velikokrat pozabljen aspekt rokovanja z dokazi, to je ko sodišče zavrže dokaze. Več razlogov je zaradi katerih so dokazi lahko zavrženi. Lahko so zavrženi že med preiskavo, pred sodiščem, ali pa jih opusti tožilec. Ker v generični implementaciji verige blokov kasnejše spremembe niso mogoče, je v članku opisana rešitev katera uporablja dve verigi blokov za vodenje verige skrbništva. S tem je omogočeno razveljevanje transakcij, ki se navezujejo na dokaze.

Ključne besede / Keywords: veriga skrbništva, dokazi, veriga blokov, transakcije

2.16 Detecting Cyberbullying "Hotspots" on Twitter: A Predictive Analytics Approach

In this article, we describe the work done in the field of detecting cyberbullying from online texts from the article by Ho et al.. Furthermore we introduce methods and available software by explaining what cyberbullying hotspots are and how we can computationally find them. We also describe other approaches and work in the field of detecting cyberbullying. Lastly, we implement our own classifiers and try to computationally infer whether tweets are cyberbullying orientated ourselves and discuss the results and possible future applications of these methods.

Ključne besede / Keywords: cyberbullying, field review, predictive analysis, logistic regression, text mining, social media, Formspring, Twitter

3 Forenzika datotečnega sistema / File system forensics

Sintaktično klesanje PNG slik in avtomatsko generiranje testnih primerov

Jakob Bevc
Fakulteta za računalništvo in informatiko, Večna pot 113
Ljubljana, Slovenija
jb2285@student.uni-lj.si

Matej Kristan
Fakulteta za računalništvo in informatiko, Večna pot 113
Ljubljana, Slovenija
mk9566@student.uni-lj.si

Lojze Žust
Fakulteta za računalništvo in informatiko, Večna pot 113
Ljubljana, Slovenija
lojze.zust@student.uni-lj.si

POVZETEK

Pridobivanje datotek iz medija brez uporabe datotečnega sistema se imenuje klesanje datotek. Če je datoteka na mediju shranjena zvezno je klesanje dokaj enostavno, čim pa je datoteka shranjena fragmentirano postane klesanje težavno in kompleksno opravilo. Tudi najbolj osnovni scenariji fragmentacij povzročajo težave dostopnim programom za klesanje datotek.

V tem delu predstavimo algoritem za sintaktično klesanje datotek, ki za ta namen izkorišča specifičen format PNG datoteke. Algoritem uporabimo na PNG datotekah in pokažemo kako izkoristiti dani format, da čim bolj omejimo preiskovalno območje. Opišemo implementacijo takšne metode in za namen evalvacije predstavimo avtomatsko generiranje testnih podatkov.

Ključne besede

Sintaktično klesanje datotek, PNG, generiranje testnih množic

1. UVOD

Velik del forenzične preiskave temelji na zajemu podatkov s pomnilnega medija. Z analiziranjem datotečnega sistema na mediju, preiskovalci zajamejo podatke, ki so lahko ključni v preiskavi. S podrobno analizo se lahko dokopljejo tudi do podatkov, ki so bili izbrisani. Vendar to ni vedno mogoče zaradi več morebitnih razlogov. Morda zaradi manjkajočih zapisov v datotečnem sistemu ali pa zaradi manjkajočih zapisov, ki tvorijo podatke, lahko pa je kriva tudi mehanična poškodba medija.

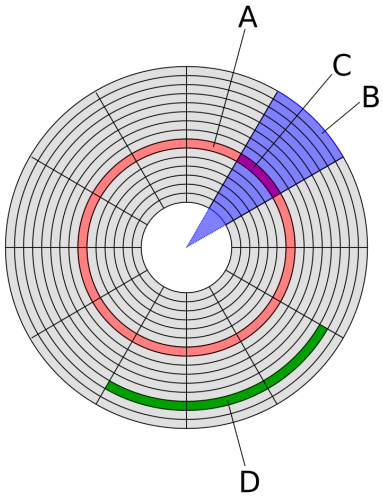
Pridobivanju podatkov iz medija, o katerih nimamo dodatne informacije, kje na mediju se nahajajo, rečemo klesanje. Podatki so na mediju shranjeni v datotekah, posamezna datoteka pa je shranjena v sektorjih. Na Sliki 1 vidimo logično strukturo pomnilnega medija. Če predpostavimo, da si sektorji, ki hranijo datoteko sledijo zaporedno, klesanje poteka od začetka proti koncu datoteke, oziroma od niza bajtov, ki predstavlja začetek do niza, ki predstavlja konec datoteke. Čim pa je datoteka shranjena v sektorjih, ki niso zaporedni ampak so razpršeni po mediju, postane takšno klesanje nezanesljivo.

V članku [3] so avtorji naredili analizo klesanja datotek in navajajo, da je avtomatsko klesanje fragmentiranih datotek pomembno tudi za digitalno forenziko. Opišejo, da je metoda surove sile, zaradi vse večjih pomnilnih medijev, časovno prezahtevna in kako čim bolj skrčiti preiskovalno območje fragmentov. Avtomatsko klesanje pa ostaja daleč od enostavne naloge. Najprej je potrebno odkriti začetke in konce posameznih fragmentov, nato pa jih urediti v smiselni vrstni red.

V nadaljevanju opišemo kako izkoristiti format datoteke, pri klesanju in si pogledamo klesanje slikovnih datotek shranjenih v formatu PNG. Izbrani format ima zelo ugodno strukturo za klesanje saj ga tvori več kosov, ki si sledijo v urejenem vrstnem redu. PNG standard ima dokumentiranih 26 različnih kosov [8], ki imajo točno določeno strukturo. Na začetku vsakega kosa se nahaja predpisano zaporedje bajtov, ki ga imenujemo podpis. Ta določa vrsto kosa in s tem tudi njegovo strukturo.

Avtorji [5] opišejo postopek klesanja v več zaporednih fazah, razlog za to je, da je velika večina datotek shranjenih v enem fragmentu. V prvi fazi odkrijemo vse datoteke v enem fragmentu in posledično izločimo uporabljene fragmente, ki jih v naslednjih, časovno bolj zahtevnih, fazah ne rabimo upoštevati. V drugi fazi odkrivamo datoteke, ki so shranjene v dveh fragmentih in tako naprej.

Za evalvacijo opisanega pristopa potrebujemo kopijo diska, ki vsebujejo fragmentirane PNG slike. Na spletu obstaja več tovrstnih testnih množic, med katerimi pa nobena ne vsebuje PNG slik. Zaradi tega avtorji [5] razvijejo ogrodje za avtomatsko generiranja testnih množic, ki ga uporabijo za evalvacijo. Predlagajo, da bi z razvitim generatorjem lahko primerjali delovanje različnih programov za klesanje PNG datotek. Predlagana metoda klesanja naj bi delovala bolje kot popularni orodji Scalpel [7] in PhotoRec [2]. Generator testnih množic pa je dostopen preko spleta [6].



Slika 1: Logična struktura pomnilnega medija [9]. Črke označujejo različne dele in sicer A označuje sled, B označuje geometrični sektor, C označuje sektor in D gručo.

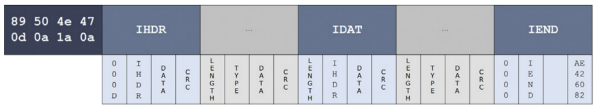
V prvem delu tega članka opišemo osnovno strukturo formata PNG. V drugem delu nato povzamemo delovanje algoritma za sintaktično klesanje PNG datotek, ki upošteva to strukturo. V zadnjem delu pa predstavimo generator evalvacijskih podatkovnih množic in podamo evalvacijo sintaktične metode za klesanje.

2. FORMAT PNG

PNG je okrajšava angleškega izraza Portable Network Graphics ali po slovensko prenosljiva spletna grafika. Iz imena lahko sklepamo, da je namenjen prenosu slik po spletu, njegova uporaba pa se je razširila na zelo različna področja. Prva verzija formata PNG je izšla leta 1996, predvsem zato ker v formatu GIF naletimo na omejitev 256 barv in tudi zaradi najave Unisys, ki je zahtevala patent za algoritem brez izgubnega stiskanja ki se uporablja v formatu GIF.

Strukture PNG datotek ne bomo opisovali podrobno s stališča predstavitve slik ampak si bomo pogledali pomembnejše lastnosti, ki olajšajo klesanje. Datoteka PNG se vedno začne s PNG podpisom, ki sestoji iz 64 bitov. V šestnajstiškem zapisu ga predstavimo kot zaporedje osmih bajtov: `89 50 4e 47 0d 0a 1a 0a`. PNG podpisu sledi telo, ki ga sestavljajo različni kosi. Vsak kos sestavljajo 4 polja. Prvo polje so štirje bajti, ki predstavljajo dolžino kosa. Sledi podpis iz štirih bajtov, ki definira tip kosa, nato so podatki, ki so spremenljive dolžine in na koncu sledijo štirje bajti, ki hranijo CRC vsoto.

Dolžina kosa je shranjena kot ne predznačeno celo število in predstavlja dolžino podatkovnega polja v bajtih. Dolžina je lahko tudi nič, kar pomeni, da kos nima podatkovnega polja. Na tem mestu poudarimo, da je dolžina celotnega kosa enaka številu bajtov v podatkovnem polju plus 12 bajtov, ki predstavljajo dolžino, podpis in CRC vsoto kosa.



Slika 2: Prikazuje obvezne kose in njihov vrstni red v PNG formatu.

Sledi podpis iz štirih bajtov, ki kodirajo štiri črke, v ISO 646 standardu, in definira tip kosa. Za podpisom sledijo podatki iz toliko bajtov kot predpisuje dolžina kosa, ki se nahaja na začetku posameznega kosa. Za podatki pa se nahajajo štirje bajti, ki predstavljajo CRC vsoto. Vsota je namenjena temu, da potrdimo integriteto podatkov. Omenimo, da je CRC vsota izračunana na štirih bajtih, ki predstavljajo podpis in na bajtih, ki predstavljajo podatke.

PNG standard definira tudi vrstni red v katerem si morajo posamezni kosi slediti. Obstajata dva glavna tipa kosov in sicer obvezni in opcijski kosi. Obvezni kosi (glej Sliko 2) so IHDR, IDAT in IEND, kar pomeni da te kose vsebuje vsaka PNG datoteka. V datoteki se nahajata natanko en IHDR kos (glava PNG) in IEND kos (konec PNG) kos, ter poljubno število IDAT kosov (sama slika, razdeljena na več delov). Razdelitev slike na več IDAT kosov omogoča postopno prenašanje in prikazovanje slike prek spleta, hkrati pa jih lahko izkoristimo za bolj robustno klesanje, ki ga opišemo v nadaljevanju.

3. OBSTOJEČI PRISTOPI

Specifična struktura PNG datotek odpira zanimive možnosti za bolj robustno klesanje datotek tega formata. Pred nastankom predstavljene metode, nobeden izmed obstoječih pristopov ni v večji meri izkoriščal strukture formata. Večinoma so se uporabljali splošni pristopi, ki se uporabljajo pri klesanju vseh formatov. Najbolj očiten pristop je tako imenovano klesanje med glavo in nogo. Pri tem pristopu iščemo specifična zaporedja bajtov, ki označujejo začetke in konce določenih datotek, da najdemo začetek in konec datoteke. Daljši kot je ta podpis tipa datoteke, bolj natančna je metoda. Videli smo, da ima PNG format na začetku datoteke fiksen podpis dolžine 8 bajtov. Zaradi fiksne strukture kosa IHDR, lahko za podpis uporabimo tudi polji dolžina in tip. Skupaj tako dobimo začetni podpis dolžine 16 bajtov. Podatki v kosu IEND so prazni, zato ga lahko uporabimo za zaključni podpis datoteke. Tak princip uporablja odprto kodno orodje Scalpel . Očitno je, da tako klesanje odpove v primeru fragmentiranih datotek.

Podoben pristop je uporaba informacije o dolžini datoteke, ki se nahaja v glavi. Tu konec datoteke najdemo tako, da začetni poziciji prištejemo ustrezen odmik, ki ga preberemo iz glave. PNG format ne vsebuje globalnega podatka o dolžini datoteke, zato pa vsak posamezen kos vsebuje informacijo o lastni dolžini. Nekateri algoritmi [1, 2] uporabljajo prilagojeno različico te metode. Izvajamo zaporedne skoke, dokler ne pridemo do konca ali neveljavnega dela. Tako lahko obnovimo vsaj del PNG datoteke do prve točke fragmentacije.

Nekateri pristopi so prilagojeni tudi za primer fragmentacije. Eden izmed prvih pristopov na tem področju je klesanje bifragmentiranih datotek z vrzeljo. Ta pristop skuša ugotoviti

vrzel med začetnim in končnim delom datoteke. To stori s premikanjem vrzeli in njenim postopnim zmanjševanjem. Veljavnost pozicije vrzeli preverja s hitrimi testi veljavnosti datoteke, ki so specifični za posamezne tipe datotek. Ta pristop deluje solidno za primere, kjer je datoteka razdeljena na dva dela in je vrzel med njima dovolj majhna. Algoritem ne deluje na datotekah, ki so fragmentirane na več kot dva dela.

Še zadnja skupina pristopov pa deluje na podlagi dejanske vsebine datotek. Pri teh metodah lahko na primer spremljamo vrednosti pikslov pri JPEG in tako določimo točke fragmentacije. Potrebno je dekodiranje in dekompresija podatkov v delu datoteke. Gre za podoben pristop kot pri zgoraj omenjeni bifragmentaciji, le da za posamezne točke fragmentacije dobimo verjetnosti namesto binarne veljavnosti.

Nekatere metode torej izkoriščajo generalne pristope, ki delujejo na vseh datotekah, druge pa delujejo na dejanskih podatkih, kar je zahteven in počasen proces. Nobena od metod pa v polnosti ne izkorišča specifik PNG formata. Poleg tega je večina metod sposobna zgolj zaznati točke fragmentacije, ne pa jih tudi popraviti. Struktura formata omogoča bolj robustno detekcijo in rekonstrukcijo fragmentirane datoteke od nekaterih ostalih formatov. V predstavljeni metodi te lastnosti smiselno izkoristijo za hitrejše in bolj robustno klesanje PNG datotek.

4. SINTAKTIČNO KLESANJE PNG SLIK

Klesanje PNG datotek se izvaja v več fazah – najprej se uporabijo enostavnejši pristopi, ki detektirajo enostavne datotek, potem pa se kompleksnost stopnjuje, da zaznamo tudi bolj zahtevne fragmentacijske scenarije. S takim hierarhičnim pristopom na začetku polovimo veliko število enostavnih primerov in posledično tudi zmanjšamo velikost problema nadaljnih korakov, saj tega dela diska ni potrebno več pregledovati.

4.1 Iskanje podpisa

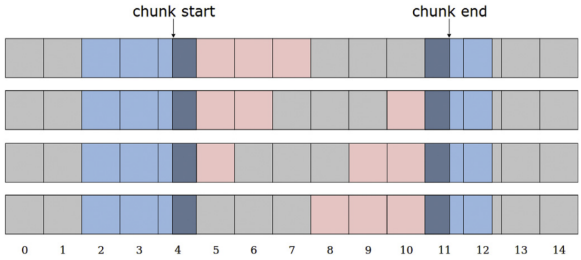
V tej fazi poiščemo vse začetne podpise PNG datotek. Poleg tega iščemo tudi začetke posameznih kosov, tako da poskušamo najti začetne podpise vseh 26 različnih tipov kosov definiranih v PNG specifikaciji. Vsaka PNG datoteka ima natanko en IHDR kos takoj za podpisom, zato preverimo tudi če se število začetnih podpisov ujema s številom IHDR kosov.

4.2 Zaporedni skoki

V tej fazi detektiramo nefragmenitrana zaporedja kosov. Klesanje se začne pri podpisu in skače od kosa do kosa. Iz glave posameznega kosa preberemo njegovo dolžino in naredimo ustrezen skok na pričakovano pozicijo konca kosa. To počnemo, dokler je skok veljaven, ali pa dosežemo končni delček IEND. Da je skok veljaven, mora po skoku veljati naslednje:

1. Naslednja pozicija predstavlja začetek veljavnega PNG kosa.

2. Vrstni red tipa kosov mora biti skladen s PNG specifikacijo.



Slika 3: Primer generiranja bifragmentiranih blokov za tri bloke. Bloki se dodajajo na konec ene strani in odstranjujejo z druge, dokler ne najdemo ustrezne kombinacije.

3. CRC na koncu kosa se mora ujemati s CRC-jem izračunanim na preskočenih podatkih.

Nefragmentirane datoteke, kjer s tem postopkom uspešno pridemo do končnega IEND delčka, lahko v celoti izklešemo. V vseh ostalih primerih, odkrita zaporedja preidejo v drugo fazo, kjer skušamo odkriti vrzeli med pari zaporedij.

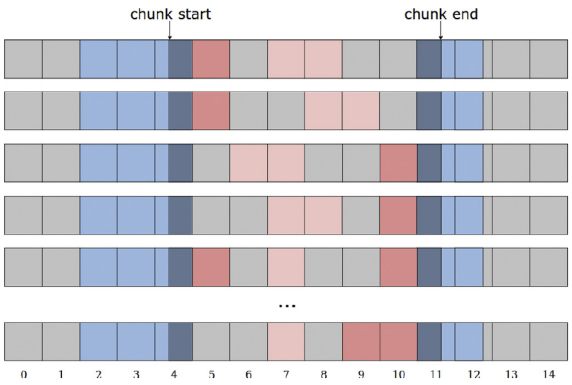
4.3 Generator bifragmentiranih blokov

V primeru neuspešnega skoka je prišlo do fragmentacije znotraj trenutnega kosa. V tem koraku predpostavljamo, da je kos razpadel na dva dela, kar velja za večino fragmentiranih kosov. Za vsak fragmentiran kos želimo poiskati naslednji zaporedni kos v datoteki. Število kandidatov lahko bistveno zmanjšamo z upoštevanjem strukture diska. Ker je velikost blokov fiksna in imamo podatek o dolžini kosa, poznamo pričakovan odmik začetka naslednjega kosa znotraj bloka. Kose, ki nimajo ustreznega odmika izločimo. Pri zmanjševanju števila kandidatov upoštevamo tudi omejitve dovoljenih vrstnih redov tipov v PNG specifikaciji.

Iz podatka o dolžini kosa, lahko natančno določimo število potrebnih blokov. Za preostale kandidate lahko nato generiramo možne kombinacije blokov, tako da povečujemo število blokov za začetkom delčka in ustrezno zmanjšujemo število blokov pred kandidatom, da ohranjamo skupno število blokov (glej Sliko 3). Pri tem ignoriramo bloke, ki smo jih v prejšnjih fazah že izklesali. Vsako konfiguracijo preverimo z uporabo CRC na koncu kosa. Če se CRC ujema, potem smo našli ustrezno konfiguracijo in lahko uspešno povežemo kose. Tak postopek deluje tudi v primeru, da fragmentirana dela nista v pravilnem zaporedju. Ker je pomembna le fragmentacija znotraj delčka, tak postopek deluje tudi na datotekah, ki so fragmentirane v več delov, če le do večkratne fragmentacije ne pride znotraj istega kosa.

4.4 Generiranje blokov z drsečim oknom

V primeru fragmentacije kosa na več kot dva dela, prejšnja faza odpove. V takem primeru vsaj eden izmed fragmentov ne vsebuje nikakršnega podpisa, kar onemogoča njegovo detekcijo. Še vedno pa poznamo število blokov, ki jih kos zaseda, s CRC-jem pa lahko preverimo ustreznost kombinacije blokov. V tej fazi se ustvari okno v velikosti pričakovanega števila blokov. Okno se nato pomika od začetka kosa do kandidata. Okno predstavlja nekakšen fragmentni otok, ki



Slika 4: Primer delovanja generiranja blokov z drsečim oknom za tri bloke. S svetlo roza barvo so označeni bloki v oknu, s temno roza pa fiksni bloku na začetku in koncu.

ga dve vrzeli ločujeta od začetnega in končnega dela kosa. Metodo okna kombiniramo z metodo dodajanja blokov na za začetek in konec iz prejšnje faze. To storimo tako, da okno postopno zmanjšujemo, preostale bloke pa razporejamo na začetek in konec kosa (glej Sliko 4).

S tem pristopom smo sposobni detektirati fragmentacijo kosov, ki razpadejo na tri dele. V primeru, da tudi ta metoda ne najde ustrezne konfiguracije, se izvede brute-force metoda, ki preveri vse možne konfiguracije blokov med začetkom in koncem. Pri tem seveda ohranja znano fiksno število potrebnih blokov. V evalvaciji brute-force metode niso uporabili.

4.5 Poškodovane PNG datoteke

V primeru, da ne najdemo nobene ustrezne kombinacije blokov, ali pa je število preostalih blokov manjše od potrebnega števila, se PNG smatra za poškodovanega. V takem primeru izklešemo le del, ki smo ga uspešno validirali. Sekvenčnost PNG formata namreč omogoča, da v takem primeru lahko prikažemo vsaj del originalne slike.

4.6 Optimizacija

Število kombinacij, ki jih morajo faze preveriti, bistveno vpliva na učinkovitost algoritma. Omenili smo že zmanjševanje števila kandidatov z upoštevanjem pričakovanega odmika in zaporedja tipov. Pri tem veliko vlogo igrajo tudi uspešne detekcije. Ko uspešno izklešemo določeno datoteko, vse bloke označimo za zasedene, ter tako zmanjšamo iskalni prostor nadaljnjih metod. Poleg tega v naprednejših fazah (od drsečega okna naprej) še enkrat poženemo začetne faze. Odkrite datoteke so namreč lahko spremenile zasedenost blokov do te mere, da enostavnejše metode sedaj znova delujejo. Poleg tega, opisanih metod ne poženemo od začetnih PNG podpisov naprej, ampak tudi na vseh najdenih zečetkih delčkov. Tako hitreje označimo bloke kot zasedene in zmanjšamo iskalni prostor. Poleg tega se procesi lahko izvajajo vzporedno, saj delamo na posameznih delčkih in ne potrebujemo podatkov iz že validiranih delov datoteke.

5. PODATKOVNE MNOŽICE ZA KLESANJE DATOTEK

V preteklosti je bilo razvitih več algoritmov za klesanje datotek pri katerih pa se pojavlja vprašanje o enotni evalvaciji uspešnosti posameznih algoritmov. Na celotnem področju raziskav digitalne forenzike je le nekaj javno dostopnih podatkovnih množic [4], ki omogočajo kvalitetno primerjavo obstoječih pristopov klesanja datotek. Zaradi omenjenih razlogov avtorji predlagajo enotno generiranje podatkovnih množic na katerih nato izvedejo primerjavo obstoječih pristopov in njihovega algoritma.

Za vrednotenje algoritmov morajo testne podatkovne množice vsebovati različne primere fragmentacij datotek in napak v samih podatkih. Za osnovo pri generiranju so bili uporabljeni primeri iz DFRWS izzivov v letih od 2006 do 2007. DFRWS so vsakoletni globalni izzivi na področju digitalne forenzike. Pogosto vsebujejo tudi naloge iz področja klesanje datotek, vendar do sedaj nobeden izmed teh ni vseboval PNG datotek, zato so avtorji implementirali ogrodje za avtomatsko generiranje testnih množic s PNG datotekami.

5.1 Avtomatsko generiranje podatkovnih množic za klesanje datotek

Fragmenti so osnovni elementi uporabljeni v generatorju. Delimo jih na dve vrsti: *Datotečni Fragment*, ki vsebuje dejanske podatke za klesanje in *Polnilni Fragment*, ki je ročno definiran fragment z naključnimi podatki. Na obeh vrstah fragmentov lahko izvajamo naslednje operacije:

- **Delitev**: Nove fragmente lahko kreiramo z delitvijo obstoječih fragmentov. To naredimo tako, da fragmente naključno razdelimo na določeno število novih fragmentov ali pa po prej definiranem pravilu razdelimo fragmente na več novih. Operacija je rekurzivna. Vsako deljenje je izvedeno na podlagi velikosti bloka na disku.

- **Brisanje** je naslednja metoda s katero lahko spreminjamo fragmente, tako da izbrišemo bodisi začetek ali konec izbranega fragmenta. Velikost izbrisanega dela definiramo v bajtih.

V naslednjem koraku so fragmenti združeni v **Scenarije**. Ti delujejo kot ovojnica okoli množice fragmentov in omogočajo izvajanje različnih operacij nad množico fragmentov:

- **Dodajanje** je najpreprostejša operacija, ki omogoča dodajanje novega fragmenta k scenariju.

- **Prepletanje** je operacija, ki omogoča dodajanje novih fragmentov kot alternirajočo vrsto med obstoječe fragmente.

- **Mešanje**, **Obračanje**, **Spreminjanje vrstnega reda**, so operacije, ki omogočajo spreminjanje vrstnega reda na več različnih načinov. Prvi je naključno mešanje fragmentov, drugi je spreminjanje smeri v kateri nastopajo kosi in tretji omogoči spreminjanje vrstnega reda.

V zadnjem koraku je več **scenarijev** dodanih v **Sliko diska**, ki hrani informacije o pripravi scenarijev in nato zapiše podatke na disk.

5.2 Rekreiranje DFRWS izzivov

Omenjeni izzivi že v osnovi pokrivajo veliko različnih scenarijev za ocenjevanje uspešnosti algoritmov za klesanje podatkov. Njihova glavna pomanjkljivost je, da ne vsebujejo PNG slik, zato so se avtorji odločili, da uporabijo prej definirane scenarije vendar za datoteke tipa PNG. Karakteristike, ki definirajo različne scenarije DFRWS izzivov lahko delimo na štiri kategorije:

- **Datoteke** Izbrane datoteke in njihovo število za določen scenarij. Maksimalno 4 datoteke za posamezen scenarij.
- **Fragmenti** Datoteke so razdeljene v določeno *število* fragmentov. Točke na katerih se izvede deljenje datoteke se določi naključno. Vsaka datoteka ima maksimalno 4 fragmente.
- **Zaporedje** Kreirani fragmenti so bili urejeni v določeno zaporedje. To vključuje tudi **Obračanje** in **Prepletanje**, ki sta posebna načina urejanja zaporedja fragmentov.
- **Polnila** Nekateri scenariji vsebujejo polnila (Polnilni Fragmenti) med ostalimi fragmenti. Vsebina teh je naključna. Vsi delčki so zapolnjeni od velikost bloka (512 bajtov). Tudi to polnilo sodi k tej kategoriji.

Na podlagi specifikacij podanih na DFSRW izzivih so avtorji kreirali več različnih scenarijev, ki vsebujejo PNG slike. Slike so bile naključno vzorčene iz zbirke slik, katera vsebuje slike različnih velikosti, strukture in vsebine. V nekaterih scenarijih so tudi slike, ki nimajo začetnega ali končnega dela, za katere že vnaprej vemo, da jih nebo mogoče prikazati. Vsi kreirani scenariji so predstavljeni na Sliki 5.

6. EVALVACIJA

Avtorji primerjajo svojo implementacijo sintaktičnega klesalnika PNG datotek z obstoječimi prosto dostopnimi klesalnik, kot so Scalpel [7], Foremost [1] in PhotoRec [2]. V posameznih od naštetih so bile najdene določene pomanjkljivosti npr.: napačna glava PNG datoteke pri Scalpel-u. Prav tako je bilo potrebno, kjer je bilo to mogoče, povečati velikost pričakovane datoteke, saj so v naboru PNG slike velikosti do 200 MB.

Za vsakega od predlaganih testnih scenarijev, se naredi 50 instanc, kjer je vsaka od teh unikatna glede na točke delitve PNG datotek. Na ta način postane evalvacija vseh pristopov bolj robustna. Vsi klesalniki so bili testirani na vsakem scenariju, z časovno omejitvijo ene ure na eno instanco scenarija.

Rezultati (izklesane datoteke) so bili ročno preverjeni in razdeljeni v štiri kategorije glede na kvaliteto izklesane datoteke.

- **Popolne datoteke**: PNG datoteka je bila v celoti izklesana, brez manjkajočih podatkov.
- **Manjše napake**: PNG datoteka, ki je lahko pregledana z urejevalnikom slik in ima pravilnih vsaj 50% pikslov.
- **Velike napake**: PNG datoteka, ki je lahko pregledana z urejevalnikom slik in ima < 50 % vsebine napačne.
- **Pokvarjene/manjkajoče** PNG datoteka, ki ne more biti pregledana z urejevalnikom slik in slike ki jih klesalnik sploh ni izklesal.

Iz rezultatov (glej Sliko 6) lahko razberemo, da nobeden izmed klesalnikov ni izklesal slike, ki dejansko tudi ne obstaja. Izkaže se, da predlagan algoritem sintaktičnega klesanja uspe popolnoma izklesati vse datoteke v 20 od 26 scenarijev. Potrebno je omeniti, da je med slikami, ki niso bile uspešno izklesane večina takih, ki imajo *IDAT* 'chunk' razdeljen v več fragmentov. To je bila tudi naša prvotna opazka pri branju članka, kar bomo bolj natančno omenili v zaključku. Pri takem dogodku dobimo vsaj en delček, ki ne vsebuje nobene sintakse, ki bi bila uporabna za predlagan algoritem. Edina možnost v takem primeru je preizkus vseh mogočih kombinacij, ki se skladajo po velikosti kar pa je časovno zelo potratno.

Ostali klesalniki so se po pričakovanjih dobro odrezali na primerih kjer so bili podatki shranjeni zaporedno, vendar odpovedo pri preprostih deljenjih na delčke (Slika 6). Avtorji omenjajo tudi slabe rezultate pridobljen z Foremost-om in PhotoRec-om, velja omeniti, da slab rezultat deloma izvira tudi iz velikosti PNG slik, saj Foremost uporablja omejitev na 3000 x 3000 pikslov. Iz tega stališča je eksperiment deloma pristranski, saj na ta način omejuje Foremost algoritem z nadpovprečno velikimi slikami. Slednje ne pomeni, da je algoritem tako slab, kakor nakazujejo rezultati na Sliki 6 (že za najbolj osnovne primere, z zaporedno hranjenimi podatki).

7. RAZŠIRITEV METODE NA OSTALE FORMATE

Opisana metoda s pridom izkorišča strukturo PNG datotek in omogoča robustno klesanje kompleksnih scenarijev fragmentacije, kar je razvidno v evalvaciji. V tem delu želimo preveriti, ali bi lahko razvito metodo prilagodili in jo uporabili tudi za klesanje datotek drugega formata. Da je format primeren za uporabo metode, mora zadostiti dvema pomembnima lasnostima formata PNG, ki jih opisana metoda klesanja izkorišča:

- razdelitev podatkov na **več kosov**
- metapodatki kosa, ki vsebujejo **podatek o dolžini kosa** in **podatek za preverjanje pravilnosti vsebine** (npr. CRC).

Razbitje datoteke na večje število manjših kosov olajša proces klesanja. Več kosov namreč vsebuje več sintaktične informacije, ki jo izkoriščamo za klesanje, poleg tega pa se zaradi manjših kosov povprečno število fragmentov znotraj posameznega kosa bistveno zmanjša. Pri slikovnih formatih daleč največji del datoteke predstavlja vsebina slike, zato je

Scenario	PNGs	Fragments	Description	Order
1	1	1	non fragmented	1
2	1	2	non-sequential	1b 1a
3	1	2	with filler	1a FILLER 1b
4	2	2	with PNG in between	1a 2 1b
5	1	3	with filler	1a FILLER 1b FILLER 1c
6	1	3	non-sequential	1a 1c 1b
7	1	3	non-sequential	1b 1a 1c
8	1	3	non-sequential	1b 1c 1a
9	1	3	non-sequential	1c 1a 1b
10	1	3	non-sequential	1c 1b 1a
11	1	4	non-sequential	1a 1c 1b 1d
12	1†	2	missing end	1a
13	1†	3	missing middle	1a 1c
14	1†	3	missing end	1a FILLER 1b
15	1†	3	missing middle, non-sequential	1c 1a
16	1†	3	missing end, non-sequential	1b 1a
17	2	2	intertwined, non-sequential	1b 2b 1a 2a
18	2†	2	intertwined, 2nd missing end	1a 2a 1b
19	2†	2	1st missing end, 2nd missing start, continuous	1a 2b
20	2†	2	1st missing end, 2nd missing start, with filler	1a FILLER 2b
21	2†	2	1st missing end, 2nd out of order	1a 2b 2a
22	2†	2	2nd missing start and in between 1st fragments	1a 2b 1b
23	2†	2	1st missing end, with filler	1a 2a FILLER 2b
24	2	*	intertwined, sequential	*
25	3	*	intertwined, sequential	*
26	4	*	intertwined, sequential	*

Slika 5: Različni scenariji za evalvacijo algoritmov. V zadnjem stolpcu je prikazan način fragmentacija, kjer številka predstavlja datoteko, črka pa del te datoteke.

#	PNGs	pngCarver					Foremost					Scalpel					PhotoRec				
1	50	50	50	0	0	0	50	13	0	37	0	50	50	0	0	0	50	50	0	0	0
2	50	50	50	0	0	0	50	0	4	46	0	50	2	28	20	0	37	0	14	23	13
3	50	50	50	0	0	0	50	0	5	45	0	50	0	27	23	0	0	0	0	0	50
4	100	100	100	0	0	0	96	13	5	78	4	100	51	32	17	0	83	81	2	0	17
5	50	50	48	0	2	0	50	0	2	48	0	50	0	18	30	2	0	0	0	0	50
6	50	50	48	1	1	0	50	0	1	49	0	50	0	15	35	0	0	0	0	0	50
7	50	50	50	0	0	0	50	0	6	42	2	50	0	17	31	2	4	0	0	4	46
8	50	50	50	0	0	0	50	0	2	48	0	50	0	18	31	1	40	0	10	30	10
9	50	50	50	0	0	0	50	0	4	46	0	45	0	29	16	5	36	0	20	16	14
10	50	50	50	0	0	0	50	0	0	50	0	50	0	15	31	4	45	0	5	41	4
11	50	50	44	1	5	0	50	0	0	50	0	50	0	5	45	0	6	0	0	6	44
12	50†	50	50	0	0	0	50	9	6	34	1	50	50	0	0	0	22	22	0	0	28
13	50†	50	50	0	0	0	50	8	18	24	0	50	49	0	1	0	0	0	0	0	50
14	50†	50	37	8	5	0	50	0	11	37	2	50	0	28	21	1	7	0	4	4	42
15	50†	50	50	0	0	0	50	16	11	23	0	50	49	1	0	0	41	41	0	0	9
16	50†	50	36	7	7	0	50	0	13	37	0	50	0	32	18	0	45	0	28	17	4
17	100	100	100	0	0	0	95	0	11	79	10	100	0	44	54	2	100	0	45	53	2
18	100†	100	100	0	0	0	97	11	14	72	3	100	50	33	17	0	50	0	31	19	50
19	50†	50	50	0	0	0	50	9	7	34	0	50	50	0	0	0	0	0	0	0	50
20	50†	50	50	0	0	0	50	13	5	32	0	50	50	0	0	0	2	2	0	0	48
21	100†	100	100	0	0	0	100	17	17	66	0	100	50	29	21	0	54	5	26	23	46
22	50†	50	49	0	1	0	50	15	6	29	0	50	39	8	3	0	0	0	0	0	50
23	100†	100	100	0	0	0	97	10	15	72	3	100	50	28	22	0	50	43	7	0	50
24	100	100	100	0	0	0	95	0	13	82	5	100	0	39	61	0	50	0	19	31	50
25	150	150	150	0	0	0	145	0	15	130	5	150	0	49	99	2	75	0	23	52	75
26	200	200	200	0	0	0	193	0	30	162	8	200	0	80	119	1	104	0	49	53	98

Slika 6: Rezultati algoritmov za klesanje, stolpci pod vsakim posameznik klesalnikom predstavljajo naslednje: 1. število PNG slik, 2. popolne slik, 3. manjše napake v slik, 4. velike napake v slik, 5. pokvarjene ali manjkajoče slik

še posebej zaželjeno, da format podpira delitev same vsebine slike na več takih kosov. Če format tega ne podpira bo prednost predstavljenega pristopa le minimalna.

7.1 Analiza formatov

JPEG Format je sicer sestavljen iz več segmentov, vendar segmenti ne vsebujejo podatka za preverjanje pravilnosti vsebine. Poleg tega sama vsebina slike ni vsebovana znotraj segmenta in zato nima vnaprej napovedane dolžine, ter jo je potrebno brati zaporedoma. Format zato ni posebej primeren za opisano sintaktično klesanje.

BMP Datoteka vsebuje dvodimenzionalno polje, ki vsebuje sliko in ločeno hrani barvne tabele, ki definirajo barve posameznih slikovnih točk. Sama zasnova formata obeta strukturo iz različnih kosov. Format sestavljajo kosi fiksnih dolžin, ki jim rečemo glave in kosi spremenljivih dolžin. Posamezni kosi si morajo slediti v predpisanem vrstnem redu. Strukturo sestavlja BMP glava iz 14 bajtov in hrani velikost datoteke v bajtih. Sledi DIB glava, ki je lahko različne dolžine, odvisno od uporabljene verzije, in hrani podrobne informacije kako bo slika prikazan na zaslonu. Sledi tabela barv, ki je spremenljive dolžine, ki je bodisi predpisana ali pa shranjena v DIB glavi. Nato sledi dvodimenzionalno polje, ki hrani sliko. Obstaja množica opcijskih kosov, ki tvorijo BMP strukturo in jih ne bomo navajali eksplicitno.

Klesanje: Pomembne lastnosti strukture, ki jih lahko izkoristimo pri klesanju so sledeče. V BMP glavi je zapisana velikost datoteke v bajtih. Odmik kosa znotraj datoteke moramo izračunati s pomočjo predhodnih kosov, velikost pa je bodisi zapisana v predhodni glavi ali pa jo izračunamo po predpisanem postopku. Podatka za preverjanje pravilnosti posameznega kosa, kot je to CRC v primeru PNG formata, tukaj nimamo. Lahko pa izkoristimo podatek o velikosti BMP datoteke, ki se nahaj v BMP glavi, in sicer za preverjanje ali je skupna dolžina izklesanih kosov enaka predpisani dolžini na začetku datoteke.

TIFF Datoteke so sestavljene iz treh glavnih kosov; Glava datoteke (GD, eng.: IFH), ki vsebuje podatke o verziji TIFF formata in odmiku prvega PD kosa; Podatki datoteke (PD, eng.: IFD), ki vsebuje oznako, ki kaže na bitno sliko za ta kos. Zadnji 4 bajti pa vsebujejo podatek o odmiku naslednjega PD kosa; zadnji kos je bitna slika (BS) kjer se nahajajo dejanski podatki.

Klesanje: Vsaka datoteka vsebuje glavo, kjer so specifične informacije glede verzije TIFF formata. S pomočjo znanih verzij TIFF formatov bi tako lahko identificirali glave vseh TIFF datotek na disku katerega obdelujemo. Iz podatkov GD-ja bi lahko razbrali kje se nahaja prvi PD element, iz tega bi nato razbrali kje se nahaja drugi in tako naprej do zadnjega. Problem tega formata je, da ne vsebuje podatka za preverjanje pravilnosti vsebine, kar zelo oteži klesanje datotek katerih PD kos je več kot dvakrat fragmentiran, saj ni možnosti, da bi preverili če so dobljeni podatki pravilni. Opisan pristop za PNG slike bi lahko delno aplicirali tudi na TIFF format, vendar ne v celoti zaradi manjkajočih podatkov o CRC-u.

8. ZAKLJUČEK

V tem delu smo predstavili analizo članka, ki predlaga nov algoritem za sintaktično klesanje PNG datotek. Algoritem s pridom izkorišča posebne lastnosti PNG formata slik pri postopku klesanja. S tem algoritem prekosi obstoječe pristope, ki so običajno uspešni le, ko je datoteka na mediju shranjena zvezno. Avtorji prav tako predlagajo ogrodje za generiranje testnih množic, za namen evalvacije algoritmov, za klesanje datotek. Z uporabo omenjenega ogrodja tudi primerjajo predlagan algoritem z obstoječimi. Izkaže se da novo predlagani algoritem sintaktičnega klesanja prekosi vse do sedaj uporabljene pristope. Uspešen je v 20 od 26 testnih scenarijev, kjer ga v primeru neuspešnosti pogosto omejuje časovna omejitev ene ure.

Zaznali smo tudi potencialen problem metode. Ker se algoritem zanaša na strukturo PNG kosov, bo odpovedal v primerih, ko do fragmentacije pride kjerkoli na robu med enim in drugim kosom. V takem primeru izgubimo sintaktično informacijo, ter ne moremo uporabiti pristopov opisanih v članku.

Predlagan algoritem se sicer zelo dobro izkaže na omenjeni testni množici, vendar smo sami (študentje) odkrili tudi nekaj pomanjkljivosti, za katere smatramo, da jih je potrebno izpostaviti. Prva je ta, da je generirana testna množica nekoliko pristranska, saj z velikostjo vsebovanih PNG datotek omejuje ostale algoritme v primerjavi. Foremost ima namreč vgrajeno omejitev glede velikosti datotek, tako da rezultati ne odražajo povsem delovanja algoritma.

V nadaljnjem delu bi izboljšali pripravo tesne množice, da bi zajemala več različnih dimenzij slik. Tako bi odpravili pristranskost predlagane testne množice in omogočili boljšo primerjavo vseh obstoječih pristopov za klesanje slik.

9. VIRI

[1] Foremost file carver. http://foremost.sourceforge.net. Accessed: 2020-04-30.

[2] Photorec, digital picture and file recovery. https://www.cgsecurity.org/wiki/PhotoRec. Accessed: 2020-04-30.

[3] S. L. Garfinkel. Carving contiguous and fragmented files with fast object validation. *digital investigation*, 4:2–12, 2007.

[4] C. Grajeda, F. Breitinger, and I. Baggili. Availability of datasets for digital forensics–and what is missing. *Digital Investigation*, 22:S94–S105, 2017.

[5] J.-N. Hilgert, M. Lambertz, M. Rybalka, and R. Schell. Syntactical carving of pngs and automated generation of reproducible datasets. *Digital Investigation*, 29:S22–S30, 2019.

[6] J.-N. Hilgert, M. Lambertz, M. Rybalka, and R. Schell. Syntactical Carving of PNGs and Automated Generation of Reproducible Datasets, Aug. 2019.

[7] G. G. Richard III and V. Roussev. Scalpel: A frugal, high performance file carver. 01 2005.

[8] G. Roelofs and R. Koman. *PNG: The Definitive Guide*. O'Reilly & Associates, Inc., USA, 1999.

[9] Wikipedia. Disk sector — Wikipedia, the free encyclopedia, 2013. [Online; dostopano 1.maj.2020].

Artefakti za detekcijo manipulacije časovnih žigov v datotečnem sistemu NTFS v operacijskem sistemu Windows in njihova zanesljivost

[Ponovitev eksperimenta in analiza]

Matej Bizjak
Fakulteta za računalništvo in informatiko, Univerza v Ljubljani
Večna pot 113
1000, Ljubljana
mb7628@student.uni-lj.si

Miha Jamšek
Fakulteta za računalništvo in informatiko, Univerza v Ljubljani
Večna pot 113
1000, Ljubljana
mj6243@student.uni-lj.si

Gregor Kerševan
Fakulteta za računalništvo in informatiko, Univerza v Ljubljani
Večna pot 113
1000, Ljubljana
gk6431@student.uni-lj.si

POVZETEK

V seminarski nalogi raziščemo ugotovitve iz krovnega članka [4]. Ponovimo vse poskuse manipulacije s časovnimi žigi in jih skušamo na načine opisane v članku tudi odkriti oziroma dokazati, da so se zgodili. Vse poskuse manipulacije in postopke odkrivanja le-te opišemo. Zapišemo tudi razne probleme, s katerimi smo se srečali pri ponovitvi poskusa in skušamo obrazložiti njihove vzroke. V nekaj primerih je prišlo do neskladja z rezultati članka – kar smo tudi zapisali.

KLJUČNE BESEDE

časovni žigi, ponarejanje dokazov, manipulacija časovnih žigov, ntfs, dnevniki, digitalna forenzika

1. UVOD

Digitalni forenziki imajo ob preiskovanju digitalnih materialov precej dela z raziskovanjem koristnosti informacij, hkrati pa tudi z legitimnostjo podatkov. Pri tem so lahko zelo koristni dnevniški zapisi, med katerimi so nekateri artefakti še posebej koristni. Prvotni članek, na katerem temelji naše delo, se ukvarja z analizo dnevniških zapisov znotraj NTFS v operacijskem sistemu Windows. Poudarek je predvsem na manipulaciji časovnih žigov, orodjih, ki to omogočajo, ter možnostjo zaznave delovanja teh orodij.

Naš cilj je ponoviti njihove poskuse ter preveriti veljavnost njihovih ugotovitev. Postavljeno testno okolje je zelo podobno tistemu iz prvotnega članka, z nekaj spremembami virtualnega okolja in verzijami uporabljenih orodij. V delu se osredotočamo na tri poglavitna vprašanja, ki predstavljajo temelj naše raziskave: kateri artefakti so koristni pri zaznavanju spreminjanja časovnih žigov, kako zanesljivi so ti artefakti ter ali je mogoče zaznati delovanje zlonamernih orodij, ki omogočajo spreminjanje časovnih žigov.

2. PRIPRAVA IN IZVEDBA EKSPERIMENTA

Še preden gremo na sam potek eksperimenta, si najprej poglejmo, katero programsko opremo smo zanj potrebovali:

- Windows 10 Home verzije 1909
- Hyper-V Manager 10.0.18362.1
- FTK Imager 4.2.1
- Autopsy 4.14.0
- nTimestomp (x64) v1.1
- Timestomp
- LogFileParser v2.0.0.48
- NtfsFileExtractor v4.0.0.6
- Event Log Explorer 4.9.0
- Windows Event Viewer

Pri izvedbi eksperimenta smo sledili poskusu iz članka [4] in ga tako razdelili na 5 osnovnih korakov:

1. **Priprava**: Najprej smo znotraj virtualizacijskega okolja Hyper-V ustvarili nov navidezni stroj z navideznim trdim diskom fiksne velikosti 40 GB. Nanj smo namestili operacijski sistem Windows 10 z eno samo particijo. Vanj smo prenesli orodji za manipulacijo časovnih žigov nTimestomp in Timestomp, ustvarili pa smo tudi 3 tekstovne datoteke *f_default.txt*, *f_nTimestomp.txt* in *f_Timestomp.txt*. Prva predstavlja datoteko, katere časovni žigi ne bodo spremenjeni, drugima dvema pa bomo le-te spremenili. Poleg tega smo želeli omogočili dnevnike $USNJrnl s pomočjo ukaza: *fsutil USN createjournal m=1000 a=100 c:*, vendar smo kasneje pri analizi teh dnevnikov ugotovili, da so dnevniki $USNJrnl ob namestitvi Windows 10 že privzeto vključeni

- več o tem v poglavju 3.3. Naj omenimo tudi to, da v nasprotju s poskusom iz članka [4], nismo uspeli uporabiti orodja za manipulacijo časovnih žigov SetMace, zaradi določenih napak pri njegovem izvajanju. Vendar to nima nikakršnega vpliva pri rezultatih eksperimenta, saj orodje deluje na enak način kot ostala dva.

2. **Spreminjanje časovnih žigov:** Pognali smo obe orodji in pripadajočima testnima datoteka spremenili časovne žige. Programa poženemo v konzoli, pri čem mu podamo pot do datoteke in nove časovne žige. Primer izvedbe za program Timestomp izgleda takole: *.\timestomp.exe f_Timestomp.txt -z "Tuesday 4/28/2020 2:50:01 PM"*. V tabeli 1 so prikazani prvotni in spremenjeni časovni žigi za vse tri datoteke.

3. **Pridobivanje dokazov in njihova analiza:** Nato smo naredili sliko navideznega stroja z orodjem FTK Imager in nadaljevali s pridobivanjem podatkov in njihovo analizo. Izluščili smo zapiske o dnevnikih $LogFile, $USNjrnl in Windows Event Logs ter podatke o nastalih datotekah Prefetch in LNK. Nad zbranimi podatki smo izvedli podrobnejšo analizo, ki je opisana v poglavju 3.

4. **Ugotavljanje zanesljivosti pridobljenih dokazov:** S tem korakom smo želeli zakriti vse sledi, ki bi lahko kazale na to, da so bili časovni žigi datotek spremenjeni in da so bili pognani zlonamerni programi, ki to omogočajo. Tako bomo lahko ugotovili, ali so izbrani artefakti odporni proti aktivnemu napadalcu oz. škodljivi programski opremi. Kako smo to storili, je prav tako opisano v poglavju 3.

5. **Ponovno pridobivanje dokazov in njihova analiza:** Po koraku 4 smo ponovno ustvarili sliko navideznega stroja in podatke ponovno analizirali (korak 3), da bi preverili, če smo dokaze uspešno zakrili in ali so s tem nastali kakšni novi dokazi, s pomočjo katerih bi lahko dokazali, da je aktivni napadalec sledi poskušal zakriti.

V naslednjem poglavju si bomo pogledali podrobnosti izvedbe in rezultate pravkar opisanega eksperimenta.

3. ARTEFAKTI IN NJIHOVA ANALIZA

Pri analizi dokazov smo se osredotočili na artefakte, ki so forenzikom že znani, vendar jih večina še ni bila preizkušena za uporabo pri prepoznavanju manipulacije časovnih žigov datotečnega sistema NTFS. Tej so sledeči:

- Dnevnik $LogFile
- Datoteke Prefetch
- Dnevnik $USNjrnl
- Datoteke LNK
- Dnevnik Windows dogodkov

V nadaljevanju članka bomo zaradi boljše berljivosti uporabljali sledeče okrajšave:

- SIA: atribut NTFS-ja $STANDARD INFORMATION
- $MFT: tabela NTFS-ja Master file table
- MACE: časovni žigi Modified, Accessed, Created, MFT Changed
- tako torej SIA-M pomeni $STANDARD INFORMATION Modified časovni žig, SIA-A $STANDARD INFORMATION Accessed časovni žig itd.

Pogledali si bomo njihove lastnosti, kakšne dokaze smo iz slik uspeli izluščiti in ali je posamezen artefakt dovolj zanesljiv.

3.1 Dnevnik $LogFile

Windows operacijski sistem vse spremembe na datotečnem sistemu beleži v datoteko *C:\$LogFile*. Zatorej morajo tudi vsi poskusi manipulacije časovnega žiga datoteke spremeniti oziroma izbrisati zapis v $LogFile.

3.1.1 Ekstrakcija podatkov iz $LogFile

Za pridobivanje podatkov iz $LogFile smo uporabili dve prosto dostopni orodji. Najprej smo uporabili NtfsFileExtractor, s katerim smo pridobili $LogFile iz podatkovnega nosilca ($LogFile je tretji zapis v $MFT [2] [1]), nato pa smo z LogFileParser to datoteko prebrali v CSV format, kjer smo lahko iskali podatke o manipuliranih datotekah.

3.1.2 Dokazovanje manipulacije pri nespremenjenem $LogFile

Najprej smo pregledali sliko nosilca podatkov, kjer smo datotekam spremenili časovni žig, datoteko $LogFile pa pustili nespremenjeno. Po pretvorbi v CSV, opisani v prejšnjem poglavju (3.1.1), smo naredili preprosto iskanje po zapisih in precej hitro odkrili zapis o manipulirani datoteki s prvotnim časovnim žigom.

3.1.3 Dokazovanje manipulacije pri spremenjenem $LogFile - prvi poskus

$LogFile ima eno veliko pomankljivost pri odkrivanju manipulacije in sicer njegova velikost. Datoteka je običajno omejena na približno 60 MB in ko doseže to velikost, se stare vrednosti prepišejo z novimi in tako izgubimo informacijo o spremembah. Članek [4] navaja, da je taka količina podatkov dovolj le za 3 ure normalnega delovanja računalnika, kar pa ni prav veliko.

Tako smo spisali skripto (Izsek izvorne kode 1), ki nam neko datoteko 10.000 krat premakne iz enega imenika v drugi, v upanju, da nam bo prepisala dokaze o manipulaciji v $LogFile.

Pri zagonu skripte smo pri prvem poskusu bili nepazljivi in skripto pognali v navadnem delovanju Powershell terminala. Ko smo analizirali vsebino $LogFile smo opazili, da nekaterih podatkov o datotekah naša skripta ni prepisala - natančneje, datoteke so imele ustvarjeno simbolično povezavo v imeniku RecentItems katerih zapis je bilo moč najti v $LogFile in so vsebovale prvotni časovni žig.

Tabela 1: Primerjava časovnih žigov pred in po manipulaciji

Ime datoteke	Prvoten SIA-C	Spremenjen SIA-MACE	Čas zagona orodja
f_default.txt	28/04/2020 14:10 CEST	/	/
f_nTimestomp.txt	28/04/2020 14:11 CEST	28/04/2020 14:50:01:1234567 CEST	28/04/2020 15:19
f_Timestomp.txt	28/04/2020 14:11 CEST	28/04/2020 14:50:01 CEST	28/04/2020 15:20

Izsek izvorne kode 1: Skripta za prepis $LogFile

```
if not exist "C:\flood" mkdir C:\flood
for /l %%x in (1, 1, 10000) do (
    echo %%x
    copy NUL "C:\flood\file.txt"
    echo index%%x >> C:\flood\file.txt
    echo C:\flood\file.txt > NUL
    del C:\flood\file.txt
)
```

3.1.4 Dokazovanje manipulacije pri spremenjenem $LogFile - drugi poskus

Prejšnji poskus smo ponovili, s to razliko, da smo tokrat skripto pognali v privilegiranemu načinu delovanja terminala Powershell.

V tem primeru, smo dosegli, da je informacija o prvotnem časovnem žigu datotek bila izbrisana iz $LogFile.

3.2 Datoteke Prefetch

Windows OS ob prvem zagonu programa avtomatsko ustvari datoteko Prefetch - PF, z namenom da pohitri vse naslednje zagone tega programa [5]. Ob vsakem naslednjem zagonu bo Windows poiskal pripadajoče datoteke PF, katere so običajno shranjene na lokaciji *C:/Windows/Prefetch*, in z njeno pomočjo pohitril njegov zagon. Windows si s pomočjo datotek PF beleži podatke o zadnjih osmih zagonih programa in te ostanejo v sistemu, tudi če je program izbrisan.

V primeru, da bi bilo orodje za manipulacijo časovnih žigov pognano večkrat zapored v kratkem času (na primer, za manipuliranje žigov več datotek), bi lahko s pomočjo datotek PF zaznali, da je bilo na delu sumljivo orodje, saj bi imele datoteke PF približno enak SIA-C.

Vendar ker so datoteke PF običajne nezaščitene datoteke operacijskega sistema, jih je trivialno izbrisati. Zelo težko je ugotoviti, da kakšna datoteka manjka, saj lahko izbrišemo vsako posebej in bo tako še vedno v mapi veliko drugih datotek PF, ki pripadajo drugim programom. Nam je sicer za razliko od avtorjev prvotnega poskusa s pomočjo orodja Autopsy iz nedodeljenga prostora na disku uspelo izluščiti nekaj datotek PF, ki so pripadale orodjema nTimestomp in Timestomp. Obnovljena datoteka *TIMESTOMP.EXE-D3ED6F07.pf* je še vedno vsebovala čas SIA-C, in sicer 2020-04-28 15:20:29 CEST, prav tako smo našli datoteke PF programa nTimestomp, ki so prav tako še vsebovale čas nastanka.

Zaradi enostavnosti brisanja datotek PF, se te niso izkazale za dober način zaznavanja manipulacije časovnih žigov. S preiskovanjem nedodeljenega dela particije nam sicer mogoče uspe obnoviti podatke o zagonu programa, vendar se takšni programi lahko zamaskirajo z različnimi imeni, kar nam dodatno oteži delo. Prav tako bi po našem mnenju težko dokazali, da je program spremenil časovni žig neke druge datoteke. Poleg tega pa lahko uporabnik operacijskega sistema po želji onemogoči storitev Superfetch (v novejših verzijah operacijskega sistema Windows je znana pod tem imenom) in tako prepreči ustvarjanje datotek PF.

3.3 Dnevnik $USNjrnl

Dnevnik $USNjrnl vsebuje zapise o vsakršnih spremembah nad datotekami. Za ta artefakt smo našli nekaj možnih napak v originalnem članku. Ena izmed teh je trditev, da je $USNjrnl samodejno izključen in da ga je potrebno najprej vključiti, če ga želimo nato analizirati. Na našem sistemu je bil vključen od začetka, poleg tega pa smo dokaze za to našli tudi v drugih virih, ki pravijo, da je od operacijskega sistema Windows Vista naprej privzeto vključen [8]. V originalnemu članku so dnevnik $USNjrnl najprej izvlekli z Autopsy-jem, nato pa ga analizirali še z UsnJrnl2Csv. Alternativna možnost, ki smo je našli, je uporaba UsnJrnl2Csv v načinu *scan*, ki sicer porabi več časa, saj pregleda celoten disk, vendar lahko najde tudi skrite ali celo izbrisane podatke. Iz teh dveh sprememb sklepamo, da je koristnost dnevnika $USNjrnl v članku slabo ocenjena.

BASIC_INFO_CHANGE je en izmed bolj koristnih zapisov v dnevniku, saj vključuje spremembe nad metapodatki in časovnimi žigi datotek. Našli smo ga pri obeh spremenjenih datotekah, kar nakazuje na spreminjanje metapodatkov. Ravno tako lahko s pomočjo tega dogodka najdemo programe, ki so spreminjali datoteke, s čimer smo našli programa nTimestomp in Timestomp. Oba sta imela zapise ob enakih časih kot spremenjene datoteke, kar nakazuje na njihovo povezanost.

Če bi nekdo želel izbrisati in izključiti dnevnik $USNjrnl, bi to lahko izvedel z ukazom *fsutil usn deletejournal /d c:*. Izbris $USNjrnl povzroči nastanek dogodka z ID-jem 3079 v dnevniku Windows dogodkov, iz česar lahko sklepamo na možno manipulacijo podatkov. Našli pa smo nekaj problemov ob izvedbi tega ukaza, saj nekatere aplikacije zahtevajo obstoj $USNjrnl za svoje delovanje in ga, če je izključen, same vključijo [6] [7]. V našem primeru sam izbris dnevnika ni imel velikega vpliva, saj smo z orodjem UsnJrnl2Csv uspeli najti velik del izbrisanega dnevnika.

S tem v mislih lahko zaključimo, da dnevnik $USNjrnl lahko doprinese veliko več, kot je opisano v originalnem članku in lahko z uporabo močnih orodij, kot je UsnJrnl2Csv, doprinese veliko informacij k forenzični analizi.

3.4 Datoteke LNK

Datoteke LNK predstavljajo bližnjice do lokalnih datotek. Lahko jih uporabnik ustvari ročno ali pa so avtomatsko ustvarjene s strani operacijskega sistema Windows. Vsakič ko ustvarimo ali prvič odpremo datoteko na lokalnem disku, se ustvari bližnjica do te datoteke. Te se nahajajo na lokaciji *C:/Users/NAME/AppData/Roaming/Microsoft/Windows/*

Recent. Datoteke LNK so se nam pri eksperimentu zdele pomembne, ker imajo svoj lasten $MFT vnos, ki ga lahko analiziramo. Vsakič ko odpremo datoteko na lokalnem disku, se popravita tudi časa SIA-A in SIA-E pri pripadajoči bližnjici. Velja torej pravilo, da si morata biti časa SIA-A in SIA-E med datoteko in pripadajočo bližnjico zelo blizu. Pri našem poskusu sta časa zelo odstopala, saj sta po manipulaciji časovnega žiga tekstovnih datotek časa bližnjic ostala nespremenjena in je bil to zato dober dokaz.

Datoteke LNK so torej lahko pokazatelj, da so bili časovni žigi prisilno spremenjeni, vendar je treba biti pozoren še na, recimo, antivirusne programe, ki ob pregledu LNK datotek spremenijo čas SIA-A. Tako da moramo biti pri pregledovanju tega artefakta pozorni le na čas SIA-E.

Problem zanesljivosti tega artefakta je dejstvo, da so datoteke LNK navadne datoteke, ki jih lahko tako kot datoteke PF enostavno izbrišemo, poleg tega pa jim lahko tudi spremenimo časovni žig, na enak način kot smo storili pripadajoči datoteki. Tako kot avtorjem prvotnega poskusa, tudi nam ni uspelo obnoviti izbrisane datoteke LNK in tako nismo mogli dokazati ponaredbe časovnih žigov.

Prav tako kot lahko uporabnik onemogoči avtomatsko ustvarjanje datotek PF, lahko to stori tudi za datoteke LNK. To se stori še lažje in sicer kar v nastavitvah, kjer nastaviš, da ne želiš, da OS shranjuje podatke o nedavnih lokacijah in datotekah. Eden izmed avtorjev članka, je imel to nastavitev na svojem računalniku izklopljeno, kar bi lahko pomenilo, da bi bil za nezanemarljiv delež uporabnikov operacijskega sistema Windows ta artefakt že takoj izključen.

3.5 Dnevnik Windows dogodkov

Dnevnik Windows dogodkov ne hrani podatkov, s katerimi bi lahko neposredno dokazali manipulacijo časovnih žigov. Avtor članka [4] se je tako opiral na podatke o začetku in koncu seje uporabnika (dogodki prijave in odjave - 7001 in 7002) in uspel dokazati manipulacijo v primeru, ko je manipulirani časovni žig bil izven časa trajanja uporabniške seje. V primeru, da časovni žig ne kaže na čas izven seje, preiskovalec ne more (z izključno dnevniki Windows dogodkov) dokazati manipulacije.

V primeru pa, da je časovni žig izven uporabniške seje, lahko samo na en način prikrijemo čas prijave in odjave uporabnika - z izpraznitvijo dnevnikov. Če izpraznimo dnevnike, se informacije o začetku in koncu uporabniške seje izgubijo.

V našem poskusu smo te dnevnike izbrisali v Windows orodju Event Viewer, da bi prikrili našo manipulacijo. To bi lahko napadalec storil tudi programsko, vendar bi za to potreboval administratorske pravice. Podatkovni nosilec z izpraznjenimi dnevniki smo nato z uporabo orodja FTK Imager kopirali v surovo sliko (zapis *.dd) [3]. To sliko smo z orodjem Autopsy odprli, iz nje izvlekli dnevniške datoteke, in jih nato odprli z orodjem Event Log Explorer. Hitro smo odkrili zapis o dogodku 104, ki zapiše kdo in kdaj je ta dnevnik izpraznil. Informacija, ki sama po sebi ne dokaže manipulacije s časovnim žigom, lahko pa nam služi kot potrditev suma o skrivanju aktivnosti.

Prav tako smo ob izbrisu prej omenjenega $USNjrnl (poglavje 3.3) odkrili, da se v aplikacijskem delu dnevnika ustvari dogodek 3079, ki nam pove, da je le-ta bil izbrisan. Žal pa nam kot pri izbrisu celotnega dnevnika, ta podatek le potrdi sum manipulacije, ne more pa biti uporabljen kot dokaz.

4. ZAKLJUČEK

Spreminjanje časovnih žigov z orodji, kot so nTimestomp in Timestomp, postaja vedno bolj enostavno in zato tudi vedno bolj popularno za uporabo pri prikrivanju zločinov. Z njihovim razvojem pa morajo tudi forenziki najti nove načine prepoznavanja sprememb nad metapodatki datotek. S tem člankom smo raziskali uporabnost novih metod razpoznavanja delovanja takih programov. Sledili smo preizkusom iz krovnega članka [4] ter preverili veljavnost njihovih zaključkov.

S prvotnim virom se skoraj v celoti strinjamo, nekaj neskladnosti pa smo zaznali pri testiranju $USNjrnl. Pri tem dnevniku smo mnenja, da je lahko zelo koristen pri forenzični analizi, saj beleži zelo pomemben dogodek, ki se sproži ob spreminjanju metapodatkov datotek. Ravno tako pomemben dnevnik je $LogFile, nad katerim je bilo opravljenih že veliko raziskav in predstavlja v mnogih primerih zelo pomemben artefakt. Oba dnevnika pa imata problem s časovno veljavnostjo, saj po določenem času začneta izgubljati starejše zapise. Kot rešitev zato predlagamo analizo čim večih različnih artefaktov.

V tem članku smo opisali več različnih dnevnikov datotečenega sistema NTFS v operacijskem sistemu Windows, kateri se med seboj dopolnjujejo in skupaj predstavijo bolj jasno sliko dogodkov. Je pa to šele začetek v raziskovanju manipulacije časovnih žigov, kjer ostaja še veliko prostora za nadaljnje raziskave.

5. VIRI

[1] J. Fichera and S. Bolt. Chapter 6 - Host Analysis. In J. Fichera and S. Bolt, editors, *Network Intrusion Analysis*, pages 153 – 167. Syngress, Boston, 2013.

[2] N. A. Hassan and R. Hijazi. *Data Hiding Techniques in Windows OS*. Syngress, 2017.

[3] R. Murphey. Automated Windows event log forensics. *Digital Investigation*, 4:92 – 100, 2007.

[4] D. Palmbach and F. Breitinger. Artifacts for detecting timestamp manipulation in NTFS on Windows and their reliability. 2020. DFRWS EU 2020.

[5] N. K. Shashidhar and D. Novak. Digital forensic analysis on prefetch files. *International Journal of Information Security Science*, 4(2):39–49, 2015.

[6] Microsoft (2017). Fsutil usn. https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/fsutil-usn, last accessed 02/05/2020.

[7] Microsoft (2018). Change Journals. https://docs.microsoft.com/sl-si/windows/win32/fileio/change-journals?redirectedfrom=msdn, last accessed 02/05/2020.

[8] UsnJrnl2Csv (2019). https://github.com/jschicht/usnjrnl2csv, last accessed 02/05/2020.

Restavriranje podatkov iz podatkovne baze SQLite

Nejc Ribič
Fakulteta za Računalništvo in Informatiko
Univerza v Ljubljani
nr4758@student.uni-lj.si

Matic Adamič
Fakulteta za Računalništvo in Informatiko
Univerza v Ljubljani
ma5094@student.uni-lj.si

ABSTRACT

Seminarska naloga predstavlja pristope in koncepte restavriranja podatkovne baze SQLite. Za zgled smo uporabili članek [5], ki prikazuje osnovne in napredne metode restavriranja podatkov. V seminarski nalogi smo predstavili delovanje podatkovne baze SQLite, opisali metodologijo, ki jo uporablja članek [5] in hkrati opisali možne izboljšave. Opisali in raziskali smo tudi delovanje orodja `bring2lite` in ga predstavili v praksi.

General Terms
Iskanje izbrisanih podatkov, podatkovna baza SQLite, forenzika

Keywords
SQLite, forenzika, podatki

1. UVOD

Kot seminarsko nalogo smo si izbrali članek z naslovom *A structural Concept and Tool for Forensic Data Analysis and Recovery of Deleted SQLite Record* [5]. Cilj seminarske naloge je podrobna analiza in delovanje podatkovne baze SQLite ter iskanje različnih možnosti restavriranja izbrisanih podatkov. Za zgled in začetno idejo smo si izbrali članek [5], ki opisuje nekaj osnovnih in hkrati tudi nekaj naprednih pristopov restavriranja že izbrisanih podatkov.

Seminarska naloga je sestavljena iz petih delov. V prvem delu predstavimo kratko zgodovino podatkovne baze SQLite ter kako SQLite v splošnem hrani podatke. Nato predstavimo možne nastavitve za usvarjanje podatkovne baze. Sledi predstavitev notranje strukture podatkovne shrambe ter lokacije izbrisanih podatkov. Nato predstavimo še pristope, ki jih lahko uporabimo za restaviranje podatkove in na kratko opišemo njihove prednosti in slabosti. Na koncu sledita še sklep in zaključek, kjer predstavimo subjektivno mnjenje o restavriranju podatkov podatkovne baze SQLite in podamo ideje za možne nadgradnje.

2. PODATKOVNA BAZA SQLITE

Podatkovna baza SQLite je knjižnica (angl. library), ki omogoča hranjenje podatkov lokalno brez potrebe servira-nja strežnika. Za samo postavitev podatkovna baza ne potrebuje nobene konfiguracije in temelji na SQL transakcijski strukturi [4]. Izvorna koda podatkovne baze SQLite je odprtokodna, kar pomeni, da jo lahko kdorkoli uporabi za komercialne kot tudi osebne potrebe. Podatkovna baza SQLite je najbolj pogosto uporabljena podatkovna baza na svetu, uporabljena je na ogromno znanih projektih, kot so recimo Android, Adobe, Apple, ipd.

SQLite ima vgrajeno delovanje SQL jezika, vendar za razliko od drugih SQL podatkovnih baz, SQLite ne potrebuje strežnika za procesiranje jezika. V osnovi SQLite piše in bere neposredno na datotečni sistem - celotna struktura podatkovne baze, kot so: indeksi, tabele, sprožilci in pogledi so shranjeni neposredno v eni datoteki. Ena izmed prednosti podatkovne baze SQLite je, da je struktura datoteke nedovisna od datotečnega sistema - slednje pomeni, da je za datoteko popolnoma vseeno, če sistem uporablja debeli konec (angl. big-endian) ali mali konec (angl. little-endian). Potrebno pa je omeniti, da podatkovna baza SQLite ne nadomesti dejanskih podatkovnih baz temveč le izboljša način hranjena podatkov na nivoju datotek [4].

Ker je podatkovna baza SQLite hranjena na datotečnem sistemu, je hitrost delovanja podatkovne baze odvisna od hitrosti diskovja. Podatkovno bazo lahko prav tako pohitrimo, če sistemu dodamo čimvečjo količimo spomina (angl. RAM) in s tem zagotovimo izredno hitro branje iz podatkovne baze [4]. Podatkovna baza uporablja vse koncepte atomarnosti, konsistence, izolacije ter trajnosti - krajše ACID.

3. NASTAVITVE OB USTVARJANJU BAZE

Ob postavitvi same SQLite baze se lahko nastavijo številni parametri oziroma pragme in pa velikost ene strani. Za velikost strani lahko izberemo katero koli številko, ki je potenca števila 2 in je med 512 in 65.536. Pragem je približno 70, odvisno od same verzije SQLite. Forenzično pomembne pa so naslednje: [5, 1, 3].

- **secure_delete**: Vrednosti lahko nastavimo na *0*, *1* ali *FAST*. Prvi nastavitvi *0* se zasabo pusti forenzične artefakte v seznamu prostih strani (angl. freelists), te so opisani v poglavju 4. Pri vrednosti *1* pa se vse zbrisane zapise prepiše z ničlami. Nastavitev *FAST* izbriše vse podatke iz strani v B-drevesih, zaseboj pa pusti forenzične sledi na seznamu praznih strani.

- **auto_vacuum**: Funkcija je lahko vklopnjena ali izklopljena. Če je vklopljena pomeni, da se za sabo zbrišejo vse neuporabljene strani in teh tudi ne obdrži v seznamu prostih strani.

- **journal_mode**: Možne vrednosti so *DELETE, TRUNCATE, PERSIST, MEMORY, WAL* in *OFF*. Parametri *DELETE, TRUNCATE, PERSIST* in *MEMORY* ustvarijo dodatno datoteko s končnico "-journal", ki vsebuje dnevnik podatkovne baze. Parameter *WAL* pa ustvari datoteko s končnico "-wal". Ta datoteke se nikoli za seboj ne zbriše in ima fiksno velikost. Ko je velikost dosežena, se vnosti začnejo prepisovati. V primeru nastavitve *OFF* se dnevnika ne ustvari.

4. NOTRANJA STRUKTURA

SQLite datoteka je interno strukturirana kot B-drevo. Na višjem nivoju pa je razdeljena na dva dela: glave (angl.) header in telo (angl. body) [1]. Glava je fiksne velikosti in zavzema prvih 100 bajtov datoteke. Vsebuje informacije o podatkovni bazi kot so verzija SQLite baze, velikost baze v straneh, število prostih strani, kodiranje teksta (UTF-8, UTF-16, ...) itd [2].

Po glavi sledi telo, kjer je kot prvi zapis zapisana prva stran. Ta vsebuje glavno tabelo (angl. master table), ki vsebuje informacije o vseh tabelah, indeksih in predstavlja vhodno točko do obeh omenjenih stvareh.

Vsaka stran je enega od naslednih tipov:

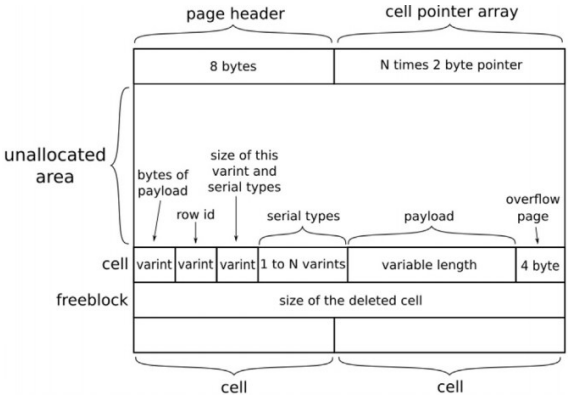
- **Freelist trunk page:** Stran je organizirana v zloge po 4 bajte. Vsak izmednjih je kazalnik, ki kaže na strani. Prvi kazalnik kaže na naslednji *freelist trunk page*, ostali pa kažejo na številko strani, ki je tipa *freelist trunk page*. Kazalec, ki kaže na prvi zapis v tej strukturi je shranjen v glavi na zamiku 32.

- **Freelist leaf page:** Je stran, kamor lahko shranimo nove zapise o straneh tipa *Table-bree leaf page*. Sama stran ne vsebuje podatkovnih zapisov, le kazalnike.

- **Table b-tree interior page:** To so strani, ki vsebujejo kazalnike na svoje otroke (angl. children) in na desnega najglobjega otroka. Ne vsebujejo podatkovnih zapisov. Kazalniki so dolgi 4 bajte in kažejo na strani istega tipa ali na liste drevesa, ki so tipa *Table b-tree leaf page*.

- **Table b-tree leaf page:** So ključnega pomena, saj vsebuejo podatkovne zapise. To so strani, ki so listi B-drevesa in so edine strani, ki vsebujejo trenutno obstoječe podatke v podatkovni bazi. Vsaka stran je razdeljena na glavo (angl. header) in telo (angl. body). Glava je dolga 8 bajtov, kjer so zapisani podatki o strukturi strani. Te podatki definirajo tip strani, začetni prazni blok (angl. freeblock), prazne celice, itd. Kazalec na strukturo prostih blokov se začne z odmikom 1, in kaže na prvi blok v verigi blokov. Te bloki zajemajo prazne celice v praznem prostoru strani (angl. unallocated area). Ker so bloki urejeni v verigo, ima vsak blok določeno strukturo. Prvi del je dolg 2 bajta in je kazalec na naslednji blok v verigi (če je vrednost kazalca 0, potem je to zadnji blok v verigi). Druge del je dolg 2 bajta in definira dolžino trenutnega bloka. Zadnji del definira prazen prostor v bloku - ta del je potencialno zanimiv, ker se tudi tam potencialno nahajajo zbrisani podatki. Glavi sledi polje kazalnikov, ki kažejo na celice (angl. cells), ki vsebujejo končne podatkovne zapise.

Strani tipa *Table b-tree leaf page* imajo nenaslovljen prostor med poljem kazalnikov celic in dejanskimi celicami, ki se vedno shranjujejo kar se da daleč na koncu strani. Ko se dodajajo novi zapisi oziroma podatki v podatkovno bazo, se iz konca strani polnijo celice, ki vsebujejo te podatke in od začetka se podaljšuje polje kazalcov, ki kažejo na te celice. Poleg aktivnih celic se nahajajo tudi zbrisane celice, ki so naslovljene v stukturi praznih blokov. V primeru, da se zapolni celoten prostor v celici, se ustvari nova stran, nanjo pa kaže prelivni (angl. overflow) kazalec, ki se nahaja na koncu celice. Prelivne strani so med seboj povezane kot povezan seznam. Struktura strani je vidna na sliki 1 [5, 1, 2].



Struktura listne strani (angl. table B-tree leaf page).

5. METODOLOGIJA

Pri iskanju zbrisanih podatkov se obdela več diskretnih komponent podatkovne baze SQLite. Za vsakega opišejo postopke in težave metod, ki so jih uporabili. Podatke poskušajo obnoviti iz nenaslovljenega prostora, prostih blokov in seznama prostih strani.

5.1 Ekstrakcija iz prostih blokov

Poskušajo pridobiti zbrisane podatke iz prostih blokov, nenaslovljenega prostora v listnih straneh - *table b-tree leaf page* in seznama prostih straneh. Na začetku se pregleda glava glavne tabele (angl. master table), saj ta vsebuje pomembne informacije o vseh tabelah, shemi in indeksih. Uporabijo algoritem, viden v 4, ki med seboj poveže vse strani, kot je specificirano v shemi baze. V naslednjem koraku poskušajo obnoviti zbrisane podatke v iz aktivnih strani. V akrivnih straneh se zbirsani podatki lahko nahajajo v prostih blokih ali pa v nenaslovljenem prostoru.

Pri obnovitvi podatkov iz prostih blokov, se najprej prebere kazalec v glavi strani, ki kaže na prvi blok v verigi. Sprehodijo se skozi vse bloke, ki so med seboj povezani kot povezan seznam. Bloki vsebujejo informacije kot so identifikacijska številka zapisa oziroma vrstice (angl. rowid) in podatkovni tip zapisa. Za določanje dolžino glave celice, s katero bi lahko določili tudi dolžino podatka v celici, naredijo sledeč izračun:

Naj bo v dolžina prvih treh številk (angl. varints), kaj številke predstavljajo je vidno na sliki 1 pri odseku "cell". Naj bo s število vseh primitivnih tipov (angl. serial types), ki so definirani v shemi trenutne strani. Naj bo b število podatkovnih tipov, ki imajo lahko različno dolžino (velik binarni objekt (angl. blob, binary large object) and tekst).

Nato izračunajo dolžino glave celice po formuli:

$$v + s + b$$

Izračun je lahko problematičen, saj sta prvi dve spremenljivki lahko neznani. Prva spremenljivka lahko naraste na maksimalno vrednost 27 bajtov, saj ima lahko vsaka izmed številk dolžino 9 bajtov. Še ena težava nastopi pri določanju spremenljivke b, saj se je ne da prebrati iz sheme. Zato v tem primeru iterirajo po spremenljivki v in preverijo za vsako možno vrednost b dokler ne dosežejo praga. V vsakem koraku iteracije se naredi primerjava med pridobljenem poljem celice in definiranim podatkovnim tipom v shemi na trenutni strani.

Po izračunu obnovijo izbisane podatke z algoritmom 3. Izračunajo dolžino zbrisanega podatka s pomočjo prejšnjih izračunanih vrednosti, dolžina se izračuna na drugi vrstici v algoritmu 3. Obnovljeni podatki so obdelani v naknadno. Algoritem kot rezultat vrne seznam podatkov. Ker je določitev začetka podatkov problematična, se obravnava več možnih začetkov, takih je lahko 6. Možno je tudi, da se v enem prostem bloku nahaja več zbrisanih celic, kar pomeni da je v seznamu lahko vrnjenih več zbrisanih podatkovnih zapisov iz celic [5].

5.2 Ekstrakcija iz nenaslovljenega prostora

V primeru ekstrakcije podatkov iz nenaslovljenega prostora uporabijo algoritem 2. Algoritem uporabijo na straneh tipa *table b-tree leaf page*. Najprej se izračuna dolžina glave strani in število celic, ki jih vsebuje stran - te so zapisane v polju kazalnikov celic. S pomočjo teh dveh lahko določijo del strani, ki je nenaslovljen prostor. Po ekstrakciji tega dela se preveri, če so kateri biti nastavljeni na 1 [5].

5.3 Ekstrakcija iz seznama prostih strani

Pri obnavljanju podatkov iz seznama prostih straneh uporabijo prilagojen algoritem 2. Te strani same bo sebi ne vsebujejo nobenih podatkovnih zapisov, vendar samo kazalnike ki kažejo na listne strani - strani, ki vsebujejo podatkovne zapise. Z izhodom (nenaslovljen prostor) iz algoritma 2, je možno obnoviti proste bloke in celice z algoritmom 3 [5].

```
Data: page (i.e. hexdump of currently processed page)
Result: processed unallocated area
header-length = get-header-length(page);
cellpointerarray-length =
get-cell-pointer-array-length(page);
stop = get-start-of-cell-content-area(page);
unalloc-area = extract-unalloc-area(header-length +
cellpointerarray-length, stop);
deleted-record-pointer = NULL;
for x = 0 → length(unalloc-area) do
    byte = extract-byte(x);
    if byte != "" then
        deleted-record-pointer = x;
        break;
    end
end
result = reduce-unalloc-area(unalloc-area,
deleted-record-pointer);
return result;
```

Algoritem 3: Ekstrakcija nenaslovljenega prostora iz listne strani B-drevesa.

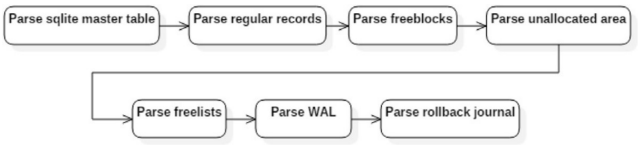
```
Data: serial types (based on the table shema), L
      (estimated value of header lengh), freeblock length
Result: multiple possible solutions that could match the
        record based on the related shema
possible-solution = [];
length =
calculate-length-of-freeblock-content(serial-types);
content = extract-content(L, length);
possible-solution.append(content);
return possible-solution;
```

Algoritem 2: Ekstrakcija podatkov iz prostega bloka.

```
Data: schemas (array of all entry pointers), page-size
Result: all schemas connected to their pages
result = [];
/* loop over all entry pages                    */
for p in schemas do
    if p > 0 then
        page = extract_page((p-1) * page_size);
        if page == interior b-tree page then
            result[p].append(
            page.header.right_most_pointer);
            numb_cells = page.header.number_of_cells;
            cells = extract_cells(page, numb_cells);
            result[p].append(cells);
        else
            result[p].append(p);
        end
    end
end
return result;
```

Algoritem 1: Algoritem za povezovanje strani glede na shemo baze.

6. ORODJE BRING2LITE



Postopki metode za obnovitev zbrisanih podatkov iz podatkovne baze SQLite.

Orodje *bring2lite* je razvito po psevdokodi iz omenjenih algoritmov. Implementacija je izvedena v programskem jeziku Python. Uporabi se ga lahko preko preprostega grafičnega vmesnika ali preko ukazne vrstice. Orodje izvede vse omenjene postopke za ekstrakcijo podatkov in kot razultat shrani podatke na željeno mesto v obliki datotečne strukture. Glavne predosti tega orodja v primerjavi z drugimi orodji je, da podatke med seboj smiselno poveže glede na definirano shemo v podatkovni bazi. Prav tako lahko določi podatkovne tipe stolpcev, kar je pomembno pri dekodiranju podatkov.

Pred ekstrakcijo podatkov se na začetku zberejo pomembne informacije o bazi. Osnovne informacije o bazi se preberejo iz glave datoteke. Zberejo se tudi informacije o shemi za povezovanje tabel. Nato program iterira skozi vse določene strani. Po končani analizi se ustvari datotečna struktura, ki vsebuje najdene podatke [5].

7. METODE TESTIRANJA IN SCENARIJI

Pri testiranju orodja *bring2lite* zgradijo več podatkovnih baz. Definirajo shemo, ki vsebuje eno tabelo, ki vsebuje stolpce: id, ime, priimek in zip. Predstavijo 6 scenarijov iz katerih poskusijo obnoviti zbrisane podatke. Ker obstaja 12 pragem, ki vplivajo na ekstrakcijo zbrisanih podatkov, ustvarijo skupno 72 podatkovnih baz, vsaka baza za eno kombinacijo scenarija in nastavitev pragm. Vsaka baza vsebuja zgoraj omejeno shemo.

Scenariji:

- Dodaj 1 zapis in ga zbriši
- Dodaj 3 zapise in z briši enega
- Dodaj 3 zapise in zbriši vse
- Dodaj toliko zapisov, da bo ustvarjena nova stran in izbriši vse zapise iz novonastale strani
- Dodaj toliko zapisov, da bo ustvarjena nova stran in izbriši vse zapise iz prve strani
- Dodaj toliko zapisov, da bo ustvarjena nova stran in izbriši vse zapise iz obeh strani

Prvi in drugi scenaij sta najbolj preprosta scenarija. Z drugim scenarijom se pokažejo tudi kaj SQLite naredi v primeru ko se iz prve strani zbrišejo vsi podatki. Zadnji trije scenariji se nanašajo na obnašanje SQLite baze pri ustvarjanju in brisanju listne strani v B-drevesu. V njihovem primeru vstavijo 159 zapisov preden se napolni ena stran, številka pa je odvisna od podatkovnih tipov stolpcev in dolžina podatkovnega zapisa [5].

8. REZULTATI IN PRIMERJAVA Z DRUGIMI ORODJI

Vzorec rezultatov je viden v tabeli 1. Rezultati označeni z *0* označujejo tudi delno obnovljene podatke. Prvi stolpec nakazuje, da je možno obnoviti vse zbrisane podatke v vseh scenarijih, če so pragme nastavlje na: *secure_delete = 0*, *autovacuum = 0* in *journalmode = OFF*. Z 'hex editor' urejevalnikom pregledajo datoteko podatkovne baze in opazijo, da so podatki še vedno zapisani, kot da ne bi bili zbrisani, z izjemo prvih 4 bajtov. V tem primeru je SQLite sprostil prostor z dodajanjem novega prostega bloka, ki spremeni podatek o prvem bloku v glavi strani.

V drugem stolpcu, kjer je bila v bazi nastavljena vrednost *secure_delete = FAST* ostali vrednosti pa sta bili nespremenjeni, orodje ni bilo zmožno obnoviti vse podatke. Ta rezultat je pričakovan zaradi nastavitve *secure_delete = FAST*, saj ta zaseboj pobriše vse vnose v straneh B-drevesa in je možno obnoviti le podatke iz seznama prostih straneh.

Za primerjanje različnih orodij izberejo korpus predstavljen v članku Nemetz et al. (2018)[6], ki je sestavljen iz dveh segmentov. Prvi segment testira generalno funkcionalnost orodja: dekodiranje, branje posebnih znakov in dolga imena stolpcev tabel. Drugi segment se nanaša na zbrisane zapise. Ta je razdeljen na 5 delov, vsaka je poimenovana z dvomestno oznako (npr. *0A*). Skupno je 27 datotek, ki predstavljajo samostojno podatkovno zbirko. Testirajo nekaj prosto dostopnih orodij na celotnem korpusu, preostala orodja pa citirajo iz članka Nemetz et al. (2018)[6], ki so bila testina na enakem korpusu. Rezultati so vidni v tabeli 2. Stolpci kažejo koliko zbrisanih vnosev je določeno orodje uspelo obnoviti. Znak x označuje, da so bili podatki obnovljeni le delno.

Prva kategorija je označena z *0A*. V tem primeru se ustvari ena ali več tabel kjer se v vsako vstavi 20 vrstic nato pa se jih zbriše z ukazom *DROP TABLE*. V primeru podatkovne zbirke *0A-02* avtorji *bring2lite* pojasnjujejo, da do slabih rezultatov pride zaradi tega, ker zbirka vsebuje seznam prostih straneh, ki ne vsebuje informacij o prostih blokih. Njihovo orodje zaporedno pregleda stran in ne loči med podatkovnimi zapisi in prostimi bloki. Nad omejeno podatkovno zbirko pa so se izvedla naključna brisanja, prosti bloki pa so se tako ustvarili po stvarjenju prvih zapisov, kar *bring2lite* ne uspe razbrati.

V drugi kategoriji, *0B*, so podatkovne baze pripravljene na sledeč način: 1) ustvari se ena ali več tabel, 2) vse se napolni z 10 vnosi, 3) izvede se ukaz *DROP TABLE*, 4) izvedejo se ukaz *CREATE TABLE* in telo se ponovno napolni z manj kot 10 vnosi. V tej podatkovni zbirki je orodje *bring2lite* uspelo obnoviti podatke in nenaslovljenega prosota in doseže boljše rezultate kot ostala orodja.

Tretja kategorija, označena z *0C*, vsebuje podatkovne baze ustvarjene na sledeč način: 1) ustvari se ena ali več tabel, 2) vstavijo 20 zapisov, 3) izbrišejo 5-7 vnosov. V tem je orodje *Deleted Records Parser* v enem primeru obnovilo več zbrisanih vnosev kot pa *bring2lite*, vendar je obnovilo je dele vnosov in ne vsega. V drugih primerih pa se je bolje odrezal *bring2lite* in je edino orodje katero v teh primerih prido-

Tabela 1: Vzorec rezultatov pri obnovitvi zbrisanih podatkov. Znak + pomeni, da so bili vsi zbrisani podatki obnovljeni, *0* pomeni, da so podatki za seboj pustili sled in jih je mogoče obnoviti in - označuje podatke katerih niso uspeli obnoviti in za seboj ne pustijo sledi.

Scenario	secure_delete=0 auto_vacuum=0 journal_mode=OFF	secure_delete=FAST auto_vacuum=0 journal_mode=OFF	secure_delete=1 auto_vacuum=0 journal_mode=WAL	secure_delete=1 auto_vacuum=0 journal_mode=PERSIST
1	+	-	+	+
2	+	-	+	+
3	+	-	+	+
4	+	-	+	+
5	+	0	+	+
6	+	0	+	+

bljene zapise obnovi v celoti.

V kategoriji *0D* je baza zgrajena na sledeč način: 1) ustvari se ena ali več tabel, 2) vstavi se 10 zapisov, 3) izbriše se 5-10 zapisov, 4) vstavi se 3-10 zapisov. Kategorijo smatrajo kot kritično, saj najbolje odraža dogodke podatkovne zbirke realnosti. V tem primeru *SQLite Deleted Records Parser* obnovi 44% zbrisanih zapisov ampak le delno, v nasprotju z *bring2lite*, ki obnovi 20% zapisov, kjer je le peščica teh popolnoma obnovljenih.

V zadnji kategoriji *0E* pa so baze zgrajene tako, da dosežejo dovolj vnosov za prelivne strani. V teh primerih orodje *bring2lite* obnovi 66% zbrisanih zapisov, od tega je polovica obnovljena v celoti. Drugo najboljše orodje je v tem primeru obnovilo 25% zapisov [6] [5].

9. NADALJNO TESTIRANJE

Na operacijskem sistemu Linux smo s programskim jezikom python ustvarili SQLite podatkovno bazo. Bazo smo napolnili z naključnimi brez pomenskimi vrednostmi. Na koncu smo podatkovno bazo izbrisali in skušali restavrirati z orodjem *bring2lite*, ki smo ga prenesli iz spletnega vira [1].

9.1 Gradnja podatkovne baze

Podatkovno bazo smo najprej ustvarili s programskim jezikom Python. Bazo smo ustvarili brez dodatnih nastavitev. Dodali nekaj naključnih tabel in jih zapolnili s podatki. Iz tabel smo nato izbrisali nekaj vrednosti ter nad podatkovno bazo pognali orodje `bring2lite`. Orodje ni našlo izbrisanih podatkov, kar najverjetneje pomeni, da je podatkovna baza SQLite zagotovili diskretnost izbrisanih podatkov, tako kot to zagotovi formatiranje diska kadar vse vrednosti bitov nastavi na 0 ali 1.

Naslednji korak je bil gradnja podatkovne baze z dodatnimi parametri, ki so našteti spodaj.

- **auto_vacuum=0**
- **secure_delete=0**
- **journal_mode=OFF**

[1]Vir orodja bring2lite: `https://github.com/bring2lite/bring2lite`

Podrobnejša predstavitev parametrov je predstavljena v tabeli 1. Ko smo ustvarili podatkovno bazo z zgornjimi nastavitvami ter ko smo ponovili vse zgornje korake, smo opazili, da je orodje *bring2lite* našel nekaj izbrisanih vrednosti. Bolj podroben opis korakov je predstavljen v podpoglavju 9.2.

9.2 Uporaba bring2lite orodja

Orodje *bring2lite* smo uporabili na 3 različne načine. Pri prvem načinu smo vnesli v tabelo nekaj vrednosti ter nato izbrisali prvi vnos. Ko smo pognali orodje *bring2lite* smo ugotovili, da je orodje našlo zapis, ki se je pred tem nahajal v podatkovni zbirki, vendar ta zapis ni bil restaviran v celoti, saj je bila informacija o identifikatorju (ID) izgubljena. Drug način, ki smo ga izvedli je vnos treh vrednosti v tabelo, izbris prvih dveh vnosev ter takoj zatem vnos četrtega zapisa - dolžina zapisov je bila enaka. Orodje *bring2lite* je v tem primeru restavriralo en vnos. Ta vnos je bil drugi od izbrisanih. Na ta način smo ugotovili, da SQLite prepiše vrednost izbrisanih polj in na ta način omogoči optimizacijo podatkovne baze. Tretji način, ki smo ga uporabili je vnos treh zapisov ter nato izbris dveh zapisov in takoj za tem še vnos enega zapisa - ki je bil dolg samo dve črki. Orodje *bring2lite* je restavirali en celoten zapis in drug zapis skoraj v celoti, pri čemer sta manjkali dve črki - vnos z dvema črkama je pobrisal informacije o predhodnem zapisu.

10. ZAKLJUčEK

V sklopu izdelave seminarske naloge, smo se ogromno naučili o delovanje podatkovne baze SQLite. Preiskali smo kopico orodij, ki omogočajo restavriranje podatkov. V sklopu same seminarske naloge smo tudi poskusili orodje *bring2lite* in skušali simulirati izbris podatkovne baze in nato restavriranje in iskanje izgubljenih podatkov. Ugotovili smo, da je v praksi pristop oz. algoritem, ki ga uporablja orodje *bring2lite* zelo koristen za podatkovne baze manjših velikosti. O količini restavriranja izbrisanih podatkov ogromno zavisi v odvisnosti od časa uporabe datotečnega sistema po dejanskem izbrisu - večja kot je podatkovna baza, bolj je izpostavljena možnim prepisom vrednostim na disku.

Tabela 2: Tabela razultatov pri testiranju različnih orodij. Znak * označuje delno obnovljene zapise.

primer/orodje	Undark	SQLite Deleted Records Parser	Sanderson Forensic Browser for SQLite	Sqlite Forensic Explorer	bring2lite
0A-01	20/20*	0/20	0/20	0/20	20/20
0A-02	9/20*	20/20*	0/20	0/20	1/20
0A-03	20/20*	0/20	0/20	0/20	20/20
0A-04	15/20*	10/20*	0/20	0/20	13/20
0A-05	11/20*	20/20*	0/20	0/20	3/20
0B-01	0/10	0/10	0/10	0/10	4/10
0B-02	0/10	0/10	0/10	0/10	4/10
0C-01	0/7	0/7	7/7	7/7	6/7*
0C-02	0/10	0/10	10/10*	9/10	8/10*
0C-03	0/7	7/7	2/7	4/7	6/7*
0C-04	0/10	10/10*	1/10*	8/10	8/10*
0C-05	0/10	10/10*	10/10*	9/10	10/10
0C-06	0/7	0/7	0/7	5/7	6/7*
0C-07	0/10	0/10	0/10	10/10	9/10*
0C-08	0/10	10/10*	0/10	6/10	7/10*
0C-09	5/10*	10/10*	0/10	2/10	0/10
0C-10	11/20*	20/20*	0/20	2/20	5/20
0D-01	0/5	2/5*	0/5	1/5	1/5
0D-02	0/5	1/5*	0/5	1/5	1/5
0D-03	0/5	0/5	0/5	1/5	0/5
0D-04	0/5	2/5*	0/5	0/5	1/5*
0D-05	0/5	0/5	0/5	0/5	0/5
0D-06	1/10*	5/10*	0/10	0/10	0/10
0D-07	0/5	5/5*	0/5	5/5	3/5*
0D-08	0/5	5/5*	0/5	3/5	3/5*
0E-01	3/7	2/7	3/7	0/7	5/7*
0E-02	0/5	0/5	0/5	0/5	3/5*
Vsota	95/278	139/278	33/278	73/278	147/278

11. REFERENCES

[1] Architecture of sqlite. `https://www.sqlite.org/arch.html`. Accessed: 2020-05-03.

[2] Database file format. `https://www.sqlite.org/fileformat2.html`. Accessed: 2020-05-03.

[3] Pragma statements. `https://www.sqlite.org/pragma.html`. Accessed: 2020-05-03.

[4] Sqlite3. `https://www.sqlite.org/about.html`. Accessed: 2020-05-03.

[5] C. Meng and H. Baier. bring2lite: A structural concept and tool for forensic data analysis and recovery of deleted sqlite records. *Digital Investigation*, 29:S31 – S41, 2019.

[6] S. Nemetz, S. Schmitt, and F. Freiling. pages S121–S130, 03 2018.

FbHash: shema za izračun podobnosti datotek v digitalni forenziki

Timotej Knez
Fakulteta za računalništvo in informatiko
Univerza v Ljubljani

Sebastian Mežnar
Fakulteta za matematiko in fiziko
Univerza v Ljubljani

Jasmina Pegan
Fakulteta za računalništvo in informatiko
Univerza v Ljubljani

POVZETEK

Algoritmi za detekcijo podobnih datotek pomagajo digitalnim forenzikom pri obdelavi velikih količin podatkov. V delu predstavimo algoritem za detekcijo podobnih datotek `FbHash`, opisan v članku [5] in nekaj njegovih predhodnikov. Predstavimo in implementiramo tudi svojo različico algoritma `FbHash`. Implementacijo testiramo na istih množicah datotek kot avtorji članka ter predstavimo naše ugotovitve. Rezultati eksperimentov se ujemajo z rezultati v članku, saj naša implementacija algoritma doseže F-score 94%.

Kategorija in opis področja

E.3 [**Data encryption**]

Splošni izrazi

Zgoščevanje

Ključne besede

Prstni odtis datoteke, Podobnostni izvleček, Zabrisano zgoščevanje, TF-IDF, Kosinusna podobnost

1. UVOD

Živimo v obdobju shranjevanja ogromnih količin podatkov. Pri forenzičnih preiskavah se pogosto zgodi, da je pridobljenih datotek preveč za ročno pregledovanje. Digitalni forenziki se tako soočijo s problemom avtomatizacije preiskave datotek. Možna rešitev so algoritmi za detekcijo podobnih datotek (angl. *Approximate Matching algorithms*), kot so `ssdeep`, `sdhash` in `FbHash`, ki poskusijo filtrirati vnaprej znane "slabe" oziroma "dobre" datoteke. Ti algoritmi ugotavljajo delež ujemanja datotek s pomočjo (nekriptografskih) zgoščevalnih funkcij. Algoritma `ssdeep` in `sdhash` lahko preslepi aktivni napadalec, ki pametno napravi majhne spremembe na določenih mestih datoteke. Učinkovitega napada na algoritem `FbHash` pa ne poznamo [5].

1.1 Prispevek članka

V našem delu predstavimo članek [5], v katerem avtorji predstavijo zgoščevalno funkcijo `FbHash`, ki omogoča lažje filtriranje "dobrih" in "slabih" datotek pri preiskovanju.

Glavni prispevki raziskave v [5] so:

- Predstavijo shemo za približno ujemanje, ki je odporno proti aktivnemu napadalcu.
- Predstavijo algoritma za zgoščevanje stisnjenih in nestisnjenih datotek, ki temeljita na shemi TF-IDF [12].
- Podajo analizo algoritma `FbHash` z drugimi algoritmi, ki rešujejo isti problem.
- Prikažejo, da iskanje podobnosti na nivoju zlogov ni dovolj pri stisnjenih datotekah.
- Naredijo analizo varnosti za `FbHash` in pokažejo, da je varen proti napadom z aktivnim napadalcem.

Poleg tega v našem delu algoritem ponovno implementiramo in preizkusimo več različic funkcij za uteževanje, ki se pojavijo v algoritmu. Rezultati naše implementacije algoritma se ujemajo z rezultati v [5]. Algoritem vedno zazna podobnost v datotekah z vsaj 5% ujemanjem. Ugotovimo tudi, da za dobre rezultate zadošča relativno majhna baza datotek.

1.2 Struktura članka

Naš članek je organiziran na sledeč način. V 2. poglavju predstavimo predhodnike algoritma `FbHash`. V 3. poglavju podrobneje predstavimo algoritem `FbHash` in našo implementacijo. V 4. poglavju povzamemo ugotovitve v delu [5]. V 5. poglavju opišemo izvedene eksperimente, v 6. pa predstavimo njihove rezultate. Narejeno delo povzamemo in zaključimo v 7. poglavju.

2. SORODNA DELA

Prvi algoritem namenjen iskanju približnih ujemanj je bil objavljen leta 2002 pod imenom `dcfldd`. Ta algoritem je razvil N. Harbour kot izboljšano verzijo ukaza `dd` [10]. Izboljšana različica tega algoritma je `ssdeep`. Pomembnejša predhodnika algoritma `FbHash` sta tudi `MRSH-v2` in `mvHash-B`. Obstaja še `bbhash`, ki pa je časovno potraten in ga ne bomo podrobneje opisali.

2.1 ssdeep

Algoritem `ssdeep` je implementacija kontekstno sprožene kosovno zgoščevalne funkcije (angl. *Context Triggered Piecewise Hash*, CTPH), ki jo je predstavil J. Kornblum septembra 2006 v raziskavi [11]. Algoritem temelji na detektorju neželene elektronske pošte `spamsum`, ki lahko zazna sporočila, ki so podobna znanim neželenim sporočilom.

CTPH uporablja zgoščevanje po kosih (angl. *piecewise hashing*), kar pomeni, da se zgoščena vrednost izračuna na posameznih delih datoteke fiksne dolžine. Za razliko od `dcfldd` algoritem CTPH uporabi poljubno zgoščevalno funkcijo.

Drugi princip, ki ga uporablja CTPH, je zgoščevalna funkcija z drsečim oknom (angl. *rolling hash*), ki preslika zadnjih k zlogov v psevdonaključno vrednost. Vsakega naslednika je tako možno hitro izračunati iz predhodno izračunane vrednosti. Pri tem je uporabljena zgoščevalna funkcija `FNV`.

Postopek CTPH se začne z izračunom zgoščenih vrednosti z drsečim oknom. Ob določeni sprožilni zgoščeni vrednosti (angl. *trigger value*) se vzporedno s tem sproži še algoritem zgoščevanja po kosih. Ob ponovni pojavitvi sprožilne vrednosti se dotlej zbrane vrednosti druge zgoščevalne funkcije zapišejo v končni prstni odtis. Tako se ob lokalni spremembi v datoteki sprememba pozna le lokalno tudi v prstnem odtisu.

Sledi primerjava prstnih odtisov datotek, ki temelji na uteženi Levenstheinovi razdalji (angl. *edit distance*), ki je nato še skalirana in obrnjena, da predstavlja 0 povsem različna prstna odtisa, 1 pa povsem enaka.

Algoritem `ssdeep`, ki je implementacija CTPH, se izkaže pri primerjavi podobnih besedilnih datotek in dokumentov [11]. Po drugi strani pa lahko aktivni napadalec popravi "slabe" datoteke na tak način, da se izognejo črni listi [5]. Ker je prstni odtis fiksne dolžine, je algoritem primeren le za relativno majhne datoteke podobnih velikosti.

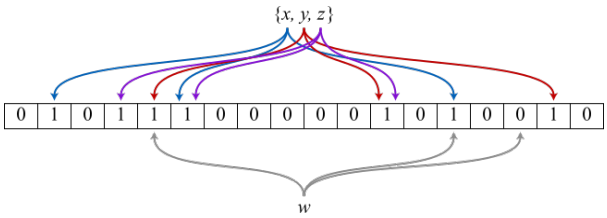
2.2 sdhash

Algoritem `sdhash` je opisal V. Roussev januarja 2010 v delu [13]. Glavna prednost tega algoritma pred predhodnimi je, da izbere statistično manj verjetne dele datotek kot izhodišče za računanje prstnega odtisa.

Postopek se začne z iskanjem statistično najmanj verjetnih delov datoteke. Izračuna se entropija skupin po k zlogov datoteke. Nato se izračuna rank vsake skupine glede na n sosednjih skupin. Izbrane so skupine, ki imajo rank večji ali enak postavljeni meji.

Sledi filtriranje skupin k zlogov, ki niso bistvene, povzročajo pa lažno pozitivne rezultate. Ocenili so, da je dobro zavreči skupine z oceno entropije pod 100 ali nad 990, ker so takšne skupine pogoste npr. v datotekah tipa JPEG.

Nato se generira prstni odtis datoteke kot zaporedje Bloomovih filtrov, ki so verjetnostni bitni vektorji, uporabljeni za prostorsko učinkovito predstavitev množic. Bloomov filter, ki predstavlja prazno množico, je vektor samih ničel. Ko dodamo element v tako predstavljeno množico, vzamemo fiksno število zgoščenih vrednosti elementa. Zgoščene vrednosti so definirane tako, da element preslikajo v pozicijo v vektorju. Nato dobljene pozicije v vektorju nastavimo na 1. Na sliki 1 vidimo, kako lahko preverimo, ali je nek element v Bloomovem filtru. Z več zgoščevalnimi funkcijami določimo pozicije v bitnem vektorju, ki morajo vse biti enake 1. Vidimo, da so x, y in z elementi filtra, w pa ne. Algoritem `sdhash` preveri za vsako izbrano skupino k zlogov, ali je že v množici, predstavljeni z Bloomovimi filtri. Če skupine ni v množici, jo algoritem doda.



Slika 1: Prikaz preverjanja, ali je element v Bloomovem filtru. Element w ni v filtru, ker zadnja zgoščevalna funkcija ne kaže na enico.

Nazadnje algoritem primerja prstne odtise datotek, torej zaporedje Bloomovih filtrov. Za vsak filter, ki predstavlja prvo datoteko, se izračuna maksimalna ocena podobnosti s filtri, ki predstavljajo drugo datoteko. Rezultat je povprečje tako pridobljenih ocen podobnosti.

Algoritem `sdhash` doseže boljša priklic in preciznost kot `ssdeep` [5]. A tudi ta algoritem ima več pomanjkljivosti: nekaterih datoteke ne more primerjati, primerjava datoteke same s seboj lahko vrne oceno med 50 in 100 ter prvih 15 zlogov sploh ne vpliva na končni prstni odtis. Poleg naštetega aktivni napadalec lahko spremeni "slabe" datoteke na tak način, da se izognejo črni listi oziroma "dobre" datoteke tako, da se obdržijo na beli listi [4].

2.3 MRSH-v2

Oktobra 2012 sta F. Breitinger in H. Baier predstavila algoritem `MRSH-v2` [3], ki se opira na predhodno razvit algoritem `MRSH` (angl. *multi-resolution similarity hashing*), ta pa temelji na algoritmu `ssdeep`.

Algoritem `MRSH` ima določene sprožilne točke $(-1) \bmod b$, kjer b pomeni povprečno velikost bloka. Namesto zgoščevalne funkcije z drsečim oknom uporabi polinomsko zgoščevalno funkcijo `djb2`, kot primitiv pa `MD5`. Namesto konkatenacije zgoščenih vrednosti `MRSH` kot prstni odtis uporabi seznam Bloomovih filtrov.

Algoritem `MRSH-v2` ponovno uporabi zgoščevalno funkcijo z drsečim oknom, kot `ssdeep`, namesto `FNV` pa uporabi funkcijo zgoščevanja `MD5`. Za večjo hitrost in v izogib napadu z dodajanjem sprožilnih točk je dodana tudi spodnja meja za velikost skupin zlogov $\frac{b}{4}$.

Algoritem `MRSH-v2` je po [3] časovno učinkovitejši od predhodnih algoritmov. Vključuje način za odkrivanje fragmentov in način za odkrivanje podobnih datotek. Po analizi leta 2014 [8], ki primerja `ssdeep`, `sdhash` in `MRSH-v2`, se v povprečju najbolje obnese `sdhash`, `ssdeep` in `sdhash` izkazu-

jeta dobro preciznost, vsi trije algoritmi pa imajo relativno slab priklic.

2.4 mvHash-B

Marca 2013 so F. Breitinger in sodelavci predstavili algoritem `mvHash-B` [2]. Ideja algoritma je, da majhne lokalne spremembe ne spremenijo končnega rezultata.

V prvem koraku se izvede večinsko glasovanje po bitih z nastavljivo mejo t. Vsakih k zlogov se tako preslika v ničle, če je število enic v zaporedju bitov manjše od t, sicer pa v enice.

Nato se zaporedje bitov zapiše na bolj kompakten način – enake zaporedne bite nadomestimo z dolžino takega niza. Če se zaporedje bitov ne začne z nekaj ničlami, bo prvo število v seznamu 0. Dobimo kodirano zaporedje dolžin nizov.

Kodirano zaporedje se razdeli v prekrivajoče se skupine fiksne dolžine. Te skupine so dodane v Bloomov filter. Nazadnje se primerja Bloomove filtre s pomočjo Hammingove razdalje, ocene pa se povpreči in odšteje od 100.

Algoritem `mvHash-B` naj bi bil hiter skoraj kot `SHA-1`, torej blizu zgornje meje učinkovitosti [2]. Vendar tudi ta algoritem ni varen pred aktivnim napadalcem, saj je možno popraviti "slabo" datoteko tako, da se izogne črni listi [6].

3. OPIS ALGORITMA

Avtorji v članku [5] predstavijo algoritem `FbHash-B`, ki je namenjen nestisnjenim (angl. *uncompressed*) in `FbHash-S`, ki je namenjen stisnjenim (angl. *compressed*) datotekam. Iskanje želenih datotek poteka na sledeč način. Najprej datoteke, ki jih želimo preveriti zgostimo s pomočjo ustreznega algoritma glede na njihov tip in tako dobim bazo podatkov. Nato zgostimo še ciljne datoteke in jih primerjamo z datotekami iz baze podatkov. Tako lahko datoteke, ki imajo s ciljnimi dovolj podobnosti zavržemo (angl. *whitelist*) oziroma dodamo med iskane (angl. *blacklist*).

3.1 FbHash-B

`FbHash-B` je bolj primeren za datoteke, ki niso stisnjene. V opisu algoritma bomo za boljšo preglednost uporabili sledečo notacijo:

- Del datoteke: niz k zaporednih zlogov.
- ch_i^D: del datoteke D, ki se začne na i-tem zlogu.
- Frekvenca dela oziroma $chf_{ch_i}^D$: število pojavitev dela datoteke ch_i v datoteki D.
- Datotečna frekvenca dela oziroma df_{ch}: število datotek, ki vsebujejo del datoteke ch.
- N: število datotek v bazi podatkov.
- $RollingHash(ch_i)$: vrednost rekurzivne zgoščevalne funkcije imenovane rolling hash na delu datoteke ch_i
- $chw_{ch_i}^D$: utež dela datoteke ch_i v datoteki D
- $docw_{ch_i}$: datotečna utež dela datoteke ch_i
- $W_{ch_i}^D$: ocena dela datoteke ch_i v datoteki D

Algoritem bomo predstavili v treh delih. V prvem delu datoteko razdelimo v dele, ki jih zgostimo z zgostitveno funkcijo in pretvorimo v utež glede na število njihovih pojavitev v datoteki. V drugem delu izračunamo datotečne uteži glede na to, v koliko datotekah se zgostitev dela pojavi. V tretjem delu pa iz datotečnih uteži in uteži dela izračunamo zgostitveno oceno, ki predstavlja datoteko.

3.1.1 Računanje frekvence delov datotek

V tem koraku se izračunajo frekvence delov datotek in njihove uteži. Vsaka datoteka vsebuje $N_D - k$ delov datotek, kjer je N_D dolžina datoteke D v zlogih in k parameter. Tako ima i-ti del datoteke D obliko $B_i^D B_{i+1}^D \cdots B_{i+k-1}^D = ch_i^D$, kjer je B_j^D j-ti zlog datoteke D.

Najprej izračunamo $RollingHash(ch_0^D)$ s formulo:

$$RollingHash(ch_0^D) = B_0^D \cdot a^{k-1} + \cdots + B_{k-1}^D \cdot a^0 \pmod n,$$

kjer so a, n in k parametri. Zaradi rekurzivne strukture funkcije $RollingHash$ lahko zgostitve preostalih delov izračunamo na sledeč način:

$$RollingHash(ch_{i+1}^D) = a \cdot RollingHash(ch_i^D) - B_i^D \cdot a^k - B_{i+k}^D \pmod n.$$

Parametre a, k in n izberemo na sledeč način. Za a izberemo neko konstanto med 2 in 255. Če za zalogo vrednosti funkcije RollingHash vzamemo 64-bitna števila, lahko k izračunamo na sledeč način:

$$B_i^D \cdot a^{k-1} \leq 2^{64} - 1.$$

Ker je maksimalna vrednost zloga in parametra a enaka 255 dobimo neenačbo $255^k \leq 2^{64} - 1$ iz česar sledi, da je k manjši ali enak 7. Tako za k izberemo 7. Če želimo zagotoviti, da ne pride do trkov med različnimi deli datotek, moramo za n izbrati praštevilo, ki je večje od $2^{56} = 256 \cdot 256^{k-1}$.

Strukturo datoteke lahko tako predstavimo kot razpršeno tabelo, kjer je ključ vrednost, ki jo dobimo s funkcijo $RollingHash$ na določenem delu datoteke, vrednost pa število pojavitev pripadajočega dela datoteke (oziroma vrednosti, ki smo jo dobili s funkcijo $RollingHash$).

Prvi korak zaključimo tako, da izračunamo uteži za dele besedila znotraj datoteke ($chw_{ch_i}^D$) s pomočjo ene izmed funkcij iz 3.4.

3.1.2 Računanje datotečnih uteži

Drugi korak je računanje datotečnih uteži delov besedila. Te nakazujejo koliko informacije prinese posamezen del datoteke znotraj določene baze podatkov.

Najprej je potrebno izračunati datotečno frekvenco za dele datotek, ki se v bazi podatkov pojavijo. To naredimo tako, da zgradimo razpršeno tabelo katere ključ je vrednost $RollingHash(ch_i)$, vrednost pa število pojavitev posameznega ključa. To s tabelo lahko sestavimo tako, da se sprehodimo čez vse ključe razpršenih tabel za posamezne datoteke in

zgoščevalni tabeli prištejemo 1, če se ključ v datoteki pojavi.

Iz frekvence posameznega dela nato izračunamo njegovo datotečno utež ($docw_{ch_i}$) s pomočjo ene izmed funkcij iz 3.4.

3.1.3 Računanje zgostitvenih ocen

V tretjem koraku izračunamo zgostitvene ocene za posamezne datoteke. S temi ocenami lahko kasneje datoteke med seboj primerjamo, kot je predstavljeno v 3.3.

V vsaki datoteki izračunamo oceno posameznega dela datoteke s formulo:

$$W^D_{ch_i} = docw_{ch_i} \cdot chw^D_{ch_i}.$$

Zgostitveno oceno za datoteko predstavimo kot razpršeno tabelo, kjer za ključ vzamemo vrednosti funkcije $RollingHash$, ki smo jih dobili v datoteki, vrednosti pa so ocene dela datoteke ($W^D_{ch_i}$).

3.2 FbHash-S

Med testiranjem različnih datotek so avtorji dela ugotovili, da algoritem `FbHash-B` na stisnjenih datotekah (tipov docx, pptx, pdf, zip ...) ne deluje najbolje. Tako datoteki, ki imata 90% podobnost v datoteki tipa docx, nimata veliko podobnosti na nivoju zlogov. Zato so razvili tudi verzijo algoritma `FbHash-S`, ki dosega dobre rezultate tudi na stisnjenih datotekah.

Algoritem `FbHash-S` deluje na sledeč način. Najprej datoteko razširimo (angl. *uncompress*) in uporabimo algoritem `FbHash-B` na vsaki izmed dobljenih datotek. Končna ocena pomembnosti je izračunana kot povprečje ocen posameznih datotek.

Zaradi dodatnega dela je algoritem `FbHash-S` časovno bolj zahteven od algoritma `FbHash-B`.

3.3 Primerjanje datotek

Podobnost med datotekama izračunamo s pomočjo kosinusne razdalje [9]. Najprej poskrbimo, da ima vsaka izmed datotek izračunano tabelo ocen pomembnosti. Podobnost med datotekama tako izračunamo s formulo:

$$Similarity(D_1, D_2) = \frac{\sum_{i=0}^{n-1} W^{D_1}_{ch_i} \cdot W^{D_2}_{ch_i}}{\sqrt{\sum_{i=0}^{n-1} (W^{D_1}_{ch_i})^2} \cdot \sqrt{\sum_{i=0}^{n-1} (W^{D_2}_{ch_i})^2}} \cdot 100.$$

Pri implementaciji lahko opazimo, da lahko za vsako datoteko vnaprej izračunamo $\sqrt{\sum_{i=0}^{n-1} (W^D_{ch_i})^2}$ in da je v števcu dovolj, da zmnožimo ocene pomembnosti, ki so v preseku ključev.

3.4 Uporabljene funkcije za uteževanje

Tukaj bomo predstavili različne funkcije, ki smo jih preizkusili za iskanje primernih uteži posameznih delov datoteke. Prve se uporabljajo za računanje uteži posameznega dela datoteke, druge pa za računanje datotečne uteži dela datoteke.

3.4.1 Funkcije za računanje uteži dela besedila

Najprej si oglejmo funkcijo za računanje uteži dela besedila, ki je predstavljena v delu [5]:

$$W^D_{ch_i}(chf^D_{ch}) = 1 + \log_{10}\left(\frac{chf^D_{ch}}{N^D}\right),$$

pri čemer N^D predstavlja število delov v datoteki, ki poskrbi za normalizacijo števila delov. Bolj kot je kos datoteke pogost, višja je njegova utež. Relativna frekvenca dela besedila je število med 0 in 1, torej je njegov logaritem med $-\infty$ in 1, saj je $\log_{10}(0)$ enak $-\infty$, $\log_{10}(1)$ pa 0. Menimo, da je to napaka (ali pa so avtorji članka uporabili drugačno funkcijo normiranja kot mi, saj je nikjer ne opišejo), saj v delu omenjata *TF-IDF* shemo, katera za izračun uteži pogosto uporabi formulo

$$\log\left(1 + \frac{chf^D_{ch}}{N^D}\right).$$

Zato smo uteži dela besedila izračunali nekoliko drugače.

Funkcijo smo popravili, da je rezultat število na intervalu med 0 in 1. Izračunamo relativno frekvenco dela in za 1 povečan rezultat logaritmiramo:

$$W^D_{ch_i}(chf^D_{ch}) = \log_2\left(1 + \frac{chf^D_{ch_i}}{N^D}\right).$$

Ker je ulomek povečan za 1 število na intervalu $[1, 2]$, smo vzeli logaritem z bazo 2. Tako kot utež dobimo število med 0 in 1.

Predlagamo še drugo funkcijo za računanje uteži, ki je razmerje med frekvenco dela besedila in številom takih delov besedila n. Tako je rezultat na intervalu $[0, 1]$:

$$W^D_{ch_i}(chf^D_{ch}) = \frac{chf^D_{ch_i}}{N^D}.$$

3.4.2 Funkcije za računanje datotečnih uteži

Prvo verzijo funkcije za izračun datotečne uteži dela besedila smo povzeli po raziskavi v [5]. Raziskava poda datotečno utež kot logaritmirano relativno frekvenco datotek, ki vsebujejo posamezen kos ch_i:

$$docw^D_{ch_i}(df_{ch}) = \log_{10}\left(\frac{N}{df_{ch}}\right).$$

Notranji ulomek je število med 1 in N, torej je rezultat število med 0 in $\log_{10}(N)$. Bolj kot je kos datoteke pogost, manjšo utež dobi.

Predlagamo še drugo verzijo funkcije za računanje datotečne uteži. Ulomek obrnemo in odštejemo od 1, da dobimo vrednost na intervalu $[0, 1]$, ki je še vedno obratno sorazmerna pogostosti dela datoteke:

$$docw^D_{ch_i}(df_{ch}) = 1 - \log_{10}\left(1 + \frac{df_{ch}}{N}\right).$$

4. UGOTOVITVE ČLANKA

Avtorji v delu svoj algoritem testirajo in pokažejo, da je varen proti aktivnemu napadalcu.

4.1 Varnost proti aktivnemu napadalcu

Napad z aktivnim napadalcem je napad, ko ima napadalec možnost spreminjanja datoteke. Varnost algoritma je prikazana tako, da opišemo znane napade na algoritme iskanja približnih ujemanj in podamo argumente, zakaj tak napad na `FbHash` ne deluje.

V članku [1] opišejo napad na `ssdeep` [11], ki datoteke spremeni tako, da se izognejo "črni listi". `Ssdeep` datoteko razdeli na 64 delov, ki jih zgosti s pomočjo kriptografske funkcije (naprimer md5). Tako je nastali podpis dolg 64 zlogov. Podpis datoteke na "črni listi" se ujema s podpisom datoteke, če se v njem ujema vsaj podniz dolg 7 zlogov. Želja napadalca je, da podpis povsem spremeni s čim manj spremembami v datoteki. To lahko naredi tako, da datoteko spremeni na način, da se vsak sedmi zlog spremeni ali pa vstavi delčke, ki zagotovijo, da se bo datoteka delila na različnih mestih in imela posledično različen podpis. Ta napad na algoritem `FbHash` ne deluje, saj se ob majhni spremembi datoteke frekvenca delov datoteke le malo spremeni.

V delu [4] prikažejo, da se lahko oceno podobnosti algoritma `sdhash` zlahka zniža pod 28%. Algoritem `sdhash` je razbit tudi v raziskavi v [7], ki pokaže, da lahko zgradimo več različnih datotek, ki bodo imele podobnost z podano datoteko enako 100%. Ta napada sta možna zato, ker je končna ocena sestavljena le iz dela datoteke. V `FbHash` je končna ocena sestavljena iz celotne datoteke, zato je vsaka sprememba opazna in ne prevelika.

Chang v delu [6] prikaže, da lahko napadalec algoritem `mvhash-v` prelisiči in dobi nizko oceno podobnosti z drugo datoteko tudi, ko sta datoteki zelo podobni. To je možno zato, ker `mvhash-v` uporabi funkcijo, ki podatke zgosti. `FbHash` zgostitvene funkcije ne uporabi, zato je na tak napad odporen.

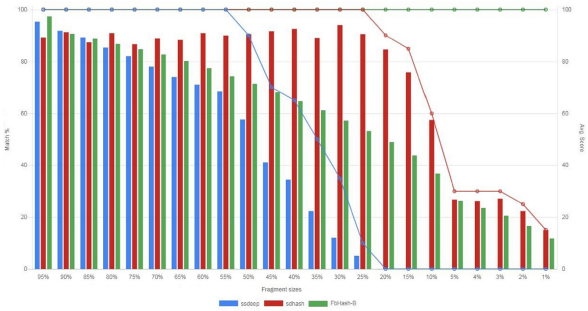
4.2 Testiranje algoritma

Algoritem `FbHash` je v članku primerjan z algoritmoma `ssdeep` in `sdhash`. Njegovo delovanje je primerjano po dveh metrikah: detekcija fragmentov in korelacija skupnega dela datoteke. Detekcija fragmentov nam pove, kako dobro algoritem zazna, da del datoteke (fragment) spada k določeni datoteki. Korelacija skupnega dela datoteke pa nam pove, kako dobro algoritem zazna, da dve datoteki vsebujeta enak del.

4.2.1 Detekcija fragmentov

Testi detekcije fragmentov so bili narejeni na bazi podatkov, ki je sestavljena iz fragmentov različnih velikosti in vzetih iz različnih mest v datoteki. Pri tem so uporabljene tekstovne datoteke in datoteke tipa docx iz [14].

Rezultati, ki jih avtorji dobijo na tekstovnih podatkih so prikazani na sliki 2. X-os grafa prikazuje velikost fragmenta, stolpci povprečno oceno podobnosti med delom in datoteko, točke povezane z daljicami pa prikazujejo procent pravilne korelacije dela z datoteko med poskusi. Vidimo lahko, da so rezultati algoritma `ssdeep` najslabši in da ocena podobnosti zelo pade, ko je fragment manjši od 50%. Za `sdhash` je opazno, da je ocena podobnosti skoraj v vseh primerih najvišja, a da pravilnost korelacije zelo pade, ko je del manjši od 15%. `FbHash` v vseh poskusih zazna, da sta del in datoteka korelirana, a je povprečna ocena podobnosti manjša od ocene algoritma `sdhash`.



Slika 2: Rezultati testiranja avtorjev na tekstovnih datotekah. Vir: [5]

Poleg tega preverijo natačnost algoritmov z metriko `F-score`, katere formula je:

$$F - score = 2 \cdot \frac{preciznost \cdot priklic}{preciznost + priklic},$$

kjer sta vrednosti *preciznost* in *priklic* izračunani s formulama

$$preciznost = \frac{TP}{TP + FP}, \quad priklic = \frac{TP}{TP + FN}.$$

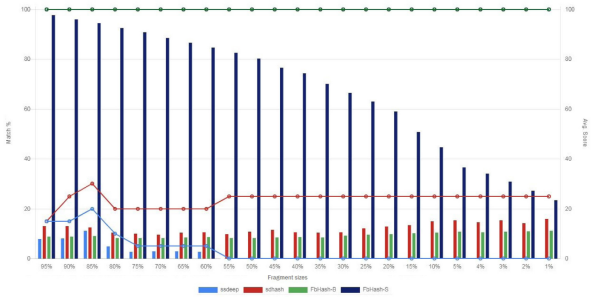
Tukaj TP predstavlja pravilno pozitiven primer, FP napačno pozitiven primer in FN napačno negativen primer. Na tekstovnih datotekah se je `FbHash` v vseh testih z metriko F-score izkazal najbolje, saj je imel rezultat med 94% in 98%. `Sdhash` in `ssdeep` pa sta imela rezultat 89,5% in 69%.

Rezultati testiranja na datotekah tipa docx so vidni na sliki 3. Vidimo lahko, da je tako kot na tekstovnih datotekah tudi tukaj `ssdeep` v obeh kategorijah najslabši in da ima `sdhash` višjo povprečno oceno kot algoritem `FbHash-B`, a del datoteke pravilno korelira z datoteko v manj primerih. `Sdhash` ima tukaj pri vsaki velikosti fragmenta podobno oceno (okoli 10%) in verjetnost korelacije (okoli 24%). Podobno je mogoče opaziti tudi v algoritmu `FbHash-B`, katerega ocena se giblje okoli 9% in verjetnost korelacije okoli 100%. Tako verjetnost korelacije ima v vseh primerih tudi algoritem `FbHash-S`, ki je bil narejen prav za datoteke tega tipa. To je vidno pri povprečni oceni, ki je pri vseh velikostih občutno višja od ocene drugih algoritmov.

Tudi mera `F-score` na datotekah tipa docx pokaže, da je `FbHash-S` za njih najbolj primeren. Ta ima oceno 92%, dočimer algoritmi `ssdeep`, `sdhash` in `FbHash-B` dosežejo 40%, 41% in 24%.

4.2.2 Korelacija skupnega dela datoteke

Testi korelacije skupnega dela datoteke so bili narejeni na bazi podatkov, ki vsebujejo datoteke s skupnimi deli različnih velikosti. Te datoteke so tekstovne ter datoteke tipa docx iz [14].



Slika 3: Rezultati testiranja avtorjev na datotekah tipa docx. Vir: [5]

Rezultati testov pokažejo, da se na tekstovnih datotekah `ssdeep` odreže najslabše in sicer z verjetnostjo $\geq 75\%$ zazna, da se datoteki ujemata, ko imata skupni del velikosti $\geq 30\%$, a ta verjetnost pri manjših delih hitro pade. `Sdhash` se bolje izkaže, saj zazna skupne dele, ki so veliki $\geq 3\%$ z verjetnostjo $\geq 93\%$. `FbHash` se izkaže najbolje, saj pade verjetnost, da zazna datoteki s skupnim delom pod 100% šele, ko imata skupen del, ki je velikosti le 1%. Takrat je verjetnost $\geq 93\%$.

Tudi na datotekah tipa docx rezultati testov pokažejo, da se `ssdeep` odreže najslabše. Ta ima največjo verjetnost, ko imata datoteki skupen del velikosti 10% in sicer 30%. Drugi velikosti skupnega dela pa dajo verjetnost $\leq 20\%$. Boljše rezultate doseže `sdhash`, ki ugotovi ujemanje datotek z skupnim delom velikosti 5% in 10% z verjetnostjo 90%. Zanimiv je rezultat funkcij `FbHash-B` in `FbHash-S`. Rezultat `FbHash-S` pokaže ujemanje 100% dokler imata datoteki skupnega več kot 2% in nato 90%. `FbHash-B` pa ugotovi, da imata datoteki skupen del z verjetnostjo 100% ko imata datoteki skupnega manj kot 40%. Ko imata skupnega več ta verjetnost pade na 80%.

5. NAŠI EKSPERIMENTI

Glavni cilj naših testov je bil ugotoviti, kateri izmed načinov določanja uteži dosega najboljše rezultate. V ta namen smo algoritma `FbHash-B` in `FbHash-S` implementirali [15], pripravili okolje za testiranje delovanja zgoščevalne funkcije ter preizkusili več načinov izračunavanja uteži.

5.1 Testiranje algoritma

Pri testiranju algoritma smo sledili postopku, ki so ga opisali avtorji dela [5].

5.1.1 Generiranje testnih primerov

Algoritem smo testirali na prosto dostopni bazi datotek [14]. Iz baze smo uporabili 100 besedilnih ter html datotek.

Testne primere za algoritem smo generirali tako, da iz zbirke vzamemo dve različni datoteki in naključen blok prvega vstavimo na naključno mesto v drugi datoteki na tak način, da je znan delež druge datoteke sedaj enak delu prve datoteke. Tako dobimo par datotek z natančno znanim ujemanjem.

5.1.2 Ocenjevanje delovanja algoritma

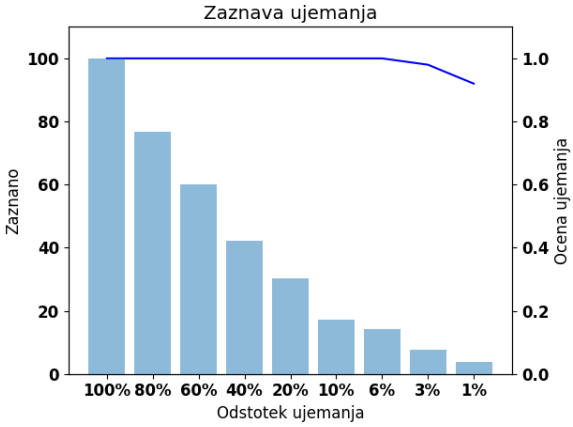
Da so naši rezultati čimbolj primerljivi s temi, ki so jih objavili v članku [5], smo tudi mi algoritem testirali na parih datotek z različnimi deleži ujemanja in opazovali dve vrednosti. Prva nam pove v kolikšnem deležu parov je algoritem odkril kakršnokoli ujemanje, druga pa predstavlja povprečno vrednost ujemanja, ki jo je algoritem izmeril. S tem lahko preverjamo kako majhne bloke besedila algoritem še prepozna in kako močna povezava obstaja med napovedano ter dejansko podobnostjo.

Tako kot v originalnem delu, smo se tudi mi odločili meriti vrednost F-score. Pri tem smo naključno generirali datoteke z različnimi merami ujemanja, med katerimi polovica datotek sploh ni imela nobenega ujemanja. Nato smo z algoritmom določili stopnjo ujemanja in opazovali, ali je ujemanje zaznal.

6. REZULTATI

Testirali smo več različnih funkcij za izračun uteži delov datotek.

Najprej si oglejmo delovanje funkcije, ki jo predlagajo avtorji članka [5] in je bila prva opisana v poglavju 3.4.1. Rezultati so predstavljeni na sliki 4. Algoritem s to formulo za uteži je dosegel F-score 94%, kar se ujema z vrednostjo, ki so jo dobili v članku.

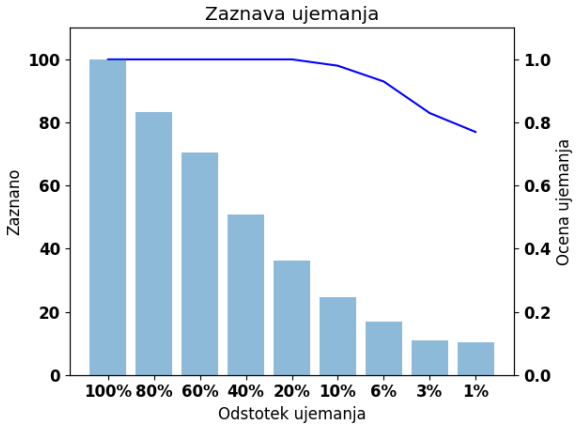


Slika 4: Rezultati testiranja z utežmi opisanimi v članku.

Rezultati, ki smo jih dobili se skladajo s tem, kar je predstavljeno v raziskavi v [5]. Vidimo lahko, da resnično ujemanje datotek ter izmerjena podobnost dobro sovpadata, vidimo pa tudi, da je algoritem v vseh primerih prepoznal delno ujemanje za vse dele datotek večje od 4% datoteke.

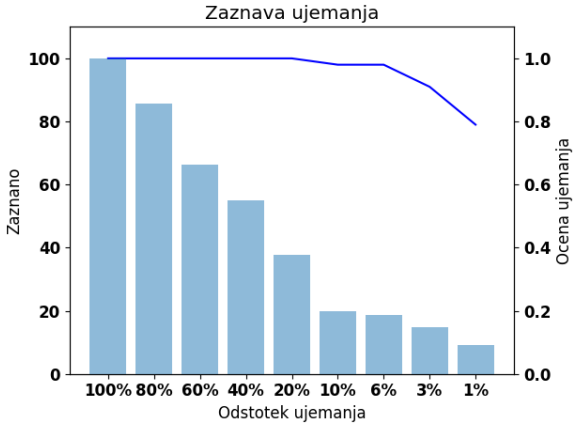
Ker se nam formula za izračun uteži ne zdi smiselna, smo jo nekoliko spremenili v obliko, ki je prav tako opisana v poglavju 3.4.1.

V tem primeru smo dosegli F-score 93%, kar je zelo blizu prejšnjega rezultata.



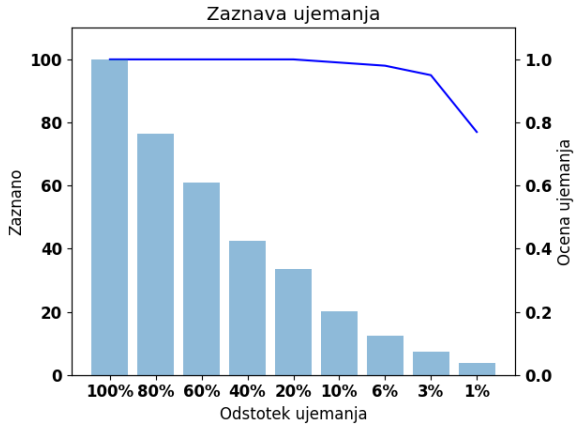
Slika 5: Rezultati testiranja s prilagojenimi utežmi.

Kot utež smo poskusili uporabiti tudi razmerje med frekvenco dela besedila in številom delov besedila. Tako smo ponovno dobili podobne rezultate (glej sliko 6). Pri tem smo izmerili F-score 92%.



Slika 6: Rezultati testiranja z razmerjem med frekvenco dela ter številom vseh delov

Pri teh testih smo opazili, da se rezultati pri različnih utežeh zelo malo razlikujejo, zato smo poskusili tudi, kaj bi se zgodilo, če bi začetno analizo zbirke datotek izpustili in kot uteži uporabili konstantno vrednost (glej sliko 7). V tem primeru smo dosegli F-score vrednost 87%. Pri tem smo ugotovili, da zaradi tega zaznava podobnosti datotek res deluje slabše, saj algoritem že pri 20% ujemanju datotek v nekaterih primerih ujemanja ni prepoznal, vendar pa povezava med napovedano in resnično podobnostjo še vedno ostaja dovolj močna. Ta rezultat pomeni, da bi lahko predlagani algoritem uporabili za primerjavo datotek tudi v primerih, kjer nimamo velike zbirke gradiva na katerem lahko izračunamo uteži.



Slika 7: Rezultati testiranja brez uporabe zbirke besedil.

Vprašali smo se tudi, kaj bi se zgodilo v primeru, da bi namesto konstantnih uteži uporabili naključne vrednosti. V tem primeru so bili rezultati podobni tisti, kjer smo uporabili konstantno vrednost, F-score vrednost pa je dosegla 89%.

7. ZAKLJUČEK

V delu smo opisali algoritem za detekcijo podobnih delov datotek `FbHash`. Algoritem smo implementirali z več različicami funkcij za uteženje posameznih delov datotek. Implementacijo smo testirali na istih množicah kot so jih uporabili avtorji algoritma. Naša implementacija je dosegla F-score 94%, kar je enako kot v članku [5]. Ugotovili smo, da je algoritem uporaben tudi z relativno majhno bazo datotek.

8. ZAHVALA

V tem delu smo se močno opirali na članek [5], zato se iskreno zahvaljujemo avtorjem članka za opravljeno delo in razvoj algoritma `FbHash`. Algoritem `FbHash` je pomemben prispevek digitalni forenziki, saj pohitri preiskovanje datotek in poskrbi, da aktivni napadalec ne more zakriti relevantnih datotek.

9. REFERENCES

[1] H. Baier and F. Breitinger. Security aspects of piecewise hashing in computer forensics. In *2011 Sixth International Conference on IT Security Incident Management and IT Forensics*, pages 21–36, 2011.

[2] F. Breitinger, K. P. Astebøl, H. Baier, and C. Busch. mvhash-b - a new approach for similarity preserving hashing. In *2013 Seventh International Conference on IT Security Incident Management and IT Forensics*, pages 33–44, March 2013.

[3] F. Breitinger and H. Baier. *Similarity Preserving Hashing: Eligible Properties and a New Algorithm MRSH-v2*. October 2012.

[4] F. Breitinger, H. Baier, and J. Beckingham. Security and implementation analysis of the similarity digest sdhash. In *First international baltic conference on network security & forensics (nesefo)*, August 2012.

[5] D. Chang, M. Ghosh, S. K. Sanadhya, M. Singh, and D. R. White. Fbhash: A new similarity hashing scheme for digital forensics. In *The Digital Forensic Research Conference*, volume 29, pages S113–S123. DFRWS, July 2019.

[6] D. Chang, S. Sanadhya, and M. Singh. Security analysis of mvhash-b similarity hashing. *Journal of Digital Forensics, Security and Law*, 11(2):2, 2016.

[7] D. Chang, S. K. Sanadhya, M. Singh, and R. Verma. A collision attack on sdhash similarity hashing.

[8] Frank Breitinger, Vassil Roussev. Automated evaluation of approximate matching algorithms on real data. volume 11, pages S10 – S17, April 2014. Proceedings of the First Annual DFRWS Europe.

[9] S. Gerard and B. Christopher. *Term-weighting approaches in automatic text retrieval*. 1988.

[10] N. Harbour. Dcfldd. defense computer forensics lab. *online*, 2002.

[11] J. Kornblum. Identifying almost identical files using context triggered piecewise hashing. volume 3, pages 91–97, September 2006. The Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS '06).

[12] J. Ramos. Using tf-idf to determine word relevance in document queries.

[13] V. Roussev. *Data Fingerprinting with Similarity Digests*, volume 337. September 2010. Advances in Digital Forensics VI. DigitalForensics.

[14] Roussev, V. The t5 corpus. `http://roussev.net/t5/t5.html`. Accessed: 2020-04-29.

[15] Sebastian Mežnar, Timotej Knez, Jasmina Pegan. df-seminarska, 2020. `https://github.com/jasminapegan/df-seminarska`.

4 Forenzika pomnilnika / Forensics of Memory

Live and Static Analysis on Leftovers from Tor Browser Usage

Mojca Kolšek
UL, Faculty of Computer and Information Science
Večna pot 113
Ljubljana, Slovenia
mk6614@student.uni-lj.si

Florentin Wieser
University of Passau
Innstrasse 41
94032 Passau, Germany
wieser14@stud.uni-passau.de

ABSTRACT

The Tor Browser bundle is often used for anonymous browsing. Apart from encrypting the messages and covering user's identity in the network, Tor Browser also aims to enable the user to remove the browser without leaving any traces [5]. In this paper, this ability was tested with two approaches; live analysis (analysis of a RAM dump remnants) and static analysis (analysis of traces left on the hard drive). Data leakage is, among other factors, dependant on the medium, where Tor Browser executable has ran from [4]. This paper presents the results of using the browser installed on a USB drive. Nevertheless, remnants of Tor Browser usage are found with both live and static analysis.

Keywords

Forensics, Tor, Tor Artifacts, live analysis, Volatility, static analysis, Autopsy.

1. INTRODUCTION

The Tor Browser Bundle provides a browser (built upon Firefox), that uses the Tor network. The server channels are encrypted, providing an additional security layer. Moreover, it routes the network packets through Tor servers to cover the real user's origin. The non-encrypted data only appears at the user's end. By using these additional routing steps, Tor prevents websites and institutions like an Internet Server Provider from tracking the user. This feature explains why "[...] journalists use Tor to communicate more safely with whistleblowers and dissidents" [1]. These people rely on the Tor browser and could face problems when browser usage is revealed. As the user's machine is a valuable source of information when conducting a forensic search of Tor Browser activity, possible remnants should be assessed.

We recreated the analysis done by Muir et al. in the article "A Forensic Audit of the Tor Browser Bundle" [4]. The authors analysed RAM dumps and machine snapshots at several points in time, thee being during and after the Tor Browsing activity. Their focus lies on the results of the analysis of artifacts found after the user logout. We have recorded a similar session (described in Section 2) of Tor Browsing. We focus on what information can be recovered when using the Tor Browsing Bundle installed on the external (USB) drive. The disk snapshot and RAM dump were obtained shortly after the Tor Browsing session was closed. Live and static analysis were carried out separately; the results are reported in Sections 3.1 and 4.1 respectively. Many artifacts produced by the operating system (OS) were recovered.

[1] `2019.www.torproject.org/about/overview.html.en`

2. TOR USAGE

In order to conduct the research, some browsing activity was performed. We ran Tor 9.0.7 in portable mode on a USB drive. The used OS was Windows 10 Enterprise, build 18362 (also known as 19H1). The following procedure was conducted on the machine:

1. Downloaded Tor installer to the USB drive from `torproject.org` with Edge,
2. Cleared history of Edge,
3. Rebooted,
4. Installed Tor Browser on the **USB drive**,
5. Deselect creation of Start Menu & Desktop shortcuts during installation,
6. Started Tor Browser from USB Drive,
7. Opened **finanzen.net** and searched for **"Snap Inc"** and selected first recommendation,
8. Search on **duckduckgo.com** for an image of a **"greyhound"** and save a file to **"greyhound.jpg"** on the USB drive,
9. Used Google scholar to search for **"forensics on android"** and opened the PDF **"Live Memory forensics on Android with Volatility"**,
10. Visited **youtube.si** and watched **"James Corden Challenges Usain Bolt to ALL the Games"** (ads and Recommendations were in German since end node was in Germany),
11. Closed Browser,
12. Formatted the USB Drive (not Quick Format).

Finally, the Random Access Memory (RAM) dump was acquired via VirtualBox `pgmphystofile` plugin [2]. For static analysis, a snapshot of the VM's disk was created with the Usage of *FTK Imager* [3].

3. LIVE ANALYSIS

Live analysis is conducted on volatile data such as network or RAM data, which frequently change and are influenced by the larger system. The acquisition time is of critical importance for the amount of information that can be retrieved. Software and hardware tools are available for acquisition of RAM dump [1] and are dependant of the target OS. Notably, both require the knowledge of the target OS, and while hardware solutions require a physical access of the machine, software tools have the disadvantage of changing the volatile data as it is a software itself, and thereby uses the RAM.

The RAM is written and read by the OS and thus not made with the intention of being human readable. Therefore parsing tools are necessary (articles [4, 8] use Volatility [6] framework on Windows 10). The parsing tools, as well as the amount of the acquired information, also depend greatly on the target OS. For optimization purposes, Microsoft has been creating files that cache more and more information about the user activity. As a result, third party software has less control over the information that is leaked and left behind on the machine. The amount of the acquired data also depends greatly on the size of RAM, OS efficiency and memory architecture of the machine (including a use of an external drive).

3.1 Results

The RAM dump was saved in *raw* file format that can be converted to *dmp* and examined by the open source Volatility [6] framework. The framework comes in a form of *plugins*, written in Python programming language and is available for Windows, Mac OS and Linux. The official release, however, supports a poor number of profiles. For Windows 10, 18362 build, one must download the source from the appropriate github branch. Many plugins are not implemented for certain profiles.

We employed the plugins used in [4]; *cmdline*, *dlllist*, *envars*, *pslist*, *pstree* and *shellbags*. Plugins *dumpfiles* and *timeliner* reported the same error. The error originated form the *psscan* plugin and has already been reported as a bug, but for a different profile [7]. The plugin *memdump* was therefore used to compensate and retrieve the memory for the process of interest.

Plugin *pslist* logged a 20 minute *firefox.exe* session, with PID of 5572 (figure 1), however, in contrast to [8], *dlllist* reports a missing Process Environment Block (PEB) (figure 2). As in [4], no environment variables, connected to the Firefox process, were found. Further on, the path of the DLLs were not found and the executables could not be retrieved using the *procdump* plugin. A DLL is a service, offered by Windows OS to third party software, that implements frequently used functions (i.e. a dialog box when downloading a figure) [9].

The existence of Tor executable is only referenced in Shellbags, which are used by Windows OS for smoother Graphical User Interface (GUI) experience [3]. The *shellbags* plugin reveals the path (figure 3) to the executable. In addition, it logged an access to two disks (C and D). The relevance to the Firefox process can only be tied by close timestamps.

The amount of data left behind is significantly lower in comparison to other processes (i.e. process with PID of 508), as seen from the output of *cmdline* plugin (figure 4). The child processes are also not listed, as seen in output of *pstree* plugin (figure 5).

The memory, reserved for the Firefox process, was dumped with *dumpmem* plugin. The dump was searched for the keywords *finanzen*, *duckduckgo*, *greyhound*, *forensics*. All searches produced relevant outputs, making the activity tracking possible even after user logout. Furthermore, the search for 'https://' gives a complete overview of the user activity. Interestingly, the recovered memory dump also contained a record of "https://www.torproject.org/", referencing Microsoft Edge browser. Most likely, this is a leftover from the Tor installation step. Furthermore, numerous data, found in memory dump of process 508 (i.e. paths to executables), was missing in Firefox memory dump.

4. STATIC ANALYSIS

Multiple other researchers were able to recover remnants from Tor usage. In this section, an overview over possible traces of Tor usage is provided.

The most detailed results were found by Muir, Leimich, and Buchanan in their paper 'A Forensic Audit of the Tor Browser Bundle' [4]. They conducted static analysis on a Virtual Machine snapshot after uninstallation of the Tor Browser on Windows 10.

Investigation of the snapshot revealed multiple traces of the previous browsing session and existence of the Tor Browser. Keywords of the browsing session were found in *NTUSER.DAT*. Special focus was put on the registry key `HKCU\Software\Microsoft\Windows\CurrentVersion\CloudStore\Store\Cache\DefaultAccount$$windows.data.taskflow.shellactivities`. It contained visited pages titles suffixed with " - Tor Browser" and the absolute path to the Tor installation directory. The authors found, that this key first appears in the Windows 10 anniversary Update. Another key of interest is `HKCU\Software\Microsoft\InternetExplorer\LowRegistry\Audio\PolicyConfig\PropertyStore\<value>` which contains the Tor installation path. According to Muir et al. this key is likely connected to streaming of audio [4].

These results are surprising, as one of Tor's design goals is to avoid the writing of browser activities on non-volatile storage [4].

[2] https://ired.team/miscellaneous-reversing-forensics/dump-virtual-box-memory

[3] https://accessdata.com/product-download/ftk-imager-version-4-3-0

[4] https://2019.www.torproject.org/projects/torbrowser/design/index.html.en

```
Offset(V)          Name        PID   PPID   Thds     Hnds   Sess  Wow64 Start                          Exit
------------------ ----------- ------ ------ ------ -------- ------ ------ ------------------------- -----------------------------
0xffffcd060a26b140 csrss.exe   508    484    12     0        1      0 2020-03-26 19:01:48 UTC+0000
0xffffcd060d5b9080 firefox.exe 5572   2356   0  --------     1      0 2020-03-26 11:12:27 UTC+0000   2020-03-26 11:47:07 UTC+0000
```

Figure 1: Volatility plugin *pslist*. The listing confirms a 25 minute Firefox session.

```
**************************************************************************
csrss.exe pid:    508
Command line : %SystemRoot%\system32\csrss.exe ObjectDirectory=\Windows SharedSection=1024,20480,768 Windows=On
SubSystemType=Windows ServerDll=basesrv,1 ServerDll=winsrv:UserServerDllInitialization,3 ServerDll=sxssrv,4
ProfileControl=Off MaxRequestThreads=16


Base                             Size          LoadCount LoadTime                       Path
------------------ ------------------ ------------------ ------------------------------ ----
0x00007ff6d20e0000             0x7000             0xffff 2020-03-26 19:01:48 UTC+0000   C:\Windows\system32\csrss.exe
0x00007ffb640c0000           0x1f0000             0xffff 2020-03-26 19:01:48 UTC+0000   C:\Windows\SYSTEM32\ntdll.dll
0x00007ffb60f70000            0x18000                0x6 2020-03-26 19:01:48 UTC+0000   C:\Windows\SYSTEM32\CSRSRV.dll
0x00007ffb60f50000            0x15000                0x6 2020-03-26 19:01:48 UTC+0000   C:\Windows\system32\basesrv.DLL
0x00007ffb60f30000            0x15000                0x6 2020-03-26 19:01:48 UTC+0000   C:\Windows\system32\winsrv.DLL
#editors note: deleted some lines
**************************************************************************
firefox.exe pid:   5572
Unable to read PEB for task.
```

Figure 2: Volatility plugin *dlllist*. Windows DLL's are frequently used by third party software.

```
**************************************************************************
Registry: \??\C:\Users\User\AppData\Local\Microsoft\Windows\UsrClass.dat
Key: Local Settings\Software\Microsoft\Windows\Shell\BagMRU\1\1
Last updated: 2020-03-26 11:12:25 UTC+0000
Value   Mru   File Name      Modified Date                  Create Date                    Access Date                    File Attr                  Path
------- ----- ------------- ------------------------------ ------------------------------ ------------------------------ -------------------------- ----
0       0     Tor Browser   2020-03-26 11:09:04 UTC+0000   2020-03-26 11:07:32 UTC+0000   2020-03-26 11:10:00 UTC+0000   DIR                        D:\.Þ€Tor Browser
**************************************************************************
```

Figure 3: Volatility plugin *shellbags*. Confirms access to the external drive D.

```
**************************************************************************
csrss.exe pid:    508
Command line : %SystemRoot%\system32\csrss.exe ObjectDirectory=\Windows
SharedSection=1024,20480,768 Windows=On SubSystemType=Windows ServerDll=basesrv,1
ServerDll=winsrv:UserServerDllInitialization,3 ServerDll=sxssrv,4 ProfileControl=Off
MaxRequestThreads=16
**************************************************************************
firefox.exe pid:   5572
```

Figure 4: Volatility plugin *cmdline*. No path for the Firefox process executable could be retrieved.

```
 0xffffcd060a2b8300:winlogon.exe                    588    484      5      0 2020-03-26 19:01:48 UTC+0000
. 0xffffcd060ce104c0:userinit.exe                  3924    588      0 ------ 2020-03-26 11:01:59 UTC+0000
.. 0xffffcd060ce130c0:explorer.exe                 3948   3924     62      0 2020-03-26 11:01:59 UTC+0000
... 0xffffcd06062ba080:OneDrive.exe                3448   3948     25      0 2020-03-26 11:02:21 UTC+0000
... 0xffffcd060d130080:VBoxTray.exe                5560   3948     10      0 2020-03-26 11:02:21 UTC+0000
... 0xffffcd060d0a3080:SecurityHealth              1012   3948      1      0 2020-03-26 11:02:20 UTC+0000
. 0xffffcd060c01b100:dwm.exe                        956    588     14      0 2020-03-26 19:01:48 UTC+0000
. 0xffffcd060a356280:fontdrvhost.ex                 736    588      5      0 2020-03-26 19:01:48 UTC+0000
 0xffffcd060a26b140:csrss.exe                       508    484     12      0 2020-03-26 19:01:48 UTC+0000
 0xffffcd060d5b9080:firefox.exe                    5572   2356      0 ------ 2020-03-26 11:12:27 UTC+0000
```

Figure 5: Volatility plugin *pstree*. No other processes were linked to the Firefox process.

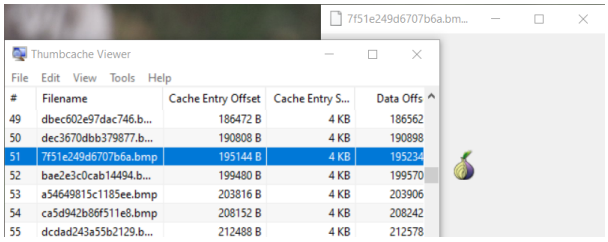


Figure 6: Tor icon found in the iconcache

Sandvik compared the Tor Browser Bundle on the three main operating systems Windows, Linux and Mac OS [5]. She used Windows 7, however some of the results may be reproducible on a Windows 10 machine.

The research resulted in three interesting places for static analysis. One is the `C:\Windows\Prefetch` folder which contained small files clearly proving the existence of the Tor Browser on a system. Another issue is the Thumbnail Cache which contained the Onion Logo icon. The cache is stored in the user folder under `AppData\Local\Microsoft\Windows\Explorer\iconcache_32.db`. Again, one registry file contained the path to the folder of the Tor installation: *UsrClass.dat* and the corresponding log file *UsrClass.dat.LOG1*.

Even though in this experiment, the Tor Browser was not used from a removable disk and the Windows version differs, it is worth to have a look in the mentioned places.

Two more forensic analyses of artifacts left by Tor Browser have shown that the registry is the most-likely place to find indicators for Tor usage. Jadoon et al. [2] and Warren [8] both found registry keys showing the installation directory of the Tor Browser.

4.1 Results

The static analysis of the hard drive disk image was conducted using Autopsy 4.14[5]. In a first step the previously found places of interest were checked and afterwards some additional work was done. The primary used tool was the 'keyword search' ingest module by Autopsy. The keywords consist of the words 'onion browser', 'Tor Browser' as well as terms related to the browsing activity.

Firstly, the thumbnail cache was extracted and then checked. The Tor Browser icon was found in the thumbnail cache confirming Sandviks [5] findings. In figure 6 the entry in the database and the respective icon can be seen. Secondly, the prefetch folder contains two files suggesting Tor usage: *TOR.EXE-3DFA27AC.pf* and *TORBROWSER-INSTALL-WIN64-9.0.-AEA0X9D6.pf*. Finally, the File *UsrClass.dat* was assessed. Again, the path to the Tor Browser installation folder was found. The indexed result found by with the keyword search could also be found in figure 7.

While Sandvik did the research on Windows 7 machines, it was possible to recreate said procedures on a Windows 10 machine.

[5] `https://www.autopsy.com/`

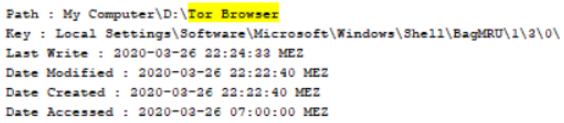


Figure 7: Key in UsrClass.dat showing the Tor installation path

In the next step, the often listed registry file *NTUSER.dat* was analyzed. Like in the previous registry file, the path to the browser is again found: `D:\TorBrowser\Browser\firefox.exe`. However, the *CloudStore\shellactivities* key was not found. It appears that this key was removed in the investigated build. In contrary, the second key concerning the *Audio policy config* still exists and again contains the path to the Tor install directory.

So far, we were able to prove the existence of the Tor Browser bundle on the machine from artifacts on various places. However, unlike Muir et. al. it was impossible to find evidence for actual browsing activity in the named places.

Therefore, additional analysis with Autopsy was done. The keyword search resulted two files with words connected to the browsing activity: `WebCacheV01.dat` and the respective log `V0100014.log`. During inspection, it became clear that the matches were on the urls 'finanzen.net', 'duckduckgo.com' and 'scholar.google.com'. However, *WebCache* contains a pile of popular websites – not only the ones visited during the browsing session. Of all the sites seen in figure 8, only 'scholar.google.com' actually was visited.

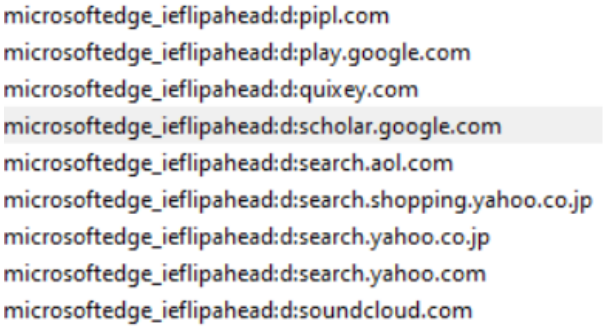


Figure 8: Popular websites found in Webcache.dat

Since the *shellactivities* registry is not used anymore [6], it must be assumed that static analysis can not determine the actual browsing activity. This thesis is supported as the keyword search did not find any results for the terms relating the actual performed actions like 'James Corden', 'Usain Bolt', 'Live Memory Forensics'.

[6] According to `https://kacos2000.github.io/WindowsTimeline/WindowsTimeline.pdf` the functionality moved to a database *Accessed on 30.04.20*

5. CONCLUSION

The live analysis alone did not provide necessary ties that would link the execution of Tor Browser on the machine; the Firefox process could not be tied to Tor Browser executable location or to any other processes. Only circumstantial evidence could be retrieved, using the time stamps of the Firefox process, assuming its connection to Tor Browser. Nevertheless, when coupled with static analysis, the Firefox process can be better tied to Tor Browser, gaining the time frame and the overview of the browsing activity, as static analysis recovered many relevant artifacts. Even though installation was only done on the thumb drive, multiple files could be found on the main drive. Our findings supports the thesis by Muir et al. [4], that the Tor Browser itself may be able to avoid writes to non-volatile memory, but the OS's routines undermine this effort.

References

[1] Kristine Amari. "Techniques and Tools for Recovering and Analyzing Data from Volatile Memory". In: *SANS* (Mar. 2009).

[2] Abid Jadoon et al. "Forensic Analysis of Tor Browser: A Case Study for Privacy and Anonymity on the Web". In: *Forensic Science International* 299 (Mar. 2019).

[3] Vincent Lo. "Windows ShellBag Forensics in Depth". In: *SANS Institute Information Security Reading Room* (Apr. 2014).

[4] Matt Muir, Petra Leimich, and William J. Buchanan. "A Forensic Audit of the Tor Browser Bundle". In: *Digital Investigation* 29 (2019), pp. 118–128.

[5] Runa A. Sandvik. "Forensic Analysis of the Tor Browser Bundle on OS X, Linux, and Windows". In: *Tor Tech Report* 2013-06-001 (June 2013).

[6] *Volatility.* `https://www.volatilityfoundation.org/`. Volatility Foundation.

[7] *Volatility - Issue #436.* `https://github.com/volatilityfoundation/volatility/issues/436`. Volatility GitHub.

[8] Aron Warren. "Tor Browser Artifacts in Windows 10". In: *SANS Institute Information Security Reading Room* (Feb. 2017).

[9] *What is a DLL?* `https://support.microsoft.com/en-us/help/815065/what-is-a-dll`. Microsoft.

HookTracer: Sistem za avtomatsko analizo zank aplikacijskih programskih vmesnikov

Vid Ribič Fakulteta za računalništvo in informatiko
Večna pot 113
Ljubljana, Slovenija
vr9223@student.uni-lj.si

Nejc Povšič Fakulteta za računalništvo in informatiko
Večna pot 113
Ljubljana, Slovenija
np9417@student.uni-lj.si

Luka Bezovšek Fakulteta za računalništvo in informatiko
Večna pot 113
Ljubljana, Slovenija
lb4263@student.uni-lj.si

POVZETEK

Zlonamerna programska oprema obstaja že od skoraj začetka računalništva in se skupaj z njim razvija. Odločili smo se, da bomo prebrali in poskusili implementirati rešitve članka, ki se ukvarja s programsko opremo, ki cilja t. i. API zanke (ang. "API hooks") in v nekaterih primerih teče v samem spominu [1]. Avtorji so v članku predstavili uporabo forenzike računalniškega spomina za zaznavo in analizo zlonamerne programske opreme ter predstavili način, kako bi lahko hitro in učinkovito ločili zlonamerne API zanke od legitimnih. Ustvarili so vtičnik *hooktracer* za orodje *volatility* [5], ki vse to omogoča, hkrati pa je hitrejši in zmogljivejši od že obstoječih rešitev. Vtičnik smo želeli preizkusiti tudi sami, a na žalost ni še na voljo za uporabo.

Ključne besede

API Hooks, Memory Forensics, Volatility

1. UVOD

Zaradi skokovitega pojava zlonamerne programske opreme, ki praktično ne koristi trajnega spomina v računalniški arhitekturi in deluje le na osnovi začasnega spomina, je digitalna forenzika na tem področju postala vsakdanja praksa v realnem svetu. Strokovnjaki za digitalno forenziko morajo z namenom odkrivanja tovrstnih zlorab izvajati kompleksne analize na podatkih, katerih obstoj ni trajen in se hranijo v delovnem pomnilniku. Na tržišču obstaja mnogo odprtokodnih rešitev, ki preiskovalcem nudijo orodja za izvajanje tovrstnih analiz, vendar pa od njih zahtevajo tudi veliko specifičnega znanja. Manj izkušeni strokovnjaki lahko kljub uporabi omenjenih orodij zlahka spregledajo nekatere zelo pomembne pasti, ki lahko škodujejo sistemu, mednje pa uvrščamo tudi t.i. aplikacijske programske vmesnike (ang. API Hooks). Gre za kompleksne programske strukture, ki spremljajo podatke, ki se pretakajo med sistemski funkcijami, še več, te podatke lahko tudi spreminjajo. Odkrivanje in analiza tovrstnih procesov je za mnoge preiskovalce danes še velika neznanka. V članku [1] so predstavljene teoretične osnove zank aplikacijskih programskih vmesnikov, predstavljena pa je tudi rešitev, ki so jo razvili in omogoča preiskovalcem avtomatizirano odkrivanje zlonamerne programske opreme te vrste.

2. ZANKE APLIKACIJSKIH PROGRAMSKIH VMESNIKOV

2.1 Vstavljanje kode

Zlonamerna programska oprema lahko s pomočjo zank aplikacijskih programskih vmesnikov popolnoma prevzame nadzor nad delovanjem sistema [1]. Tovrstna programska oprema vstavi škodljivo kodo v procese tako, da v aplikacijskem programskem vmesniku, ki ga kliče specifični proces, sorazmerno neopazno spremeni izvorno kodo [6]. Obstaja veliko različnih tehnik vstavljanja programske kode, na podlagi katerih deluje škodljiva programska oprema. Koda zlonamerne programske opreme je v tuje procese sistema vstavljena bodisi v obliki posameznih blokov kode (ang. shellcode) bodisi v obliki celotnih datotek programskih knjižnic (ang. DLL files) [1]. Večina vstavljenih struktur kode (bločnih ali programskih knjižnic), je danes v celoti implementiranih na način, da tečejo izključno v delovnem pomnilniku sistema, torej njihov obstoj ni trajen, zato jih morajo preiskovalci odkriti na nekoliko drugačen način [1]. Obstaja več različnih tehnik odkrivanja vrinjene kode v procese, večinoma z vtičniki, ki morajo biti predhodno nameščeni v sistem (npr. *malfind*, *messagehooks*, *eventhooks*). Ko je škodljiva koda enkrat vstavljena v ciljni proces, so zanke aplikacijskih programskih vmesnikov vstavljene v naslovni prostor omenjenega procesa. Na ta način lahko funkcije škodljive programske opreme nadomestijo originalne funkcije, implementirane v procesu. Poznamo dve obliki tovrstnega načina zamenjave programske kode.

2.2 IAT in EAT zanke

Prenosne programske datoteke (ang. Portable executable files, PE) se v operacijskem sistemu Windows lahko izvajajo kot samostojni programi [1]. Med prevajanjem izvorne kode ustvarjene prenosne programske datoteke definirajo, katere knjižnice in zunanje funkcije so potrebne za pravilno delovanje končne aplikacije. Ko operacijski sistem naloži aplikacijo, izvajalnik kode iz datotečnega sistema naloži vse specificirane knjižnice z uporabo vmesnika *LoadLibrary*, hkrati pa vsem funkcijam dodeli naslove za izvajanje s pomočjo vmesnika *GetProcAddress* [1]. Dodeljeni naslovi za izvajanje so zaradi optimizacije izvajalnega časa posameznih procesov shranjeni v vpoglednih tabelah (ang. lookup tables).

```
Hook mode: Usermode
Hook type: Import Address Table (IAT)
Process: 880 (svchost.exe)
Victim module: sppcomapi.dll (0x7fefac20000 - 0x7fefac5d0(
Function: slc.dll!SLGenerateOfflineInstallationId
Hook address: 0x7fefac695cc
Hooking module: sppc.dll
Disassembly(0):
0xfac695cc 48              DEC EAX
0xfac695cd 895c2410        MOV [ESP+0x10], EBX
0xfac695d1 48              DEC EAX
0xfac695d2 896c2418        MOV [ESP+0x18], EBP
0xfac695d6 56              PUSH ESI
0xfac695d7 57              PUSH EDI
0xfac695d8 41              INC ECX
0xfac695d9 56              PUSH ESI
```

Slika 1: *Primer zaznane IAT zanke v vtičniku Volatility apihook*

```
Hook mode: Usermode
Hook type: Inline/Trampoline
Process: 420 (IEXPLORE.EXE)
Victim module: mswsock.dll (0x71a50000 - 0x71a8f000)
Function: mswsock.dll!WSPStartup at 0x71a5c35b
Hook address: 0x27000a
Hooking module: <unknown>

Disassembly(0):
0x71a5c35b e9aa3c818e       JMP 0x27000a
0x71a5c360 81ec24010000     SUB ESP, 0x124
0x71a5c366 a12c72a871       MOV EAX, [0x71a8722c]
0x71a5c36b 8945fc           MOV [EBP-0x4], EAX
0x71a5c36e 8b4550           MOV EAX, [EBP+0x50]
0x71a5c371 53               PUSH EBX
0x71a5c372 8b               DB 0x8b

Disassembly(1):
0x27000a   e94b36ffff       JMP 0x26365a
```

Slika 2: *Primer zaznane medvrstične zanke v vtičniku Volatility apihook*

Naslovi funkcij, ki so v aplikacijo ali katero izmed knjižnic uvožene, so shranjeni v tabeli uvoženih naslovov (ang. Import Address Table, IAT). Naslovi funkcij, ki jih izvorna koda izvozi za uporabo znotraj drugih modulov, pa so shranjeni v tabeli izvoženih naslovov (ang. Export Address Table, EAT). Ob predpostavljanju obstoja omenjenih dveh tabel lahko škodljiva programska oprema z dostopom do leteh prepiše vnose v tabelah in nadomesti naslove funkcij z naslovi, na katerih se nahajajo zlonamerne funkcije. Po tovrstni zamenjavi naslovov v tabelah, so vsi klici funkcij, katerih naslovi so bili spremenjeni, pod nadzorom škodljive programske opreme.

Vtičnik *Volatility apihooks* tovrstne zanke zaznava tako, da najprej pregleda vse module (glavno aplikacijo in programske knjižnice) znotraj naslovnega prostora izvajanega procesa in se hkrati prepriča, da vsak naslov v uvoženi/izvoženi tabeli naslovov kaže na pravo funkcijo znotraj modula oz. zunanjo funkcijo, katere naslov je uvrščen na seznam zaupanih naslovov zunanjih funkcij. Če kateri izmed vnosov v tabeli kaže na neznani naslov, ki ga vtičnik ni našel niti na seznamu zaupanih naslovov, potem se tovrstni vnos tretira kot zlonameren.

Na sliki 1 je prikazan primer zaznane IAT zanke v vtičniku *Volatility apihook*. Označen je tip zanke (*Hook type: Import Address Table (IAT)*); proces, znotraj katerega je prišlo do zlorabe (*Process: 880 (svchost.exe)*); funkcija, katere naslov je bil preusmerjen (*Function: SLGenerateOfflineInstallationId*); modul, ki je odgovoren za preusmeritev (*sppc.dll*) in prvih nekaj vrstic ukazov, ki se nahajajo na preusmerjenem naslovu.

2.3 Medvrstične zanke

Medvrstične zanke (ang. trampoline hooks) delujejo tako, da s prepisom prvih nekaj vrstic kode v funkciji, preusmerijo izvajanje procesa na zlonamerne procese. V primerjavi s prej opisanimi procesi prepisovanja naslovov funkcij v tabelah, so tovrstni procesi bolj prikriti, saj je za njihovo odkrivanje potrebna natančna analiza vsebine posamezne funkcije, hkrati pa njihovo delovanje lahko vpliva na vse funkcije, ne samo na tiste, ki so neposredno uvožene ali izvožene v nek proces. Pri zaznavanju tovrstnih zank, si vtičnik *Volatlity apihook* pomaga s statično analizo. Najprej ta pregleda vse funkcije znotraj vseh naloženih modulov procesa, pri čemer prebere prvih nekaj ukazov v funkciji, in preveri, ali kontrolni tok (ang. control flow) morda zapusti obravnavano funkcijo. Če je tovrstna sprememba v kontrolnem toku dogodkov zaznana, bo vtičnik prepoznal medvrstično zanko in jo sporočil v obliki, ki je prikazana na sliki 2. Opisani postopek je primeren za zaznavanje večine medvrstičnih zank, redke izjeme, ki kodo prepišejo globlje v funkcijah, pa na ta način ne bodo zaznane.

3. SLABOSTI TRENUTNIH METOD FORENZIČNE ANALIZE SPOMINA ZA ZAZNAVO API ZANK

Današnji algoritmi za forenzično analizo spomina so dokaj uspešni pri zaznavi API zank, a vrnejo ogromno količino podatkov, ki jo tudi poznavalci le stežka obdelajo. Prav tako je za odkrito zanko težko trditi, ali jo je ustvarila zlonamerna ali legitimna programska oprema, saj za to potrebujemo ročni pregled vsake zanke in njene izvorne kode, kar je zelo časovno potratno.

3.1 Ločevanje legitimnih zank od zlonamernih

Prve metode forenzične analize spomina so bile razvite v času Windows XP, ko so bile skoraj vse zanke zlonamerne, saj jih operacijski sistem skoraj ni uporabljal. Takrat je bilo vsako odkritje zanke vredno pregledati, saj je bila zelo velika verjetnost, da ni legitimna. Danes je temu drugače, saj novejše verzije Windows uporabljajo API zanke za zagotavljanje združljivosti s starejšimi programi.
V tabeli 1 vidimo, kako se je število zank na vsaki verziji Windows povečalo. Vsak podatek je bil pridobljen z na novo naloženega operacijskega sistema v orodju VMWare Fusion, kjer se je uporabnik samo prijavil in odprl privzeti internetni brskalnik. Posnetek spomina so raziskovalci pridobili tako, da so zamrznili navidezni računalnik in analizirali *vmem* in *vmss* datoteki.
Kot lahko vidimo, je že z Windows 7 število zank tako naraslo, da je njihovo ročno pregledovanje skoraj nemogoče. Prav tako ne obstaja noben način, ki bi dovoljene zanke pravilno prepoznal brez zelo natančnega poznavanja delovanja operacijskega sistema. Število zank se zelo poveča tudi z uporabo antivirusnega programa, ki jih prav tako pogosto

Operacijski sistem	*Število API zank*
Windows XP	36
Windows 7	296
Windows 8	622
Windows 10	32.458

Tabela 1: Število zank v različnih verzijah operacijskega sistema Windows

uporablja.

3.2 Analiza zank zahteva ročni pregled

Ogromno število zank v novih verzijah Windowsa in sistemih, ki imajo naložen operacijski sistem, predstavlja velik problem za preiskovalce, saj obstoječe rešitve ne znajo dovolj dobro opredeliti, katere zanke so zlonamerne. Analiza zank je v času pisanja članka [1] zahtevala uporabo treh vtičnikov za Volatility: *apihooks*, *volshell* in *vadinfo*. Ta postopek omogoča zares globinsko analizo zank, a je zelo časovno intenziven, prav tako pa rezultate razumejo samo najbolj izkušeni programerji, ki zelo dobro poznajo delovanje operacijskega sistema.

4. AVTOMATIZACIJA ANALIZE ZANK

Z namenom avtomatske analize in filtriranja zank aplikacijskih programskih vmesnikov na osnovi netrajnega spomina so avtorji članka [1] razvili nov vtičnik Volatility, ki so ga poimenovali *hooktracer*. V tem poglavju bomo na kratko opisali delovanje vtičnika, kot so ga predstavili njegovi avtorji.

4.1 Zbiranje zank

Prvi korak pri delovanju vtičnika je postopek iskanja morebitnih zank aplikacijskih programskih vmesnikov v sistemu. Pri implementaciji te funkcionalnosti so si razvijalci pomagali z že obstoječimi tehnikami, ki jih nudi vtičnik *apihooks*. Faza zbiranja zank je po besedah avtorjev relativno dolgotrajen proces, saj ta zahteva pregled velikega števila funkcij v sistemu [1]. Po pregledu sistema se nabor odkritih zank pošlje na vhod vtičnika *hooktracer* v formatu JSON (Javascript Object Notation).

4.2 Analiza in simulacija delovanja zank

Odkrite zanke v naboru, je v naslednjem koraku potrebno analizirati. V okolju, ki posnema delovanje fizične strojne opreme, se posamezna zanka zažene, med samim izvajanjem procesa, pa se lahko odkrijejo tisti deli kode, na katere vpliva obravnavana zanka. Takšen proces, ki posnema vhodno/izhodne operacije nekega realnega sistema, imenujemo tudi emulacija (ang. emulation) [4]. Po besedah avtorjev, je uporaba emulacije v procesu avtomatskega iskanja škodljive programske opreme za razliko od obstoječe implementacije vtičnika *apihooks* bolj fleksibilna in bolj odporna na pasti, na katere lahko naletimo med procesom iskanja in analize. Veliko obstoječih vzorcev zlonamerne programske opreme je v obstoječi implementaciji namreč statično predefiniranih, kar pomeni, da tovrstna implementacija lahko najde samo omejeno količino škodljivih ukazov [1] .

Preden se lahko prične proces izvajanja zank, je potrebno ustrezno pripraviti okolje za emulacijo. Pri implementaciji emulacijskega okolja, so si avtorji vtičnika hooktracer pomagali z emulatorjem *unicorn* [3], ki je napisan v programskem jeziku python in poleg široke podpore mnogih platform in arhitektur, omogoča tudi popoln nadzor nad izvajanjem vtičnikov Volatility. Med pripravo okolja je pomembno, da razvijalec za vsak dogodek, kateremu želi slediti med izvajanjem, definira ustrezen povratni klic (ang. callback). Avtorji vtičnika hooktracer, so s pomočjo emulatorja unicorn omogočili sledenje naslednjim dogodkom med izvajanjem [1]:

- sledenje ukazom (ang. Instruction tracing),
- sledenje osnovnim blokom (ang. Basic block tracing),
- sledenje procesom branja in pisanja (ang. Memory reads and writes) ter
- sledenje dostopom do neveljavnih oz. praznih blokov spomina.

Enako kot izvajanje procesov v realnem okolju za izvajanje potrebujejo pomnilnik, le-tega potrebujejo tudi procesi, ki tečejo znotraj emulatorja. Emulator unicorn navideznega naslovnega proces ne določi samodejno, zato je naloga razvijalca, da poskrbi za določitev ustreznega naslovnega prostora in inicializacijo kazalca, ki kaže na prvo mesto ustvarjenega prostora [1].Pri uporabi dodeljenega pomnilniškega prostora med emulacijo je pomembno, da tisti deli pomnilnika, ki trenutno niso v uporabi ves čas ostanejo nedotaknjeni. Da se morebitni prej uporabljeni odseki ne bi nehote prepisali s kodo, ki se trenutno izvaja v emulatorju, se inicializirani sklad v sklopu navideznega spomina dodeli v odseke navideznega naslovnega prostora v jedru (ang. kernel virtual address space), saj uporabniška koda med izvajanjem nima dostopa do tega prostora, s tem pa je dostop za vse ostale procese, ki niso del našega emulacijskega okolja, onemogočen [1]. V vtičnik hooktracer, so avtorji implementirali tudi dodatni varnostni mehanizem, ki v primeru poskusa dostopa do naslovnega prostora jedra iz naslova, ki ne pripada naslovnemu prostoru procesa emulacije, zaustavi celotno izvajanje in s tem prepreči morebitne napačno pridobljene rezultate na koncu. Poleg pravilne oblike in vsebine navideznega sklada je med samim procesom emulacije potrebno paziti tudi na to, da se posamezna zanka izvede v celoti, saj se le tako lahko pravilno prepozna način delovanja le-te. S pomočjo nadzora dostopa do pomnilnika med procesom emulacije, ustrezni mehanizmi spremljajo operacije branja in pisanja iz začetnega naslovnega prostora v skladu, ki je na začetku prazen. Njihova naloga je, da tovrstne operacije dopuščajo samo funkcijam, ki tečejo znotraj emulacijskega okolja in hkrati skrbijo, da se procesi branja in pisanja izmenjujejo v pravilnem vrstnem redu [1]. To pomeni, da mora branju vrednosti iz začetnega naslova v skladu slediti operacija zapisa nove vrednosti. Če pred njo nastopi še ena operacija branja, se proces emulacije zaustavi. Hkrati je v vtičniku poskrbljeno tudi za primer, ko določen naslov zaradi specifičnih razlogov morda ni dosegljiv. V tem primeru se podatki, ki bi morali biti prebrani iz tega naslova "navidezno zapišejo"v ciljni naslovni prostor, sam proces emulacije pa lahko nemoteno teče dalje.

4.3 Zbiranje in analiza osnovnih blokov

```
2044 IEXPLORE.EXE     user32.dll!GetMessageA
PAGE_EXECUTE_READWRITE <Non-File Backed Region: 0x7ff80000
0x7ffadfff>
PAGE_EXECUTE_WRITECOPY \Device\HarddiskVolume1\WINDOWS\
system32\user32.dll (10)
PAGE_EXECUTE_WRITECOPY \Device\HarddiskVolume1\WINDOWS\
system32\ntdll.dll (5)
PAGE_EXECUTE_WRITECOPY \Device\HarddiskVolume1\WINDOWS\
system32\user32.dll (3)
PAGE_EXECUTE_READWRITE <Non-File Backed Region: 0x7ff80000
0x7ffadfff> (12)
PAGE_EXECUTE_WRITECOPY \Device\HarddiskVolume1\WINDOWS\
system32\kernel32.dll (3)
PAGE_EXECUTE_READWRITE <Non-File Backed Region: 0x7ff80000
0x7ffadfff> (2)
```

Slika 3: *Primer izpisa z uporabo vtičnika hooktracer. Vir: [1]*

Med procesom izvajanja neke zanke znotraj emulatorja unicorn se sproti proži povratni dogodkovni klic. Ta se izvede vsakič, ko je dosežen nov osnovni blok (ang. basic block). S tem izrazom označujemo osnovno enoto ukazov, ki se izvajajo zaporedno in nanje ne vpliva noben izmed pogojev znotraj izvorne kode (torej se izvedejo v vsakem primeru) [1]. Vtičnik hooktracer uporabi rezultate obravnavanega povratnega klica z namenom shranjevanja posameznih osnovnih blokov za nadaljnjo analizo.

5. ANALIZA Z VTIČNIKOM HOOKTRACER

Primer izpisa vtičnika hooktracer je na sliki 3. Vidimo, da je izpis zelo jasen in berljiv. Proces *IEXPLORE.EXE* s PID 2044 je vezan na funkcijo *GetMessageA* znotraj datoteke *user32.dll*. API je nato preusmerjen na tisti del spominskega sektorja (0x7ff80000), ki ni podprt z datoteko na disku, ampak se neposredno izvaja iz spomina (angl. non-file backed region). Poleg tega ima ta del spomina pravico branja, pisanja in izvrševanja. To je očiten namig, da je ta del kode kompromitiran. Dovoljenje vseh treh pravic kodi omogoča izvrševanje shell ukazov, zato je to pogosta praksa škodljivih programov.

Običajno je koda prepisana iz datoteke na disku in ni neposredno izvršena v spominu kot v tem primeru. Vendar iz tega ne gre prehitro sklepati, da so vse take vrstice (non-file backed region) tudi škodljive. Primer legitimne kode, ki se izvaja na tak način so antivirusni programi, saj se ti prav tako kot škodljiva koda želijo čim bolje skriti. Vtičnik *apihooks* tako potencialno detektira veliko število lažno pozitivnih primerov, ko preiskovalec išče zlonamerno kodo na disku. Tako si mora pri iskanju pomagati še z orodjem *volshell* in izvesti povratni inženiring, da ugotovi, ali gre za antivirusni program ali ne. Pri izpisu z vtičnikom *hooktracer* pa se vidi, da so po kodi, izvršeni neposredno v spominu, na vrsto prišle datoteke DLL znotraj antivirusnega programa (na sliki 4 je to AVG).

Hooktracer s pomočjo treh različnih filtrov preiskovalcu omogoča, da tako kodo izloči iz izpisa in ga tako naredi bolj berljivega.

5.1 Izločanje ukazov znotraj določene mape

Prvi filter (t. i. filter "All containing" oz. "Vsi, ki vsebujejo") izloči iz izpisa vse ukaze znotraj neke določene datoteke ali mape. Preiskovalec lahko na primer izloči vse ukaze znotraj

```
3068 iexplore.exe     ntdll.dll!LdrLoadDll at 0x776a22b8
PAGE_EXECUTE_READWRITE <Non-File Backed Region: 0x74c60000
0x74c6afff>
PAGE_EXECUTE_WRITECOPY \Device\HarddiskVolume1\
Program Files\AVG\Antivirus\snxhk.dll (2)
PAGE_EXECUTE_READWRITE <Non-File Backed Region: 0x74c60000
0x74c6afff> (46)
PAGE_EXECUTE_WRITECOPY \Device\HarddiskVolume1\
Program Files\AVG\Antivirus\aswhookx.dll (2)
PAGE_EXECUTE_READWRITE <Non-File Backed Region: 0x6f670000
0x6f67ffff> (4)
PAGE_EXECUTE_WRITECOPY \Device\HarddiskVolume1\Windows\
System32\ntdll.dll (2)
```

Slika 4: *Primer izpisa ob prisotnosti antivirusnega programa AVG. Vir: [1]*

mape System32, saj Windows sam preprečuje spremembo datotek DLL znotraj nje.

V članku [1] so avtorji pokazali, da so na ta način zmanjšali število ukazov iz 32.458 na samo 178, kar bistveno poenostavi brskanje po izpisu in iskanje zlonamerne kode. Samo z izločitvijo datotek iz System32 so izpis zmanjšali za več kot 99 %.

Nadaljnji pregled izpisa je pokazal, da se zelo pogosto pojavljajo ukazi vezani na programa Visual C++ in OneDrive. Ko so avtorji [1] dodali še poti do teh dveh map, je število izpisanih vrstic padlo na 0. Nobena izmed operacij ni zahtevala povratnega inženiringa, vsaka izvedba plugina *hooktracer* pa je trajala manj kot 30 sekund na tipičnem prenosnem računalniku.

5.2 Izločanje ukazov, pri katerih vsaj en ustreza podani poti

Drugi filter ("Any containing" oz. "Katerikoli, ki vsebuje") omogoča, da iz izpisa izločimo vse ukaze, pri katerih vsaj en spominski sektor ustreza podani poti. Npr. če imamo na operacijskem sistemu naložen antivirusni program kot v primeru zgoraj, lahko s tem filtrom izločimo vse ukaze, pri kateri se vsaj en del nanaša na mapo z njim.

Vtičnik *apihooks* je v [1] za čist operacijski sistem naštel 296 zank in 1625 po aktivaciji programa AVG. Z uporabo filtra *Any containing* za AVG in *All containing* za datoteke iz System32 je številka padla na 175, s filtriranjem ukazov za Internet Explorer in Visual C++ pa znova na 0.

5.3 Grupiranje množice ukazov

Tretji filter pa omogoča grupiranje množice ukazov med več procesi. Avtorji v [1] delovanje tega filtra pokažejo na primeru računalnika, okuženega z zlonamerno kodo *Zeus*.

Virus *zeus* je zelo agresiven in vstavi kodo v vse procese, do katerih ima pravico dostopa. Računalnik, okužen z *Zeusom*, smo želeli analizirati tudi mi. Kot že zapisano, vtičnik *hooktracer* še ni na voljo, tako da smo se lotili samo pregleda z vtičnikom *apihooks*. Postavili smo virtualno okolje z operacijskim sistemom Linux Debian, iz spleta pa potegnili zajem spomina z okuženega računalnika. Naložili smo potrebne knjižnice, med njimi tudi *volatility*, in izvedli analizo. Zaznali smo 14.024 okuženih zank oz. 23 funkcij na proces. Pregledati vse bi bilo praktično nemogoče, saj jih je bilo bistveno preveč. Razumevanje, kako deluje virus, pa bi-

```
1024 iexplore.exe
                ntdll.dll!NtCreateUserProcess
                ntdll.dll!ZwCreateUserProcess
                kernel32.dll!GetFileAttributesExW
                WININET.dll!InternetCloseHandle
                CRYPT32.dll!PFXImportCertStore
                USER32.dll!BeginPaint
                USER32.dll!CallWindowProcA
[hook listing truncated]

2468 explorer.exe
                ntdll.dll!NtCreateUserProcess
                ntdll.dll!ZwCreateUserProcess
                kernel32.dll!GetFileAttributesExW
                WININET.dll!InternetCloseHandle
                CRYPT32.dll!PFXImportCertStore
                USER32.dll!BeginPaint
                USER32.dll!CallWindowProcA
[hook listing truncated]

2564 conhost.exe
                ntdll.dll!NtCreateUserProcess
                ntdll.dll!ZwCreateUserProcess
                kernel32.dll!GetFileAttributesExW
                WININET.dll!InternetCloseHandle
                CRYPT32.dll!PFXImportCertStore
                USER32.dll!BeginPaint
                USER32.dll!CallWindowProcA
[hook listing truncated]
```

Slika 5: *Primer izpisa z uporabo vtičnika hooktracer. Vir: [1]*

stveno presega naše osnovno poznavanje področja. Analize s *hooktracerjem* nismo mogli izvesti, vemo pa, da bi trčili ob podoben problem. Brez uporabe filtrov bi vtičnik *hooktracer* našel veliko podobnih blokov izpisa, tako da bi bil pregled izpisa prav tako mukotrpen. Da pa lahko preiskovalec preskoči ročno pregledovanje vseh zlonamernih funkcij za vsak proces, lahko z uporabo filtra za grupiranje združi vse procese, kjer se je izvrševala ta koda. Preiskovalec vnese ID enega procesa in ime ugrabljene funkcije, s pomočjo katerih *hooktracer* samodejno poišče vse ostale procese in funkcije, ki sledijo istemu vzorcu. Vse zapise združi in jih prikaže v zapisu, kakršen je na sliki 5. Preiskovalci so na ta način dobili zelo jasen in zgoščen pregled vseh okuženih delov računalnika, prav tako pa tudi dober uvid v proces delovanja zlonamerne kode.

6. ZAKLJUČEK

Članek [1] nam je bil izjemno zanimiv, saj smo izvedli za nov način zlorabe računalnika, ki jo je zelo težko prepoznati. Na začetku smo povzeli delovanje zank aplikacijskih programskih vmesnikov in kako jih lahko zlonamerna programska oprema izkorišča. Nato smo opisali slabosti trenutnih metod forenzične analize spomina za zaznavo API zank in omenili naraščajoče število zank v operacijskih sistemih Windows, kar preiskovalcem otežuje pregledovanje in analiziranje zank. Nato smo opisali novejši vtičnik *hooktracer*, ki emulira API zanke v simulacijskem okolju in olajša delo preiskovalcem. Na žalost nam vtičnika ni uspelo preizkusiti, saj še ni na voljo za splošno uporabo. Smo pa videli, da se razvoj vtičnika nadaljuje. Maja 2020 je bila v [2] predstavljena razširitev *hooktracer_messagehooks*, ki uporablja *hooktracer* za detekcijo zank, ki jih izrabljajo programi za detekcijo tipkanja in prestrezanja gesel (angl. keylogger). Avtorji so v članku pokazali, da lahko s *hooktracerjem* detektirajo tudi najbolj sofisticirane aplikacije za nadzor uporabnikove aktivnosti pri uporabi tipkovnice, kot je program Turla.

7. LITERATURA

[1] A. Case, M. M. Jalalzai, M. Firoz-Ul-Amin, R. D. Maggio, A. Ali-Gombe, M. Sun, and G. G. Richard. HookTracer: A system for automated and accessible API hooks analysis. *Digital Investigation*, 29:S104–S112, July 2019.

[2] A. Case, R. Maggio, M. Firoz-Ul-Amin, M. M. Jalalzai, A. Ali-Gombe, M. Sun, and G. G. Richard III. Hooktracer: Automatic detection and analysis of keystroke loggers using memory forensics. *Computers Security*, page 101872, 05 2020.

[3] Nguyen Anh Quynh. Unicorn. https://www.unicorn-engine.org/.

[4] Rick Grush. Emulation and cognition. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.110.18

[5] The Volatility Foundation. The volatility framework. www.volatilityfoundation.org.

[6] H. Vipat and R. L. Sahita. Protecting IAT/EAT hooks from rootkit attacks using new CPU assists. Google Patents: https://patents.google.com/patent/US9037823B2/en.

5 Forenzika slik / Forensics of images

Iskanje izvora JPEG slik na podlagi informacij v glavi

Sandra Kerševan *
sandra.kersevan@student.fmf.uni-lj.si

Ada Šadl Praprotnik *
ada.sadl-praprotnik@student.fmf.uni-lj.si

Nika Kastelec *
nika.kastelec@student.fmf.uni-lj.si

Povzetek

Informacija o napravi oz. tipu naprave s katero je bila zajeta ali ustvarjena neka slika je lahko pomemben podatek v forenzični preiskavi. V članku se ukvarjamo z možnimi načini za pridobitev tega podatka za slike formata JPEG na podlagi podatkov, zapisanih v glavi slike. Na podlagi eksperimentov, ki so jih izvedli P. Mullan in sodelavci [4], pokažemo s kakšno natančnostjo je možno določiti znamko naprave za nek predhodno neznan model. Eksperimenti so bili izvedeni tudi na slikah, ki so bile predhodno obdelane z neko programsko opremo. Pričakovano se izkaže, da je natančnost v tem primeru precej slabša.

Ključne besede: forenzika slik, JPEG format, JPEG metapodatki, glava slike

1. UVOD

Digitalni forenzik mora pri svoji preiskavi pogosto pregledati veliko količino slik na nekem trdem disku. Poleg vsebine slike, datuma nastanka in drugih podobnih informacij je pomembna tudi informacija o napravi ali tipu naprave s katero je bila slika zajeta ali pa računalniško ustvarjena. V ta namen potrebuje metode s katerimi lahko dane slike hitro in zanesljivo razvrsti v neke skupine, recimo glede na to s kakšnim tipom naprave so nastale. Dva načina za iskanje 'izvora' slike sta recimo:

1. Na slikah iščemo sledi na nivoju slikovnih pik. Vsaka naprava ima namreč svoje lastne nepravilnosti, ki neizogibno nastanejo med njenim proizvajanjem in nato pri ustvarjanju slik pustijo sledi na nivoju pik. Za normalno delovanje naprave je to seveda nemoteče, je pa lahko v veliko pomoč pri forenzični obravnavi slik, ustvarjenih s to napravo. Večinoma imajo nepravilnosti naprav istega modela ali proizvajalca nekaj skupnega oz. so si zelo podobne. Prav to uporabimo za določanje tipa naprave, ki je sliko ustvarila.
2. Druga možnost je, da si ogledamo informacije v glavah slik. Tak tip pristopa je krajši oz. hitrejši, saj ne analiziramo celotne slike, ampak si ogledamo samo podatke v glavi. Lahko pa tu naletimo na težavo, če so bile slike pred našo analizo dodatno obdelane ali pa na primer deljene oz. objavljene na socialnih medijih. Informacije v glavi se pri tem lahko konkretno spremenijo.

V članku se ukvarjamo z drugim opisanim pristopom. Na to temo že poznamo nekaj metod, a te za svoje delovanje potrebujejo vnaprej podano natančno in zaključeno oz. dokončano bazo obstoječih modelov vseh (relevantnih) naprav. Tako bazo podatkov je zelo težko vzdrževati, saj na trg dnevno prihajajo novi modeli digitalnih kamer, fotoaparatov in mobilnih telefonov. Še več, na mobilnih telefonih lahko slike ustvarjamo tudi s pomočjo različnih aplikacij, ki pa se nenehno posodabljajo in nadgrajujejo. V nadaljevanju opišemo metodo za iskanje izvora slike, predstavljeno v [4], ki deluje tudi na nepopolni bazi podatkov o napravah. Pri tem želimo povezati nek predhodno neznan model z znamko.

Na podlagi omenjene metode so P. Mullan in sodelavci v članku [4] izvedli tri eksperimente na slikah formata JPEG, pridobljenih s spletne strani Flickr:

1. Klasifikacija razlik med informacijami zapisanimi v glavah slik, posnetih z napravami znamke Nikon in slik, posnetih z napravami znamke Canon.
2. Eksperimentalna ocena povezljivosti – kako natančno lahko določimo znamko naprave na podlagi modela, ki ni bil v učni množici.
3. Eksperimentalna ocena povezljivosti – kako natančno lahko določimo znamko naprave na podlagi glave slike, ki je bila obdelana z neko programsko opremo, ki ni bila v učni množici.

Za grupiranje slik glede na njihov izvor se uporabi naključne gozdove na JPEG metapodatkih in kvantizacijskih matrikah. V drugem poglavju podrobneje opišemo omenjeno metodo in predstavimo množico zajetih slik, nato pa v tretjem poglavju podamo opise in rezultate omenjenih treh eksperimentov.

1.1 Sorodna dela

Oglejmo si nekaj del, ki se ukvarjajo s podobno tematiko, kot naš članek.

*Fakulteta za matematiko in fiziko

- Članek [5] se prav tako ukvarja z iskanjem izvora slik formata JPEG na podlagi podatkov, zapisanih v glavi. Pri tem se osredotoči na mobilne telefone znamke Apple.

- Avtorji članka [3] se ukvarjajo z načini, kako ločiti slike glede na to, ali so bile narejene s fotoaparatom, telefonom, skenerjem ali pa je bila uporabljena računalniška grafika. Tehnike, ki jih predstavijo, se da tudi razširiti do te mere, da izvemo znamko in model naprave, s katero je bila slika narejena.

- V članku [2] lahko preberemo o uspešni tehniki za prepoznavanje ne samo znamke in modela fotoaparata s katero je bila slika narejena, temveč tudi točno določene naprave. Avtorji so delali s 320 slikami, posnetimi z devetimi različnimi napravami in skušali določiti, katera slika je bila posneta s katero napravo. Metoda temelji na tem, da ima vsak fotoaparat oz. digitalna kamera, ki se proizvede, na senzorju nekaj nepravilnosti. To se pozna na ravni slikovnih pik in na podlagi tega se lahko izračuna identifikacijski prstni odtis vsake naprave. Avtorji analizirajo tudi, kako se verodostojnost opisane metode spremeni, če delamo s slikami, ki so bile prej računalniško obdelane. Različni načini procesiranja slik, kot na primer kompresiranje in pretvarjanje formatov, lahko namreč sliko močno spremenijo.

- Članek [6] se ukvarja s slikami, posnetimi na mobilnih telefonih. Senzorji in slikovne pike fotoaparatov na mobilnih telefonih so manjše kot na digitalnih kamerah, zato potrebujemo drugačen pristop.

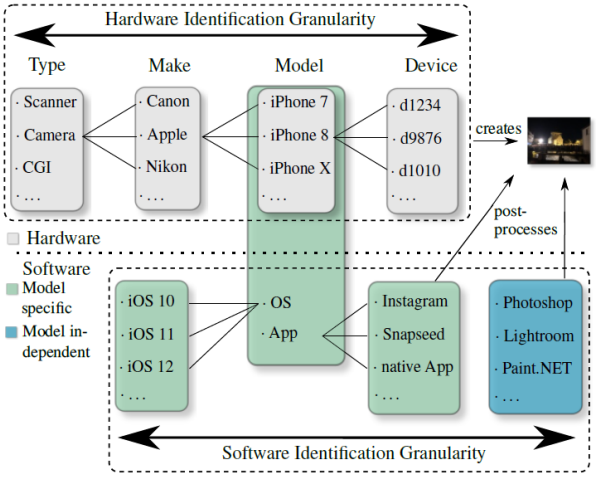
- Zanimiva je tudi raziskava, ki se ukvarja z unikatnostjo glav slik glede na posamezen model DSLR ali kompaktne kamere. O tem lahko preberemo v [1]. Izmed 1.3 milijona slik, ki so jih avtorji analizirali, jih je 62% imelo glave, ki so pripadale samo enemu specifičnemu modelu, skoraj pri vseh pa se je na podlagi glave dalo določiti proizvajalca.

2. REŠEVANJE PROBLEMA: HIERARHIJA ZNAMKA-MODEL-NAPRAVA

Zanima nas, kako povezati sliko, posneto z neznanim modelom kamere, s testnimi podatki. Če upoštevamo podatke v glavi, sta model in znamka del EXIF podatkov. Vendar se na to informacijo ne moremo zanesti, saj so ti zapisi nekonsistentni (različni zapisi, uporaba velikih in malih črk), manjkajo ali pa so popravljeni s programi za obdelavo slik.

Cilj je torej povezati podatke v glavi pod predpostavko, da so EXIF podatki o znamki in modelu nezanesljivi. Pri tem uporabljamo normalizacijo, da poenotimo zapise o znamki, modelu in verziji programov za obdelavo slik. Za izvedljivost problema predpostavimo, da je model kamere eden od elementov za identifikacijo kamere. Še nepoznan model ima lahko določene lastnosti, ki spominjajo na kateri drug model iste znamke (ponovna uporaba), vendar predvidevamo, da bo napoved še vedno izvedljiva.

Formalna predstavitev hierarhije za identifikacijo lastnosti kamere je predstavljena na sliki 1. Imamo strojno opremo (*hardware*), ki predstavlja tip slike, znamko, model in napravo, ter programsko opremo (*software*). Glavna razlika



Slika 1: Hierarhija za identifikacijo kamere.

med njima je, da se programska oprema spreminja z nadgradnjami, kar vpliva na kodiranje slike.

Cilj je pokazati, kako dobro lahko, z uporabo določenih informacij v glavi slike, povežemo neznan model kamere z znamko. To so P. Mullan in sodelavci [4] naredili v treh korakih. V prvem so pokazali, kako se med različnimi primeri slik razlikujejo podatki v glavi. Potem so pokazali, kako dobro lahko povežemo še nepoznane modele z znamkami in nazadnje, kako dobro lahko to naredimo, če so bile slike obdelane s programom.

2.1 Zbiranje podatkov in priprava

Zbranih je bilo 2 833 349 slik s strani Flickr, ki so bile naložene med leti 2008 in 2019. Obdržane so bile samo slike v veljavnem JPEG formatu in prebrane informacije iz glave slike. EXIF podatek o znamki in modelu je bil normaliziran z regularnimi izrazi, npr. "SAMSUNG" in "Samsung techwin co." zapišemo kot "Samsung", "EOD Rebel T7i" in "EOS Kiss X9i" zapišemo kot "EOS 800D". Ta podatek je privzet za resničnega. Odstranjeni so bili modeli in znamke s praznimi nizi, s pokvarjenim imenom ("cAnon") in vnosi, ki si nasprotujejo (model: iPhone, make: Canon).

Oznaka `EXIF: Software` je malo bolj zapletena. Na primer, Nikon uporablja to polje za vnos različice strojno-programske opreme (*firmware version*), nekateri programi, kot je Photoshop, pa zapišejo svoje ime. V podatkih glede na to polje ločimo "edited", če je bila slika obdelana s programom, in "unedited" sicer. Nato se polje še normalizira glede na verzijo programa, npr. "Adobe Photoshop CS6 (Windows)" in "Adobe Photoshop CS6 (Macintosh)" predstavljata isto verzijo.

Ker so v podatkih še vedno možne napake, so rezultati spodnja meja za primer popolnih podatkov. Skripta, ki omogoča prenos slik s strani Flickr je na voljo na `https://faui1-gitlab.cs.fau.de/mullanptr/flickr_data`.

2.2 Preučevanje

Standard za kompresijo JPEG datotek določa, kako stisniti sliko v bajte. Ta tok bajtov shranimo v datoteko. Najbolj uporaben je JFIF (*JPEG Interchange File Format*), zaradi metapodatkov pa je bil kasneje predstavljen EXIF (*Exchangeable Image File Format*), ki je v skladu s standardom TIFF (*Tagged Image File Format*) in dopušča izdelovalcu kamere in programov svobodo pri izbiri parametrov ter dodajanju dodatnih informacij v obliki metapodatkov.
V tem delu uporabimo dve skupini parametrov: dodatne informacije o podatkih pridobitve in barvnem profilu ter parametre za kodiranje. Te podatke preberemo direktno iz glave z ukazom `exiftool`.

EXIF skupine podatkov imenujemo IFDs (*Image File Directories*). Zanimajo nas naslednje skupine:

- `IFD0`: parametri slike (število vrstic in stolpcev),
- `IFD1`: thumbnail (sličica),
- `ExifIFD`: dodaten opis slike (hitrost zaslonke, velikost zaslonke, datum in čas zajema slike),
- `MakerNotes`: dodatne informacije vezane na proizvajalca naprave,
- `GPS`: kje je bila slika posneta,
- `ICCProfile`: *International Color Consortium* standard za predstavitev barv.

Tvorimo šest dimenzionalni vektor, ki vsebuje za vsako skupino število vnosov. Za parametre v zvezi z kodiranjem si pomagamo z JPEG kvantizacijskimi tabelami. Ta zagotovi kompromis med kvaliteto slike in njeno velikostjo. Proizvajalci in programi imajo po navadi svoje tabele, ki jih lahko preberemo iz glave brez dekodiranja slike. Matriko lineariziramo po cik cak vzorcu in vsak koeficient predstavlja eno lastnost, npr. matrika za svetlost je 64-dimenzionalni vektor.

2.3 Nadzorovano učenje

Označimo z X množico vseh primerkov. Vrstica v matriki predstavlja en primerek, stolpec pa določeno lastnost v iz množice vseh lastnosti V. Z I označimo množico znamk kamer, z J pa množico vseh modelov. Za vsako znamko $i \in I$ imamo množico razpoložljivih modelov j in velja, da za dve različni znamki ne obstaja model, ki bi jima bil skupen. Z f označimo funkcijo, ki bo predstavljala porazdelitev lastnosti v. Poskušamo dobiti mapiranje $i = f(x)$. Pri tem testne podatke predstavimo kot par (x, i).

Predpostavimo, da bodo modeli in znamke precej neenakomerno porazdeljene zaradi trga. Na takih vzorcih se naključni gozdovi izkažejo za zelo dobre. Gozd je sestavljen iz 100 dreves. Vozlišča dreves se razcepijo na podlagi 'gini impurity'. Ta meri, koliko so čisti razredi na podlagi določenih mej. Če vsi primerki spadajo v isti razred, je vrednost gini impurity 0 in vozlišče postane list. Drevesa bodo na koncu popolnoma razvejana, kar pomeni, da se končajo z listi. Za delo z odločitvenimi drevesi uporabljamo Python knjižnico `scikit`.

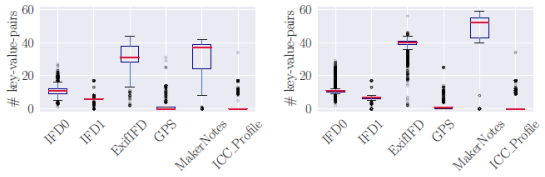
3. SPREMENLJIVOST INFORMACIJ V GLAVI

V tem razdelku bomo povzeli, kako so avtorji članka [4] potrdili svojo hipotezo, da so informacije v glavah fotografij med različnimi modeli iste znamke precej podobne, medtem ko se informacije v glavi pri različnih znamkah razlikujejo dovolj, da jih lahko ločimo med seboj. Predstavili bomo kako se te informacije od modela do modela spreminjajo, kar nam bo pomagalo razumeti klasifikacijsko napoved v nadaljevanju.

Leta 2011 je Kee [1] za DSLR in kompaktne kamere pokazal, da je veliko glav edinstvenih za specifični model, ni pa bilo še narejenih študij o specifičnih razlikah pri neujemajočih glavah in razlikah med modeli znotraj znamk in med različnimi znamkami. Spremenljivost informacije o glavi določa težavnost napovedi povezave med modelom in znamko.

3.1 Spremenljivost metapodatkov pri različnih modelih kamere

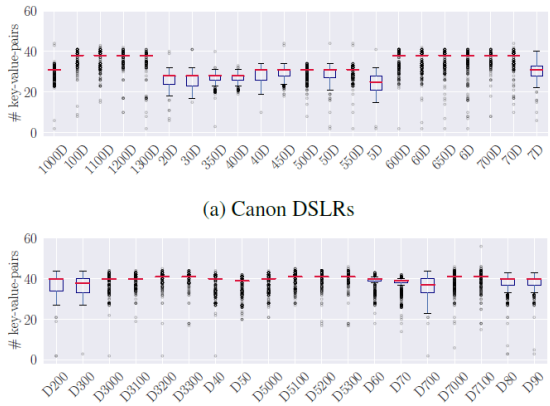
Za prvi eksperiment so avtorji izbrali 22 DSLR modelov Canonovih kamer in 19 DSLR modelov Nikonovih kamer. Za vsak model so izbrali naključnih 1000 slik, pri tem pa pazili da je bilo EXIF polje `Exif:Software` prazno in s tem zagotovili, da slike niso bile dodatno obdelane. Za te slike so prešteli število vhodov za posamezni metapodatkovni direktorij. Porazdelitev teh vrednosti je prikazana na sliki 2.



(a) Canon DSLRs (b) Nikon DSLRs

Slika 2: Skupno število parov (ključ, vrednost) v direktoriju za dve znamki. Direktorija `ExifIFD` in `MakerNotes` imata velike razlike med obema znamkama.

Škatle z brki so omejene s prvim kvartilom Q_1 in tretjim kvartilom Q_3. Manjše škatle prikazujejo šibkejšo porazdelitev, z rdečo črto pa je označena mediana. Za obe znamki je porazdelitev precej kompaktna pri direktorijih `IFD0, IFD1, GPS` in `ICC_Profile`. `ExifIFD` direktorij pa je na primer zelo kompakten pri Nikonovih modelih, ne pa tudi pri Canonovih. Pri obeh znamkah se razlikujeta tudi mediani števila vhodov pri `EcifIFD`. Porazdelitev pri Canonovih modelih se razprostira med 28 in 39 pari (ključ, vrednost), medtem ko je pri Nikonovih modelih porazdelitev kompaktna pri približno 40 parih (ključ, vrednost) z le malo osamelci. Grafa na sliki 2 nam povesta tudi, da obstaja pomembna razlika v `MakerNotes`. Spremenljivost torej obstaja tako znotraj znamk, kot tudi med različnimi znamkami, kjer se izkaže, da imajo Nikonovi modeli opazno več parov (ključ, vrednost) kot Canonovi. To opažanje obravnavamo kot prvi indikator za prepoznavanje razlik med znamkami. Slika 3 predstavlja po-



(a) Canon DSLRs

(b) Nikon DSLRs

Slika 3: Škatle z brki za število parov (ključ, vrednost) v direktoriju `ExifIFD` pri posameznem modelu.

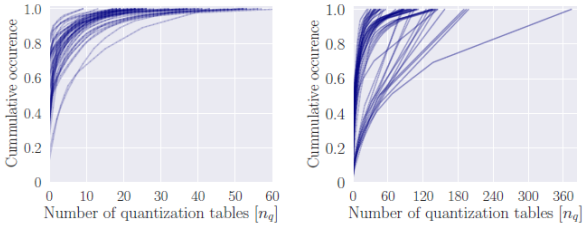
drobnejšo analizo znotraj direktorija `ExifIFD`. Škatle z brki pokažejo povezavo med modeli ene znamke. Pri Canonu obstaja skupina modelov, ki ima 38 parov (ključ, vrednost) pri direktoriju `ExifIFD`. Tudi kasnejši modeli Canonovih kamer so si podobni, ampak se število parov razteza med 20 in 30. Canonove kamere lahko na podlagi direktorija `ExifIFD` razdelimo v dve skupini. Nasprotno pa pri Nikonovih modelih opazimo manjša odstopanja mediane z vrednostmi med 29 in 41. Mediane se večkrat pojavijo na robu kvartilov, kar popači porazdelitve.

Ta študija pokaže, da s pomočjo informacij v glavi lahko ločujemo med znamkami, medtem ko se modeli razdelijo v več skupin znotraj znamke. Zato je bolj primeren pristop s kvalifikacijo modelov v razrede.

3.2 Spremenljivost kodiranih parametrov pri različnih modelih

Tukaj bomo opisali, kako so avtorji karakterizirali spremenljivost kvantizacijskih matrik JPEG formata. Za to so ponovno uporabili različne modele Canonovih in Nikonovih kamer. Izbrali so 50 različnih modelov in za vsakega od njih izbrali 100 do 1000 naključnih fotografij. Med izbranimi modeli so tako DLSR kamere kot tudi kompaktne kamere. Matrike za svetlost pri slikah istega modela so med seboj precej različne, ker so odvisne od trenutnih nastavitev kamere. Kumulativna porazdelitev kvantizacijskih matrik je predstavljena na sliki 4. Vsak graf je sestavljen iz 50-ih krivulj. Vsaka krivulja predstavlja kumulacijsko porazdelitev kvantizacijskih matrik za posamezni model. Število n_q predstavlja število kvantizacijskih matrik, ki so razporejena na x-osi, na y-osi pa imamo kumulativni procent slik z n_q matrikami. To pomeni, da če je krivulja blizu y-osi je večina kvantizacijskih matrik enakih.

Če primerjamo grafa Canonovih in Nikonovih modelov, opazimo, da veliko Nikonovih modelov uporablja občutno več kvantizacijskih matrik. Ugotovimo, da ima nek Nikonov model celo 375 različnih kvantizacijski matrik, medtem ko



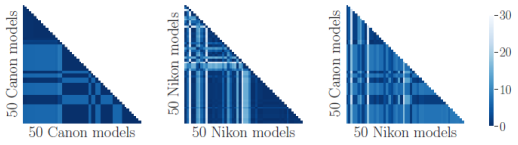
(a) 50 Canon Models

(b) 50 Nikon Models

Slika 4: Porazdelitev točnosti prepoznave znamke glede na eno od štirih kombinacij.

ima Canonov model z največjim številom teh matrik le 64 različnih. Nikonova modela, ki sta si po kvantizacijskih matrikah najbolj podobna, jih imata skupnih 38% medtem ko jih imata Canonova taka modela preko 80% ujemajočih se kvantizacijskih matrik.

V nadaljevanju prikažemo kvantitativne razlike med kvatizacijskimi matrikami znotraj in zunaj znamk. To so avtorji storili tako, da so za vsak model izbrali najbolj pogosto matriko svetlosti. V prvem eksperimentu so primerjali matrike svetlosti pri 50-ih Canonovih modelih, v drugem pri 50-ih Nikonovih, v tretjem pa so izbrali 50 enih in 50 drugih modelov. Rezultati teh treh eksperimentov so predstavljeni na



(a) RMSE among Canon models.

(b) RMSE among Nikon models.

(c) RMSE between Canon and Nikon.

Slika 5: Koren srednje kvadratne napake (RMSE) med kvantizacijskimi matrikami za svetlost med 50 Canonovimi modeli, 50 Nikonovimi modeli, in pari 50 Nikonovih in 50 Canonovih modelov.

sliki 5, kjer močnejša nasičenost prikazuje večjo podobnost med matrikami. Ti rezultati nam pokažejo, da je mogoče ločevati med znamkami na podlagi kvantizacijskih matrik.

3.3 Klasifikacija znamk neopaženih modelov

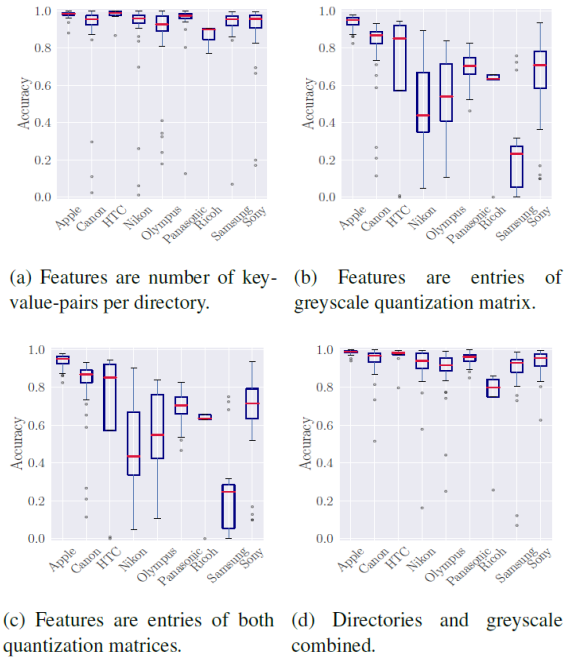
3.3.1 Nabor podatkov, učenje klasifikatorjev in meritve uspešnosti

P. Mullan in sodelavci so nabor podatkov iz prejšnjega razdelka tokrat razširili s podatki modelov 7-ih novih znamk. Da bi dobili čim bolj verodostojne rezultate so pri vsaki znamki analizirali 5 modelov, tj. toliko modelov, kolikor jih ima znamka z najmanj različnimi modeli. Ponovno so za vsak model naključno izbrali 100 do 1000 fotografij, ki niso bile računalniško obdelane.

Te podatke so razdelili v dve množici, učno in testno. Testna

množica vsebuje vse primere določenega modela znamke, učna pa vse primere preostalih modelov. Tako sta testna množica in učna množica disjunktni in klasifikator nikoli ne vidi primera na katerem ga testiramo med učenjem. Izredno naključen klasifikator je treniran na kombinaciji štirih funkcijskih vektorjev, ki so sestavljeni iz histograma metapodatkov, kvantizacijskih matrik svetlosti, koncentracije svetlosti in kvantizacijske matrike kromatičnosti ter koncentracije metapodatkov direktorija v kombinaciji s kvantizacijsko matriko svetlosti. Klasifikator na učni množici naučimo napovedati znamko kamere.

3.3.2 Rezultati in razprava



(a) Features are number of key-value-pairs per directory.

(b) Features are entries of greyscale quantization matrix.

(c) Features are entries of both quantization matrices.

(d) Directories and greyscale combined.

Slika 6: Porazdelitev točnosti prepoznave znamke pri upoštevanju različnih kombinacij lastnosti.

Na sliki 6 so od leve proti desni predstavljeni rezultati na podlagi ene od prej predstavljenih značilnosti. Na vsaki sliki so predstavljene škatle z brki za vsako znamko posebej. Le-te predstavljajo porazdelitev pravilnosti zaznavanja znamke. Pravilnost je pri prvem in zadnjem funkcijskem vektorju zelo visoka. Skoraj pri vseh znamkah je mediana pri 90% ujemanja. Ta dva eksperimenta sta nastala na podlagi metapodatkov direktorijev. Grafa na sredini, ki sta temeljila na podatkih kvantizacijskih matrik, pa prikazujeta nekoliko slabšo ujemanje med zaznano in dejansko znamko. V rezultatih vidimo, da sta modela Applove in HTC-jeve kamere dobro povezana s svojima znamkama le, če uporabimo metapodatke direktorijev. Najslabše je ujemanje pri Ricohovih modelih, a je tudi pri njih mediana na 90% ujemanja. Lahko povzamemo, da so vse znamke zelo dobro določljive, glede na izbrane podatke.

Po drugi strani pa je ujemanje glede na kvantizacijske matrike svetlosti precej slabo. Vse mediane so veliko nižje, kot če bi primerjali glede na metapodatke direktorijev. Najslabše ujemanje je pri Samsungovih napravah, kjer je mediana ujemanja pod 20%. Spet lahko opazimo da se Applove naprave najbolje določljive, ampak še vedno precej slabše kot na podlagi metapodatkov direktorija. Ujemanje je najslabše pri kombinaciji metapodatkov in kvantizacijskih matrik svetlosti in kromatičnosti, le pri Ricohovih in Samsungovih napravah opazimo, da je bilo ujemanje ob upoštevanju samo metapodatkov boljše.

3.4 Klasifikacija znamk na obdelanih fotografijah

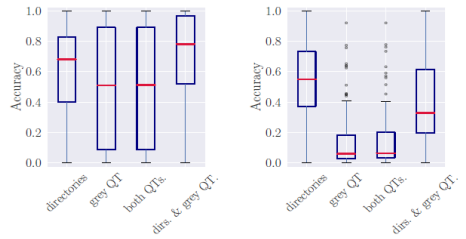
V praksi so fotografije velikokrat obdelane po njihovem nastanku s pomočjo aplikacij na pametnih telefonih oziroma računalnikih. Take obdelave pogosto vplivajo na glave fotografij, in s tem otežijo prepoznavanje modelov oziroma znamk kamer, s katerimi so bile narejene. Ogledali si bomo, kakšne rezultate so avtorji dobili pri eksperimentih s slikami, ki so bile obdelane.

3.4.1 Nabor podatkov, učenje klasifikatorjev in meritve uspešnosti

Tokrat so avtorji za učno množico vzeli kar vse zbrane podatke in jih prečistili tako, da so odstranili slike, ki kažejo znake o kasnejši obdelavi z zunanjo programsko opremo. Testiranje so nato izvedli na slikah, ki so imele v EXIF polju `EXIF:Software` ime orodja s katerim so bile spremenjene oziroma so bile naprave modificirane po nakupu.
Vse skupaj podatki obsegajo 121 namiznih programskih paketov in 27 aplikacij.

3.4.2 Rezultati in razprava

Najprej si poglejmo rezultate posebej za slike obdelane z namiznimi programskimi paketi, in nato še za slike obdelane z aplikacijami. Rezultati so predstavljeni na sliki 7. Rezultati



(a) Accuracy on images that are postprocessed by apps.

(b) Accuracy on images that are postprocessed by Desktop software.

Slika 7: Točnost prepoznave znamke pri obdelanih slikah.

so bili izračunani na podlagi istih 4-ih funkcijskih vektorjev kot prej. Mediane zavzemajo vrednosti med 50% in 75% za slike obdelane z aplikacijami in med 10% in 50% za slike obdelane z namizno programsko opremo. Verjetnost, da na pamet uganemo eno izmed devetih znamk, pa je 11,11%. V zgornji tabeli na sliki 8 vidimo aplikacije, ki ugotavljanje najbolj in najmanj poslabšajo. Opazimo, da imajo tudi aplikacije istega ponudnika precej različne vplive na sliko. Na primer, aplikacija Flicker zelo malo vpliva na prepoznavanje znamke naprave, ki je naredila sliko, medtem ko je iz

slike obdelane z aplikacijo Flicker za iPhone skoraj nemogoče prepoznati znamko naprave.

App

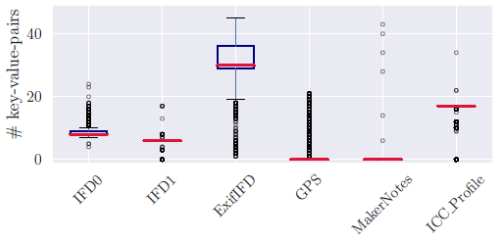
Rank	Software	Version(s)	Acc.
1	Flickr	1, 2, 6060, 3622	1.00
5	Mobile Fotos	_, 2, 1006	[0.97–1.00]
8	Instagram	_	0.95
9	Camera+	3, 6, 7, 9	[0.80–0.90]
13	PicsArt	_	0.80
14	VSCO	_	0.78
15	Camera+	5	0.77
	...		
23	Camera+	2863	0.43
24	Pixlr	_	0.40
25	Camera360	_	0.40
26	Snapseed	_	0.18
27	Flickr	for iPhone	0.00

Desktop

Rank	Software	Version	Acc.
1	Capture NX-D	1	1.00
2	Microsoft Windows Photo Viewer	6	0.93
3	Nikon Transfer	2	0.93
4	ViewNX-i	1	0.91
5	Phatch	_	0.90
6	Photoshop Express	3	0.88
7	DXO Optics Pro	v5	0.87
	...		
116	ACDSee Ultimate	10	0.08
117	Elements Organizer	14	0.07
118	ACDSee Pro	8	0.06
119	Capture One	8	0.06
121	Imagen digital ACD Systems	_	0.00

Slika 8: Primeri programske opreme so razdeljeni na aplikacije (zgoraj) in namizne programske oprem (spodaj). Navedeni so programski paketi, ki so bili najboljši in najslabši pri prepoznavi, skupaj z njeno natančnostjo.

Namizna programska oprema pa lahko prepiše glavne dele glave, zato je zaznavanje po takih procesih veliko slabše. Tak primer je program "Adobe Photoshop Lightroom 5". Slika 9 prikazuje porazdelitev parov (ključ, vrednost) za di-



Slika 9: Porazdelitev parov (ključ, vrednost) po obdelavi slike s programom "Adobe Photoshop Lightroom 5".

rektorij slik s tem parametrom na `EXIF: Software` polju. Po obdelavi slike s programom Adobe Lightroom, so vsi direktoriji `IFD0, IFD1, GPS, MakerNotes` in `ICC_Profile` virtualno enake velikosti, kot so bili. To pomeni, da Adobe Lightroom prepiše podatke v teh direktorijih. Zanimivo je, da ima direktorij `ExifIFD` večje medkvartilne razmike. To privede do hipoteze, da ta direktorij še vedno hrani nekatere originalne podatke, na podlagi katerih bi lahko prepoznavali znamko naprave, s katero je bila posneta slika. Kombinacija izbrisanih in ohranjenih podatkov bo lahko pripomogla pri določanju prstnega odtisa programskih orodij v nadaljnjih preiskavah.

4. ZAKLJUČEK

Raziskovali smo podatke o izvoru, ki se nahajajo v glavi JPEG slik. Glavni problem je ta, da je težko vzdrževati bazo modelov kamer aktualno, saj na trg neprestano prihajajo novi modeli, nadgradnje programske opreme pa spreminjajo karakteristike kamere. Rešitev je, da s pomočjo znanih modelov skušamo napovedati znamko kamere.

Predstavili smo rezultate eksperimentov, izvedenih na bazi slik s spletne strani Flickr. Eksperimenti so pokazali, kako se spreminjajo informacije v glavi slik med modeli ter s kakšno natančnostjo lahko klasificiramo znamko kamere. V primeru, da je bila slika obdelana z nekim programom, se izkaže, da če je sliko obdelala aplikacija, lahko z uporabo tako histograma kot kvantizacijskih matrik napovemo znamko z natančnostjo nad 75%. Namizni programi pa spremenijo velik del glave in je zato natančnost napovedi 55%. Z bolj dodelanimi funkcijami ali z uporabo nevronskih mrež bi bili rezultati lahko še boljši.

5. VIRI

[1] E. Kee, M. K. Johnson, and H. Farid. Digital image authentication from jpeg headers. *IEEE Transactions on Information Forensics and Security*, 6(3):1066–1075, 2011.

[2] J. Lukas, J. Fridrich, and M. Goljan. Digital camera identification from sensor pattern noise. *IEEE Transactions on Information Forensics and Security*, 1(2):205–214, 2006.

[3] C. McKay, A. Swaminathan, Hongmei Gou, and M. Wu. Image acquisition forensics: Forensic analysis to identify imaging source. In *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 1657–1660, 2008.

[4] P. Mullan, C. Riess, and F. Freiling. Towards open-set forensic source grouping on jpeg header information. *DFRWS EU – Digital Forensics Research Workshop EU*, 2020.

[5] P. Mullan, C. Riess, and F. C. Freiling. Forensic source identification using jpeg image headers: The case of smartphones. *Digital Investigation*, 28:S68–S76, 2019.

[6] B. Wronski, I. Garcia-Dorado, M. Ernst, D. Kelly, M. Krainin, C. K. Liang, M. Levoy, and P. Milanfar. Handheld multi-frame super-resolution. *ACM Transactions on Graphics*, 38(4):1–18, Jul 2019.

Copy-move Forgery Detection in Digital Images

Jošt Gombač
Faculty of Computer and Information Science
University of Ljubljana
Večna pot 113, 1000
Ljubljana, Slovenia
jg1278@student.uni-lj.si

David Penca
Faculty of Computer and Information Science
University of Ljubljana
Večna pot 113, 1000
Ljubljana, Slovenia
dp5306@student.uni-lj.si

Matej Koplan
Faculty of Computer and Information Science
University of Ljubljana
Večna pot 113, 1000
Ljubljana, Slovenia
mk7509@student.uni-lj.si

ABSTRACT
Copy-move forgery is a method of concealing or manipulating information of an image, done by copying and pasting an area of the same image. Commercialization and ease of use of the tools that enable such processes has made image tempering more widespread, whilst making detecting such forgeries harder for the human eye. In this paper, we describe the methods that are used in detecting copy-move forgery and provide a more detailed overview of the DCT, SVD, SURF based methods.

Categories and Subject Descriptors
D.2.10 [**Software Engineering**]: Design—*Methodologies*; I.4.7 [**Image Processing and Computer Vision**]: Feature Measurement

Keywords
Copy-move forgery detection, Image forgery detection, Image forensics, DCT, SURF, SVD

1. INTRODUCTION
Copy-move forgery is a process of copy-pasting of an area within the image used to conceal or modify objects in the frame. An example of concealing is shown in figure 1. Proving the authenticity of images is nowadays increasingly harder and more so important because of the availability of image processing tools. Commercialization of such tools has made manipulating the image's content more accessible and easier. Consequently, we are more likely to be exposed to tampered photos without our knowledge. The real problem arises when these photos become viral on social media where people tend to share content with certain shock value. Such spreading of misinformation can easily sway people's beliefs. Detection of such images is therefore a necessity to stop malicious intent of spreading false information.

Copy-move forgery detection is a part of *blind* or *passive* [20] image forensics. The goal of such methods is determining the authenticity of an image without using any embedded security information. These methods rely heavily on statistical values that copy-move forgery introduces.

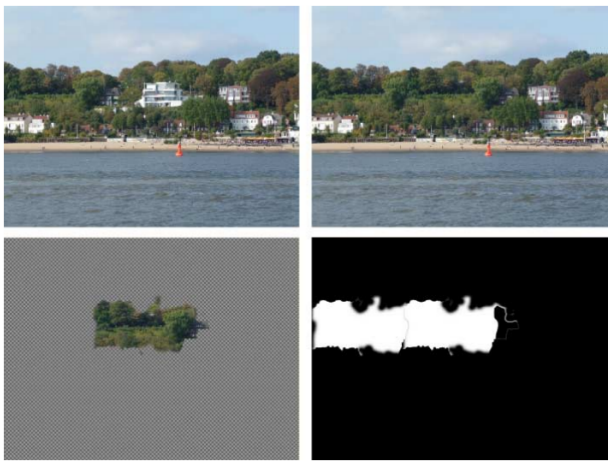


Figure 1: Example of image forgery [7]. Upper-left is the original, upper-right is the forgery, bottom-left is the copied area, bottom right is mask of the forgery.

In our paper, we first describe a general flow of copy-move forgery detection algorithms. For each approach, we provide its advantages and disadvantages. We then proceed with a description of more specific approaches and alternative methods. In conclusion, we provide metrics regarding each approach based on their feature extraction method.

2. AREA OVERVIEW
Vincent et al. [7] define a typical pipeline in 6 steps: preprocessing, two mutually exclusive processing steps, feature extraction, matching, filtering, and post-processing. Warif et al. [20] describe it in 4 stages: pre-processing, feature extraction, matching, and visualization. In our paper, we describe the first definition as it is more granular and is more descriptive of the actual procedure. We omit the visualization stage, as it is used for human interpretation of the results only, and adds no value to the algorithm itself.

2.1 Pre-processing

Authors of [20] have found, that most methods first transform image color space. Grayscale has a reduced dimensionality but proved to hold sufficient information to identify pattern matching. In some cases, grayscale images even improve algorithm prediction. This is due to color channel merging which reduces a chance of small imperfections hiding the copied regions. Methods that do not utilize grayscale transformations, rely on original color information or use other color spaces (Wandji et al. used YCbCr in [19]) where they rely on the intensity of color pixels, patterns in an image or other spatial information.

2.2 Block subdivision

Finding the copied regions could be done using an exhaustive search. Because this is computationally costly, most methods divide the image into overlapping [19] blocks. Block sizes have a significant effect on algorithm performance. Larger blocks will increase the computational complexity, while smaller blocks might produce too many false positives [10]. These sizes give varying result accuracy, which is why it is worth running this procedure multiple times and compare the results manually.

2.3 Keypoint detection

Some methods use keypoint detection [22] and omit image subdivision altogether. This method relies on copied regions having a high entropy. This statistical analysis is used to deduce important parts of the image. They then compute features for these regions i.e. keypoints only in order to reduce unnecessary comparisons in the matching state and reduce post-processing steps. The main drawback of such an approach is, that it could not determine the right keypoints. This can occur due to copied regions not exhibiting enough recognizable patterns or having repetitive content [7].

2.4 Feature extraction

In order to compare the image regions, its feature vectors must first be computed. There are multiple methods that can be used and are mainly divided into five categories based on:

Frequency

Wandji et al. [19] propose using discrete cosine transform coefficients (DCT) which proved to be resistant to JPEG compression, blurring, Gaussian noise, and rotations up to 5%. Warif et al. report DCT being the most used and advantageous approach to computing feature vectors. Bashar et al. [2] propose the use of discrete wavelet transform coefficients (DWT) which has low computational complexity, but relies on proper block subdivision, as it might miss duplicated areas if they appear in between divided blocks. Bayram et al. [2] recommend using Fourier-Mellin transform (FMT) which is robust against slight rotations, but fails if copied areas are scaled.

Dimensionality reduction

Christlein et al. [7] report dimensionality reduction based approaches having an advantage in its high performance. Bashar et al. [2] explore the usage of Kernel-PCA, a variant of principal component analysis (PCA) proposed by Popescu et al. in [17]. Zhao et al. [21] report singular value decomposition (SVD) as a robust approach against multiple image distortions, but not reliable with JPEG compression.

Intensity

Luo et al. [14] compute feature vectors based on average values of RGB color channels in addition to directional block information. Lin et al. [12] propose using average grayscale intensity values of blocks and their sub-blocks.

Moment

Mahdian and Saic [15] used blur-invariant moments. Zernike moments used by Ryu et al. in [18] excels in its performance, but have problems detecting scaled areas.

Keypoint

Most used keypoint based feature extraction is scale-invariant feature transform (SIFT) used by Zhou et al. in [22]. Speed-up robust features (SURF) introduced by Bay et al. [3] is an improvement of SIFT. Mishra et al. [16] proved that its speed performance increased, but its accuracy decreased.

2.5 Matching

Each feature vector must be compared with others in order to determine their similarity. This is typically measured using the euclidean distance. Each encounter of a similar enough (determined by a user-defined threshold) regions points to a plausible duplication.

Methods that use block subdivision often employ lexicographic sorting ([2],[6],[22],[17],[19]) before comparing each block pair in order to reduce time complexity. This is performed by placing each feature vector as a row in a matrix which is then row-wise sorted. In this case only neighbouring rows are required to be compared.

Some methods (typically keypoint-based) employ the Best-Bin-First search algorithm based on a variant of the kd-tree search algorithm as proposed by Zhou et al. [22]. Christlein et al. [7] report that in most cases, this type of matching results in a better accuracy but has its drawbacks in its space complexity.

2.6 Filtering

Found matches are filtered to reduce false positives. Two neighboring pixels oftentimes have similar characteristics and are therefore falsely detected as duplicates. To avoid such occurrences, authors in [15] remove detected regions that are spatially too close.

2.7 Post-processing

Copied and pasted regions should exhibit common behaviors. Consider having one duplicated region, i.e. two regions with the same content. These regions should contain similar translation, rotation and scaling [18]. Additionally, we should expect their matches to be spatially close to each other. In this case, any outliers which can be detected using clustering algorithms [1] or other methods, can be omitted.

2.8 Evaluation

Copy-move forgery detection algorithms are usually evaluated twofold; on image and pixel level. In practice, it is more important that an algorithm under question correctly annotates the picture as a whole, and then a human expert takes over the thorough inspection. This is important when

we are dealing with a large number of pictures where we can not afford to analyze each one individually. On the other hand, one could argue, that an algorithm cannot be evaluated on a macro level alone, and must therefore be evaluated on its region labeling capabilities as well.
When dealing with such forgeries, it is also possible that a copied region has been somehow transformed. This could be a rotation, scaling, noise introduction, blurring, or JPEG compression. Algorithms are, therefore, evaluated on such examples as well in order to evaluate their robustness to such malformations ([7],[19]).
Typical evaluation metrics are precision p and recall r

$$p = \frac{T_p}{T_p + F_p} \quad \text{and} \quad r = \frac{T_p}{T_p + F_n} \tag{1}$$

Some authors [7] suggest the use of F1 score which is the harmonic mean of aforementioned precision and recall metrics.

$$F_1 score = 2 \times \frac{p \times r}{p + r} \tag{2}$$

3. DETECTION OF COPY-MOVE FORGERY BASED ON DCT

N. N. D. Wandji et al.[19] describe an image block-matching approach based on a fuzzy matching method, using the discrete cosine transform. During the pre-processing step of the algorithm the image is converted from the RGB color space to the YUV color space, the algorithm is then performed over the R,G,B and Y components. During the block subdivision step, the R,G,B and Y channels of the input image are each split into equally-sized overlapping square blocks by sliding a fixed-size square over the input image one pixel at the time, starting from the upper left corner and finishing in the lower right corner. Upon extraction of the overlapping sub-blocks from each channel, DCT is computed over each extracted block. Per-block DTC computation is performed over the extracted blocks as per equation 3.

$$\begin{aligned} CR_{ij}(x,y) &= DCT(f_R(x+i, y+j)), \\ CG_{ij}(x,y) &= DCT(f_G(x+i, y+j)), \\ CB_{ij}(x,y) &= DCT(f_B(x+i, y+j)), \\ CY_{ij}(x,y) &= DCT(f_Y(x+i, y+j)), \end{aligned} \tag{3}$$

After the computation of DCT coefficients for each sub-block of each color channel, feature vectors representing each block of the input are constructed as per equation 4.

$$V_{ij} = [V_{ij}^1, V_{ij}^2, V_{ij}^3, V_{ij}^4, V_{ij}^5, V_{ij}^6, V_{ij}^7, V_{ij}^8, V_{ij}^9] \tag{4}$$

Where the individual elements from equation 4 are described in equation 5.

$$\begin{aligned} V_{ij}^1 &= CY_{ij}(0,0) \\ V_{ij}^2 &= CY_{ij}(0,1) \\ V_{ij}^3 &= CY_{ij}(1,0) \\ V_{ij}^4 &= CR_{ij}(0,0) \\ V_{ij}^5 &= CG_{ij}(0,0) \\ V_{ij}^6 &= CB_{ij}(0,0) \\ V_{ij}^7 &= Average_{0 \le x \le M-b+1, 0 \le y \le N-b+1}(R_{ij}(x,y)) \\ V_{ij}^7 &= Average_{0 \le x \le M-b+1, 0 \le y \le N-b+1}(B_{ij}(x,y)) \\ V_{ij}^7 &= Average_{0 \le x \le M-b+1, 0 \le y \le N-b+1}(G_{ij}(x,y)) \end{aligned} \tag{5}$$

Upon successfully generating feature vectors representing each extracted block from the input image, the vectors are arranged into an array and lexicographically sorted. After sorting each neighboring pair of feature vectors is compared to determine their similarity. As a measure of similarity, the authors use Euclidean distance. If the calculated similarity is greater than some threshold the analyzed blocks are considered suspicious. Before making a final decision it is important to check whether the pair of blocks are neighbours by checking their spatial proximity if they are spatially too close, the match is considered a false positive since neighbouring blocks are expected to have very high similarity. If two blocks are far enough apart to not be considered neighbours and if the Euclidean distance between them is smaller than some threshold, then a copy-move forgery has been detected.

4. ALTERNATIVE APPROACHES TO COPY-MOVE FORGERY DETECTION

4.1 Improved DCT-based detection method

Y. Huang et al. [9] propose a copy-move forgery detection algorithm, based on the improved DCT coefficients. The extraction of sub-blocks is performed by sliding the fixed size extraction block across a grayscale image. If the input image is in the RGB format then it is first transformed into grayscale using the following formula:

$$I = 0.299R + 0.587G + 0.114B \tag{6}$$

From the extracted blocks DCT coefficients are calculated, they are then formed into a row vector in zig-zag order. The main specialty of this algorithm is the process of truncating the DCT feature vectors. By truncating the vector we are left with a reduced dimension representation of the original feature vector. The reduction process reduces the complexity of sorting and matching procedures. The feature vectors are sorted lexicographically and the similarity between consecutive vectors is calculated. To calculate the similarity between two feature vectors the ratio of each corresponding component of the two vectors is calculated and tested to see whether the ratios are close enough. If two similar vectors are detected then the shift vector between them is calculated, if the shift vector is too small the match is discarded since it represents two areas that are close together in the picture. After a match is calculated it is marked on a mask, masks are color-coded to represent the start and end of the shift vectors with different colors. After the initial

mask is calculated, the operations of erosion and dilation are performed in order to remove noise from the image. The described algorithm has proved to be robust to JPEG compression, blurring, and additive white Gaussian noise distortion. The weakness of this approach is that it does not detect multiple copy-move forgeries well.

4.2 Improved SVD detection method

L. Kang et al. [10] describe an image block-matching approach, using the improved singular value decomposition instead of quantization DCT coefficients. The authors point out that improved singular value decomposition is superior to many other methods, mainly in regard to computational complexity. The sub-blocks are extracted from the input image in a similar method as in DCT based copy-move forgery however the user must manually specify the size of the sliding square. The algorithm the authors describe is intended for grayscale images, however an approach to apply the algorithm for color images is also described. For each block, the pixel values are extracted by columns into an array. From the extracted pixel values the singular value decomposition is performed in order to obtain singular value feature vectors. In order to reduce the computational complexity, only the k largest singular values are selected, where the principle of cumulative contribution is used to select k. From this new array of improved singular value feature vectors, the similarity is computed by first lexicographically sorting the array and then calculating the correlation coefficient for every pair of consecutive rows in the matrix. Comparing the calculated correlation coefficients with a predetermined threshold provides the answer whether copy-paste forgery was detected between the blocks, represented by the neighbouring rows in the singular value feature vector array. The raw results of the algorithm are then treated with erosion and dilation in order to remove noise and a final mask identifying copy-paste forgery is obtained. The main disadvantage of the described method is that the user is forced to input the size of the sliding block and the value of the threshold and thus greatly influences the result of the algorithm. The main advantages of this method are its robustness against noise distortion and it's low computational complexity when compared to other similar approaches that do not use the improved singular value decomposition as the basis for their fuzzy block-matching approach.

4.3 Rotation robust detection

W. Li et al. [11] present a rotation robust copy-move forgery detection algorithm. The described algorithm is an improvement of the copy-move detection algorithm, proposed Bayram et al [5]. In the first step of the algorithm sub-blocks are extracted from the original image by sliding a rectangular extraction window across the input image. Upon extraction, the Fourier transform of each block is calculated. The Fourier transform magnitude values are then re-sampled into 1-D $g(\rho)$ along the angular direction, where:

$$g(\rho) = \sum_j log \left|\rho^{0.5} I(\rho, \theta_j))\right|. \tag{7}$$

After projecting, the values of $g(\rho)$ are quantified to construct feature vectors, thus a feature vector representation of each extracted block is obtained. It is important to note that the quantization step is performed in order to get a reduced representation of feature vectors, if this step was not performed, hash comparison, performed in the following step of the algorithm would only detect exact matches. As in [5], to reduce the computational complexity of the block comparison operation, counting bloom filters are used. Using counting bloom filters comparison is performed by hashing each feature vector and saving its hash, quantization of the feature vectors performed in the previous step ensures that similar feature vectors will have identical hashes. By only comparing hashes of feature vectors and not the feature vectors themselves, the computational complexity of the algorithm is greatly reduced. After identifying similar feature vectors the distance vector between them is calculated and compared with a threshold to determine the validity of the detected matching block.

4.4 Shift-vector based detection

Weiqi Luo and Jiwu Huang [14] take the results from [17] and add several notable improvements to the process.

Their analysis of 100 natural images [14] found that it is unlikely that an image contains two very similar regions that are larger than 0.85% of the image. Same as most other algorithms, this one also uses only one color channel to detect the forgery. But, in this case, they use a more human-like transformation since they give more emphasis to the lightness of colors we see better. Their formula is: Y=0.299R+0.587G+0.114B, where Y = grayscale, R = red, G = green, B = blue. The authors assume that duplicated blocks from the same region will have similar "shift vectors" (meaning they will have to move in a similar direction and distance). This is used to further eliminate false positives, determine where the real source of a block is and position found blocks in the new region correctly. After the algorithm finds two large connected regions it determines whether a forgery has occurred based on the size of the duplicated regions. Since the duplicated regions might not be of the same size, the determining factor is the smaller region of the pair. The formula also takes into account the 0.85% threshold mentioned before. Added improvements have improved the algorithm considerably compared to the results of their baseline. For their examples, the new algorithm still produced respectable results with SNR=16dB, whereas the baseline failed at SNR $<$ 24dB.

4.5 Detection using SURF

This paper proposes a new method using Haar wavelet and ring descriptor. This makes the copied object more distinctive and reduces detection time drastically. This method is resistant to harder to detect image transformations as well, such as scaling, rotation, or even a compound of different image processing tools as seen in figure 2.

The paper builds on SIFT method developed by H.Huang, by making the computation much faster, by abandoning 128-dimensional descriptor of SIFT, developed by H.Huang [8] and proposed by Lowe. [13] Instead B.Herbert [4] proposes a new scale and rotation resistant detection SURF, which reduces the detection time compared to SIFT. While this algorithm is faster than SIFT, it also loses some accuracy. The new orientation assignment and descriptor is based on the distribution of first-order Haar wavelet response in the horizontal and vertical direction. The orientation assignment is done using a circular neighborhood around the point

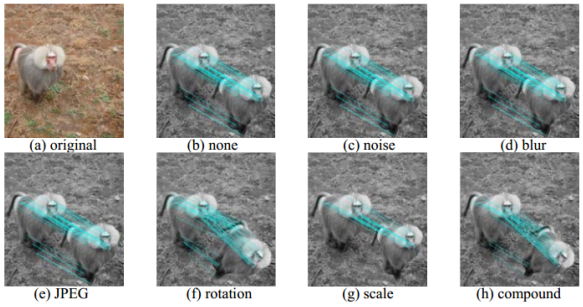


Figure 2: Different types of image forgery. Compound is a combination of: noise, rotation and JPEG compression.

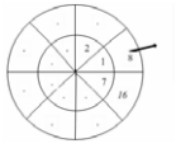


Figure 3: For each sub-region, the Haar wavelet response is computed to build the descriptor

of interest. The Haar wavelet responses are weighted with a Gaussian with *delta = 2.5s* around the center point. The dominant orientation is then estimated by summing dx and dy points, within

$$\delta = \pm\pi/3 \tag{8}$$

angle in the wavelet response space. Then the use Gaussian and Haar wavelet response to extract descriptors as well. They do that by dividing the circular region into 8 sub-regions like this. For each region, a 4-D descriptor vector is computed. After they are concatenated and form a 64-D vector and form a keypoint.

After SIFT keypoints are detected, the algorithm uses Best-Bin-First algorithm to find the best feature match.

5. RESULTS

We conclude by comparing the most common and best-performing algorithms. [7] created a challenging suite of real world tests and a framework for systematic testing. The metrics used are precision, recall, a derivative of both, and F1. In short, their experiments show that the keypoint based algorithms SIFT and SURF, along with block-based features: DCT, DWT, KPCA, PCA, and Zernike perform very well. They exhibit the best robustness against common challenges, such as noise and down-sampling, while reliably identifying the copied regions. While the implementations might not be exactly the same as described in this article (partially due to their vagueness in certain parts), they follow the same basic principles.

Tested images are from a database the authors created themselves. The dataset consists of high resolution (3000*2300px) images, with forgeries consisting of 10% of the image.

Method	Precision	Recall	F_1	τ_2
Blur	88.89	100.00	94.12	100
Bravo	87.27	100.00	93.20	200
Circle	92.31	100.00	96.00	200
DCT	78.69	100.00	88.07	1000
DWT	84.21	100.00	91.43	1000
FMT	90.57	100.00	95.05	200
Hu	67.61	100.00	80.67	50
KPCA	87.27	100.00	93.20	1000
Lin	94.12	100.00	96.97	400
Luo	87.27	100.00	93.20	300
PCA	84.21	100.00	91.43	1000
Sift	88.37	79.17	83.52	4
Surf	91.49	89.58	90.53	4
SVD	68.57	100.00	81.36	50
Zernike	92.31	100.00	96.00	800
Average	85.54	97.92	90.98	–

Figure 4: Plain copy-move forgery [7]. Measuring how many copied regions are found.

Method	Precision	Recall	F_1
Blur	98.07	78.81	86.19
Bravo	98.81	82.98	89.34
Circle	98.69	85.44	90.92
DCT	92.90	82.85	84.95
DWT	90.55	88.78	88.86
FMT	98.29	82.33	88.79
Hu	97.08	74.89	82.92
KPCA	94.38	88.36	90.24
Lin	99.21	78.87	86.69
Luo	97.75	82.31	88.41
PCA	95.88	86.51	89.82
Sift	60.80	71.48	63.10
Surf	68.13	76.43	69.54
SVD	97.53	76.53	83.71
Zernike	95.07	87.72	90.29
Average	92.21	81.62	84.92

Figure 5: Plain copy-move forgery at pixel level [7]. Measuring what percentage of the region is marked.

Method	Precision	Recall	F_1	Precision	Recall	F_1
Blur	95.24	52.50	67.31	89.91	54.11	65.20
Bravo	97.54	52.58	68.16	88.75	58.27	67.58
Circle	95.12	60.90	73.75	89.60	62.48	71.43
DCT	19.15	5.37	8.02	66.11	55.76	55.06
DWT	52.15	14.55	21.21	81.88	69.15	71.84
FMT	94.42	54.07	68.14	88.85	60.50	69.91
Hu	94.98	54.08	68.64	89.98	54.61	65.99
KPCA	37.01	7.50	12.05	87.79	62.27	70.06
Lin	96.84	51.04	66.61	90.86	59.96	70.63
Luo	95.53	51.70	66.72	89.32	58.95	68.47
PCA	37.79	9.05	13.95	88.20	61.95	71.77
Sift	11.37	4.95	6.74	17.00	7.34	10.07
Surf	37.49	21.86	26.15	38.31	22.93	26.79
SVD	91.91	59.06	71.51	71.98	58.91	59.33
Zernike	83.15	22.00	33.52	87.55	61.87	69.64
Average	69.31	34.75	44.83	77.31	53.71	60.65

Figure 6: Multiple copy-move forgeries at pixel level [7]. Measuring what percentage of the region is marked across all the copies of the same region.

Method	Feature	Matching	Postpr.	Total	Mem.
Blur	12059.66	4635.98	12.81	16712.19	924.06
Bravo	488.23	5531.52	156.27	6180.42	154.01
Circle	92.29	4987.96	19.45	5103.43	308.02
DCT	28007.86	7365.53	213.06	35590.03	9856.67
DWT	764.49	7718.25	119.66	8606.50	9856.67
FMT	766.60	6168.73	8.07	6948.03	1732.62
Hu	7.04	4436.63	5.36	4452.77	192.51
KPCA	6451.34	7048.83	88.58	13592.08	7392.50
Lin	12.41	4732.88	36.73	4785.71	346.52
Luo	42.90	4772.67	119.04	4937.81	269.52
PCA	1526.92	4322.84	7.42	5861.01	1232.08
Sift	15.61	126.15	469.14	610.96	17.18
Surf	31.07	725.68	295.34	1052.12	19.92
SVD	843.52	4961.11	7.65	5816.15	1232.08
Zernike	2131.27	4903.59	27.23	7065.18	462.03
Average	3549.41	4829.22	105.72	8487.63	2266.43

Figure 7: Averaged processing time per image in seconds and minimum peak memory requirements in MB [7]. Authors recommend double the minimum peak memory for smooth performance.

Looking at the results we can see, that a keypoint based method can be very efficient. SIFT and SURF both have excellent performance characteristics. Looking at the results it seems like SURF is a clear choice, having a better F1 score across the tests while having very similar performance. Keypoint based methods, however, are inferior to block-based methods in low-contrast regions and repetitive image content. If the image was resized before forgery detection there was an accuracy loss primarily for block-based algorithms. This becomes a significant problem with large images of 3000x2000 pixels or more.

6. CONCLUSIONS

We can conclude by separating the methods into two categories once more, evaluating them separately, and then determining the best usages for each best-in-class algorithm. Keypoint based methods have much better performance characteristics, which matters a lot considering that all algorithms take a lot of time for each image - ranging from 600 seconds to roughly 35'000. They also consume significantly less memory, which makes parallelization of the search possible. Taking this into consideration we believe, that keypoint based methods are best suited for searching a larger number of images for automated selection. The algorithm we believe is best in this class is SURF. Block-based methods are much slower, but also bring improved accuracy. Out of the tested algorithms, the authors of [7] recognize DCT, DWT, KPCA, PCA, and Zernike as the best, and recommend the use of Zernike due to a much lower memory footprint. To us, this seems like a hint at the importance of parallelization. With these advantages in each category, we recommend using keypoint based methods as a means of quickly categorizing images and later analyzing only the ones that need a closer look with block-based methods.

7. REFERENCES

[1] I. Amerini, L. Ballan, R. Caldelli, A. Del Bimbo, and G. Serra. A sift-based forensic method for copy–move attack detection and transformation recovery. *IEEE transactions on information forensics and security*, 6(3):1099–1110, 2011.

[2] M. Bashar, K. Noda, N. Ohnishi, and K. Mori. Exploring duplicated regions in natural images. *IEEE Transactions on Image Processing*, 2010.

[3] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359, 2008.

[4] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

[5] S. Bayram, H. Sencar, and N. Memon. An efficient and robust method for detecting copy-move forgery. In *2009 IEEE International Conference on Acoustics, Speech, and Signal Processing - Proceedings, ICASSP 2009*, ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings, pages 1053–1056, Sept. 2009. 2009 IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2009 ; Conference date: 19-04-2009 Through 24-04-2009.

[6] S. Bayram, H. T. Sencar, and N. Memon. An efficient and robust method for detecting copy-move forgery. In

2009 IEEE International Conference on Acoustics, Speech and Signal Processing, pages 1053–1056. IEEE, 2009.

[7] V. Christlein, C. Riess, J. Jordan, C. Riess, and E. Angelopoulou. An evaluation of popular copy-move forgery detection approaches. *IEEE Transactions on Information Forensics and Security*, 7(6):1841–1854, 2012.

[8] H. Huang, W. Guo, and Y. Zhang. Detection of copy-move forgery in digital images using sift algorithm. In *2008 IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application*, volume 2, pages 272–276. IEEE, 2008.

[9] Y. Huang, W. Lu, W. Sun, and D. Long. Improved dct-based detection of copy-move forgery in images. *Forensic science international*, 206(1-3):178–184, 2011.

[10] L. Kang and X.-p. Cheng. Copy-move forgery detection in digital image. pages 2419–2421, 10 2010.

[11] W. Li and N. Yu. Rotation robust detection of copy-move forgery. In *2010 IEEE International Conference on Image Processing*, pages 2113–2116, 2010.

[12] H.-J. Lin, C.-W. Wang, Y.-T. Kao, et al. Fast copy-move forgery detection. *WSEAS Transactions on Signal Processing*, 5(5):188–197, 2009.

[13] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[14] W. Luo, J. Huang, and G. Qiu. Robust detection of region-duplication forgery in digital image. In *18th International Conference on Pattern Recognition (ICPR'06)*, volume 4, pages 746–749. IEEE, 2006.

[15] B. Mahdian and S. Saic. Detection of copy–move forgery using a method based on blur moment invariants. *Forensic science international*, 171(2-3):180–189, 2007.

[16] P. Mishra, N. Mishra, S. Sharma, and R. Patel. Region duplication forgery detection technique based on surf and hac. *The Scientific World Journal*, 2013, 2013.

[17] A. C. Popescu and H. Farid. Exposing digital forgeries by detecting duplicated image regions. *Dept. Comput. Sci., Dartmouth College, Tech. Rep. TR2004-515*, pages 1–11, 2004.

[18] S.-J. Ryu, M.-J. Lee, and H.-K. Lee. Detection of copy-rotate-move forgery using zernike moments. In *International workshop on information hiding*, pages 51–65. Springer, 2010.

[19] N. D. Wandji, S. Xingming, and M. F. Kue. Detection of copy-move forgery in digital images based on dct. *arXiv preprint arXiv:1308.5661*, 2013.

[20] N. B. A. Warif, A. W. A. Wahab, M. Y. I. Idris, R. Ramli, R. Salleh, S. Shamshirband, and K.-K. R. Choo. Copy-move forgery detection: survey, challenges and future directions. *Journal of Network and Computer Applications*, 75:259–278, 2016.

[21] J. Zhao and J. Guo. Passive forensics for copy-move image forgery using a method based on dct and svd. *Forensic science international*, 233(1-3):158–166, 2013.

[22] Y. Zhou and Y. Xie. Copy-move forgery detection using improved sift. In *Eighth International Conference on Digital Image Processing (ICDIP 2016)*, volume 10033, page 100334S. International Society for Optics and Photonics, 2016.

Detecting Copy-Move Image Forgery

Digital Forensics, University of Ljubljana, Faculty of Computer and Information Science 2019/2020

Bernardo Manuel Costa Barbosa
University of Ljubljana
Porto, Portugal
bc4968@student.uni-lj.si

Nikolina Grabovica
University of Ljubljana
Belgrade, Serbia
ng8969@student.uni-lj.si

ABSTRACT

This paper approaches a deep neural architecture for image copy-move forgery detection (CMFD), a deep neural network solution to localize potential image manipulation via visual artifacts, copy-move regions and visual similarities.

Outperforming the state-of-the-art copy-move detection algorithms on several data sets, BusterNet [3] can be used against some of the most common crimes today, such as fake news, manipulation of surveillance images, among many others.

There are different 'weapons' that are used to discover a forged image and deep convolution learning algorithms are one of them.

Keywords

Copy-Move, Image Forgery Detection, Deep Learning

1. INTRODUCTION

Nowadays, with globalization and evolution of social networks, we are in an era where fake news and images are modified in order to make us believe that a story is true, even when it is not. Using tools like Adobe Photoshop or GIMP, it is possible to deceive the human eye very easily, being very hard to distinguish original image from a manipulated one. In addition, the cost of digital equipment such as cameras, smartphones and powerful computers has decreased, allowing the use and possession of these instruments possible for everyone.

As previously mentioned, deep convolution learning algorithms are powerful solutions. This type of algorithm can find if the image is altered such that it appears identical to the original image and is nearly undetectable to the human eye as a forgery. [1].

2. COPY-MOVE FORGERY

Image Forgery detection is currently one of the popular research areas and Copy-Move (CM) Forgery is one of the repeatedly used techniques that makes image manipulation extremely easy, whether or not with malicious intention.

The most performed operations in image forging are deleting or hiding parts in the image, adding a new object into the image and falsifying the image information.

2.1 Forgery Detection Algorithms

Copy-Move in digital image could be performed on or at least one area in the forged image. The job of the detection algorithm is to determine whether an image contains duplicated areas or not.

Since we don't know the shape and the size of the areas that were copied, the best solution is to split an image into fixed-sized overlapping blocks. This detection consists of feature extraction, matching and copy decision ground on the resemblance information between blocks. In Figure 1 we can see how the algorithm for detection copy- move forgery, based on DCT (discrete Cosine transform) as the discriminating feature, works.

2.2 Merged Deep Neural Networks as CMFD

In one study it was presented how copy-move forgery detection algorithm that distinguishes two deep neural networks — a GAN and a custom CNN-based - works.

GAN (Generative Adversarial Network) will be built first since it consists on the generator and the discriminator. The final step in the building involves merging the two output networks to create a merging network.

The outputs of these two networks serve as inputs for the merging network unit. The three bases intent for building this new, merged network are:

- Make a final resolution on the copy-move forgery image;
- Make the copy-move transaction localized;
- Make a difference between the original and possible forged area of the image.

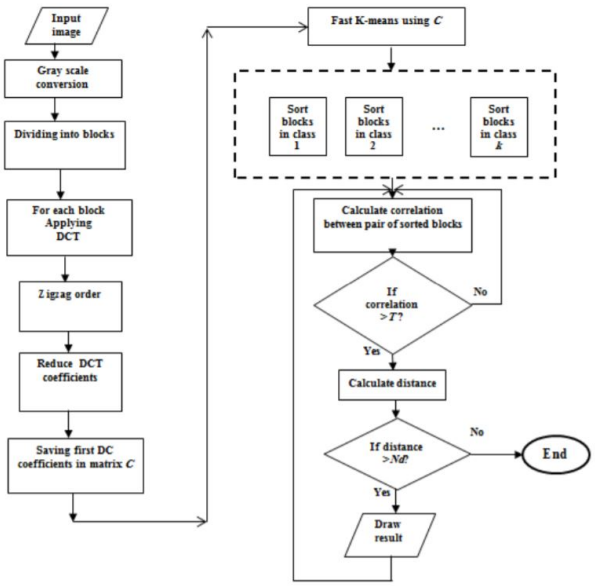


Figure 1: Rapid Copy-Move Forgery Detection method.

Basic neural network classifier features n input nodes, n possible values that can be supposed by the dependent variable. Here, we use as input values the two vectors from the other one network output.

From what follows that CMFD classifier output, in this merged network, is a node that stands for the concatenated sum of all momentous inputs.

GAN is used for detecting the forged area in the image and the similarity CNN to detect any similar area on the image.

Outputs are combining into a layer to represent new vector inputs, where the first layer represents an SVM classifier [2].

This CMFD method is shown on the Figure 2.

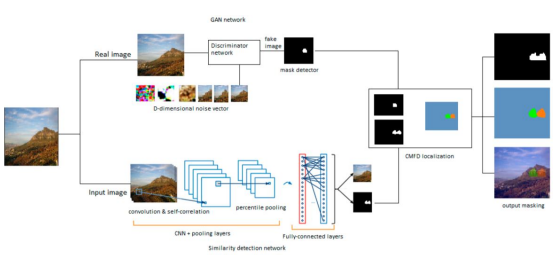


Figure 2: Merged deep neural networks copy-move forgery detection method.

2.3 Experiment in MATLAB

The most common practice of forging images is moving parts of the same image across the picture.

For example, an editor can camouflage the original object by "patching" it with a piece of background cloned from that same image, or just copy or move original objects around the picture.

We tested one of CMFD algorithms in MATLAB, and our forged image was discovered.

In this experiment, the input image was presenting two umbrellas on the beach, but after the algorithm was launched on the specific forged image, we got the results shown on the Figure 3.

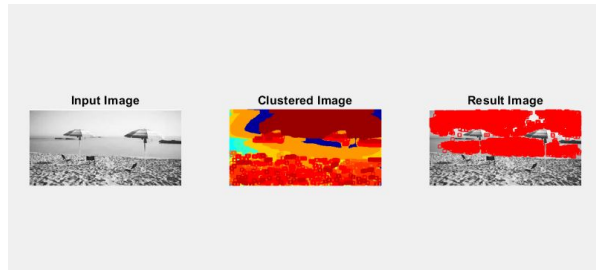


Figure 3: Result after running the rapid CMFD algorithm.

The second image stands for result after applying clustering algorithm (K-clustering algorithm), which part in CMFD is shown on the Figure 1, groups similar objects based on features into a number K of groups (K>0).

Grouping is finished by reducing the sum of squares of distances between data and the corresponding cluster center.

The third image fundamentals matching points that allow detecting the forged image and copied area.

In Figure 4 we can see how the original image was looking, and what specific area of the image was cloned.

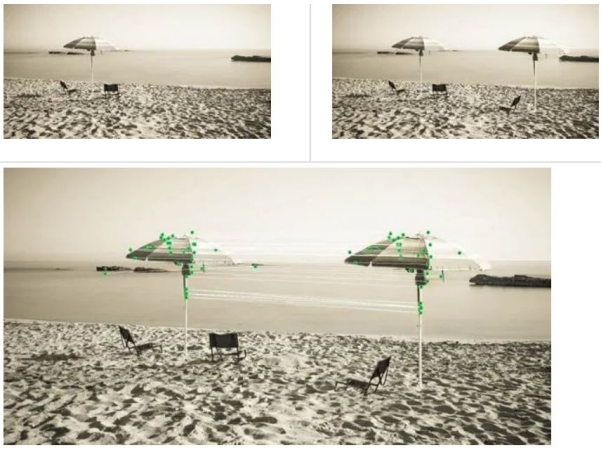


Figure 4: Forged Image.

3. BUSTERNET

3.1 Overview

BusterNet is an attempt to fix the drawbacks of classic Copy-Move Forgery Detection pipelines, based on an end-to-end trainable deep neural network that doesn't include decision rules and capable of producing distinct source/target manipulation masks that can be later used for forensic analysis, for example.[3]

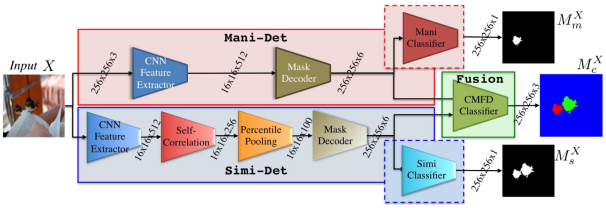


Figure 5: BusterNet Architecture Overview.

In Figure 5, we can observe the 2-branch Deep Neural Network of BusterNet, Manipulation and Similarity, as well as the Fusion module that is responsible for outputting the final mask, which is color coded to represent pixel classes, pristine (blue), source-copy (green) and target-copy (red).

3.2 Manipulation Branch

This branch seeks to detect if there was any manipulation in the image by extracting features using a CNN feature extractor and then up sampling them to the original image size using a Mask Decoder.

Lastly, it produces a pixel-level X manipulation mask via Binary Classifier, i.e. M_m^X.

3.3 Similarity Branch

The Similarity Branch is a little bit more complex than the Manipulation Branch, since it targets the detection of cloned regions.

Sharing the same CNN feature extractor, with different weights, it computes similarities via Self-Correlation module, followed by collecting useful statistics via Percentile Pooling and up sampling feature maps to the original image size using a Mask Decoder.

At last, it also produces a pixel-level mask X using Binary Classifier, i.e. M_s^X .

3.4 Fusion Module

Finally, Fusion module is responsible by predicting pixel-level copy-move masks differentiating pristine, source-copy, and target-copy classes.

This process is based on taking Mask Decoder's features from both branches, concatenate and fuse them using BN-Inception and predicting the three-class CMFD mask using a Conv2D layer, i.e M_c^X.

3.5 Results

Overall, results are impressive, even on objects that are not included in the datasets used for training, such as flowers and sand, per example.

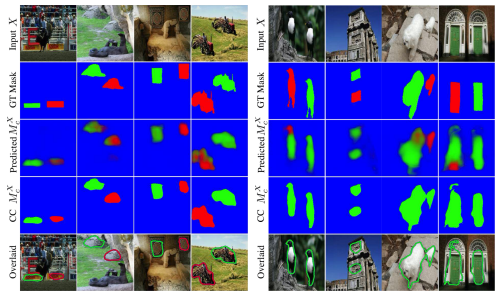


Figure 6: BusterNet - Positive and Negative Results.

In Figure 6, it is possible to see the input images, the masks obtained and the ground truth positions.

On the left, BusterNet was able to properly detect where the image was copy-move forgery, and according to the data provided on the original paper, outperforms state-of-the arts methods by a large margin and is also robust against various known CMFD attacks.

4. CONCLUSION

In this paper we presented some problem of image forging, one of the global phenomenons that is spreading mostly through social media.

This global problem requires special tools and algorithms capable of finding forged parts of an image, even when it is impossible for the human eye to recognized it.

In the first part, we discussed accelerated CMFD algorithm and two-merged deep neural networks CMFD algorithms.

We tested an implementation in *MATLAB* to check if the forged area was recognized, which turned out successfully.

In the second part, we focused on BusterNet, an advanced tool for image copy-move forgery detection, mainly on the it's architecture and what we can achieve with it.

We were surprised by the results, since some of the positive results are very hard to detect for human eyes, but like everything, it's not perfect and perhaps it can get even better if trained with a more sophisticated dataset.

Last but not least, tools like BusterNet are definitely a no-brainer for Digital Forensics, being a powerful addition to the already existing methods.

5. REFERENCES

[1] Y. Abdalla, M. T. Iqbal, and M. Shehata. Copy-move forgery detection and localization using a generative adversarial network and convolutional neural-network. *Information*, 10(9):286, 2019.

[2] S. M. Fadl and N. A. Semary. A proposed accelerated image copy-move forgery detection. *2014 IEEE Visual Communications and Image Processing Conference*, 2014.

[3] Y. Wu, W. Abd-Almageed, and P. Natarajan. Busternet: Detecting copy-move image forgery with source/target localization. *Computer Vision – ECCV 2018 Lecture Notes in Computer Science*, page 170–186, 2018.

Stvaritev in zaznavanje Deepfake videoposnetkov z globokim učenjem

Digitalna forenzika 2019/2020

Jaka Koren
Fakulteta za Računalništvo in Informatiko
Večna pot 113
1000, Ljubljana
jk6977@student.uni-lj.si

Vid Trtnik
Fakulteta za Računalništvo in Informatiko
Večna pot 113
1000, Ljubljana
vt4438@student.uni-lj.si

Enej Guček Puhar
Fakulteta za Računalništvo in Informatiko
Večna pot 113
1000, Ljubljana
eg1304@student.uni-lj.si

POVZETEK

Razvoj metod globokega učenja nam je omogočil reševanje kompleksnih, večdimenzionalnih problemov na področju digitalnih medijev in računalniškega vida. Hkrati pa je privedel do nastanka tehnologij, ki predstavljajo grožnjo za zasebnost, varnost in zaupnost. Deepfake tehnologija je ena od bolj izpostavljenih; lahko ustvari ponarejene slike in posnetke ljudi, ki so zelo težko ločljivi od pristnih. Potreba po zanesljivih metodah preverjanja pristnosti takih medijev narašča, zato bomo v tem prispevku predstavili izzive, s katerimi se digitalni forenziki soočajo pri tem, ter naredili pregled predlaganih modernih metod za samodejno prepoznavanje ponaredkov.

Ključne besede

deep learning, deepfakes, computer vision, forensics, digital forensics, audiovideo manipulation

1. UVOD

Beseda "Deepfake" je tvorjenka sestavljena iz "deep learning" in "fake". Gre za obliko sintetičnega medija v katerem se osebi iz nekega obstoječega videa ali slike priredi podoba neke druge osebe (Slika 1). Rezultat pretvorbe je navidezno pristna podoba osebe, ki izvaja dejanja ali govori nekaj, kar v resnici nikoli ni. Takšni ponaredki lahko prelisičijo nepozorne opazovalce. Ustvarjanje Deepfake ponaredkov temelji na metodah globokega učenja, kot so samopretvorniki in generativne adversarialne mreže (GAN - generative adversarial networks).

Deepfake tehnologije so se v akademskih krogih pojavile že v zgodnjih 1990-ih letih predvsem na področju računalniškega vida. Eden izmed prelomnih projektov je bil tako imenovan "Program za prepis videov" (Video Rewrite program), ki je izšel leta 1997 [2]. Ta program spremeni mimiko osebe na posnetku tako, da se usta osebe v videu premikajo v skladu z zvokom z drugega zvočnega posnetka. Šlo je za prvi primer sistema, ki je avtomatiziral takšno vrsto obrazne animacije in se je predvsem osredotočal na uporabo strojnega učenja. Kasnejši akademski projekti so se osredotočili predvsem na izboljševanje hitrosti metod ter izdelavo bolj realističnih posnetkov.

Slika 1: Primer zamenjave obraza v video posnetku z uporabo Deepfake tehnologij. Ponarejen posnetek je na levi, izvirni pa na desni. Drugače lahko namesto druge slike na tarčni obraz prenesemo samo drugačno obrazno mimiko.

Sam izraz "Deepfake" izvira iz leta 2017, iz uporabniškega imena "deepfakes" uporabnika strani Reddit. Uporabnik je bil eden od večih, ki so delili s to metodo prirejene posnetke po omrežju z drugimi. Te posnetki so bili tipično prizori iz raznih filmov, kjer se je obraz enega izmed igralcev zamenjal z obrazon kakšnega drugega zvezdnika (predvsem popularno je bilo spreminjanje obraze igralcev z obrazom Nicholas Cage-a). Posnetki, kjer so bili obrazi slavnih oseb postavljeni na obraze igralk v pornografskih vsebinah, so bili tudi priljubljeni. Gosteitelji Reddit-a so večino teh uporabnikov in njihovih objav odstranili, a na straneh, kjer takšni posnetki še niso prepovedani, se deepfake ponaredki še vedno delijo in širijo.

S širjenjem priljubljenosti so se tudi medijska podjetja začela

spuščati v to področja. V začetku leta 2018 je na splet prispelo več aplikacij, ki laikom omogočajo podobne predelave slik in posnetkov, kot s Faceapp, Fakeapp in DeepFaceLab. Spletne strani, kot npr. thispersondoesnotexist, z GAN mrežami ustvarjajo fotorealistične podobe umetnih ljudi. Podjetje DataGrid iz Japonske je izdelalo orodje, ki na podoben način ustvari sliko celega telesa za potrebe modnih hiš [1]. Drugi primeri deepfake tehnologij vključujejo tudi aplikacije za ponarejanje zvokov oz. oponašanje glasu določene osebe, ali pa generiranje slik povsem novih ljudi.

Deepfake tehnologija je pridobila široko pozornost zaradi svoje vpletenosti v lažne novice ("fake news"), prevare, finančne goljufije, zvezdniško pornografijo itd. To je posledično povzročilo da tako v industriji kot v vladi sedaj ugotavljajo načine kako učinkovito in zanesljivo zaznati Deepfake-e in kako omejiti njihovo uporabo.

2. LEGALNOST IN ZAKONITOST

Spreminjanje slik, avdio ali video zapisov samo po sebi ni nič novega in nič škodljivega. Ponuja uporabniku mnogo zanimivih, koristnih in tudi zabavnih možnosti pri urejanju in obdelavi slikovnih, avdio in video vsebin. Vendar pa se ob uporabi Deepfake aplikacij soočamo z primerom zlonamernega in žaljivega ponarejanja, prirejanja in razširjanja neresničnih in lažnih vsebin preko spleta ali drugih medijev. Že omenjeni pornografski videi[1] so imeli izjemen odmev ter vplivali na ugled in dobro ime vrste javno izpostavljenih oseb, predvsem oseb iz filma, estrade in politike. Takšne vsebine, ustvarjene z Deepfake tehnologijo, žalijo in razširjajo neresnične informacije o javno izpostavljenih osebnostih z namenom vplivanja na javno mnenje ali z namenom javne, osebne, moralne, strokovne ali politične diskvalifikacije posamezniq-ka ali inštitucije. To pa so naklepna dejanja, ki so lahko tudi kazensko sankcionirana.

S tehnologijo Deepfake se lahko ponaredi posnete izjave javnih oseb ali politikov, in se jih potem uporabi tudi za izsiljevanje, spodbujanje politične napetosti in nemira ali za javno širjenje laži in neresnic. V kriminalističnih preiskavah se lahko takšne vsebine pojavijo kot dokazni obremenilni material. Velika težave pri preprečevanju in omejevanju tovrstnih tehnologij je, da so danes Deepfake tehnologije zelo razširjene. Ni problem v tehnologiji. Problem je v človekovi zlorabi tehnologije v inkriminirane namene. Na spletu je moč dnevno najti ogromno prosto dostopnih slikovnih, video in avdio vsebin javnih oseb, ki so iz dneva v dan objekt napada in poskusa javne diskreditacije.

V legalnih postopkih je treba zagotoviti, da so pridobljeni materialni dokazi o uporabi Deepfake tehnologij in vsebin verodostojni, zakoniti in ustrezno dokumentirani. V nasprotnem primeru je lahko obsojena nedolžna oseba ali pa opuščen sodni postopek zoper obdolženca zaradi pomanjkanja oz. nezadostnih materialnih dokazov. To velja tudi za digitalne dokaze, kot so npr. slike, avdio in video posnetki, računalniške datoteke in programska orodja, ki so bili shranjeni, obdelani ali uporabljeni na digitalnih napravah. Deepfake tehnologija predstavlja danes nov izziv za digitalne forenzike, kriminaliste, tožilce, sodne izvedence in sodnike.

[1]Nekonsenzualna pornografija (v več primerih tudi t.i. maščevalni porniči) predstavlja danes 96-odstotkov deepfake izdelkov, ki so nameščeni na svetovnem spletu.

Zato ni slučajno, da se je s popularizacijo in širjenjem Deepfake tehnologije pojavila tudi potreba po izvedeniško zanesljivih preiskovalnih metodah in tehnikah odkrivanja in prepoznavanja Deepfake vsebin.

3. TEHNOLOGIJA DEEPFAKE

Tehnologija Deepfake je dandanes zelo razširjena preko mnogih aplikacij, ki uporabnikom olajšajo oblikovanje takšnih vsebin. Praviloma so te aplikacije osnovane na metodah globokega učenja, ki se pogosto uporabljajo za reševanje kompleksnih, večdimenzionalnih problemov računalniškega vida in umetne inteligence. Prvi poskusi Deepfake-ov so uporabljali par samodejnih kodirnikov in dekodirnikov (Slika 1). Kodirnik iz slike obraza izloči latentne informacije o mimiki, kretnjah, dekodirnik pa iz njih rekonstruira sliko. Za zamenjavo obrazov potrebujemo dva takšna para, ki se učita na ločenih množicah slik, kodirnika pa si delita parametre. Skupni kodirnik se tako nauči najti ujemanja obraznih značilk. Postopek olajša tudi dejstvo, da imamo vsi ljudje podobno obrazno strukturo. Kvaliteto in točnost ponaredkov se da še izboljšati z dodajanjem konvolucijskih slojev v globoko omrežje.

Moderne metode vključujejo generiranje in treniranje GAN mrež. Te poleg treniranja generatorja za obdelavo slik pošiljajo učne podatke in rezultate skozi diskriminator, ki poskuša oceniti, ali je bila slika izdelana umetno z generatorjem ali ne. Na ta način spodbuja generator k boljši točnosti pri sintezi novih slik in videov. Oba algoritma se izboljšujeta v igri ničelne vsote (Zero sum game), na koncu pa mreža lahko predeluje in sintetizira skoraj fotorealistične podobe.

4. PREPOZNAVANJE DEEPFAKE VSEBIN

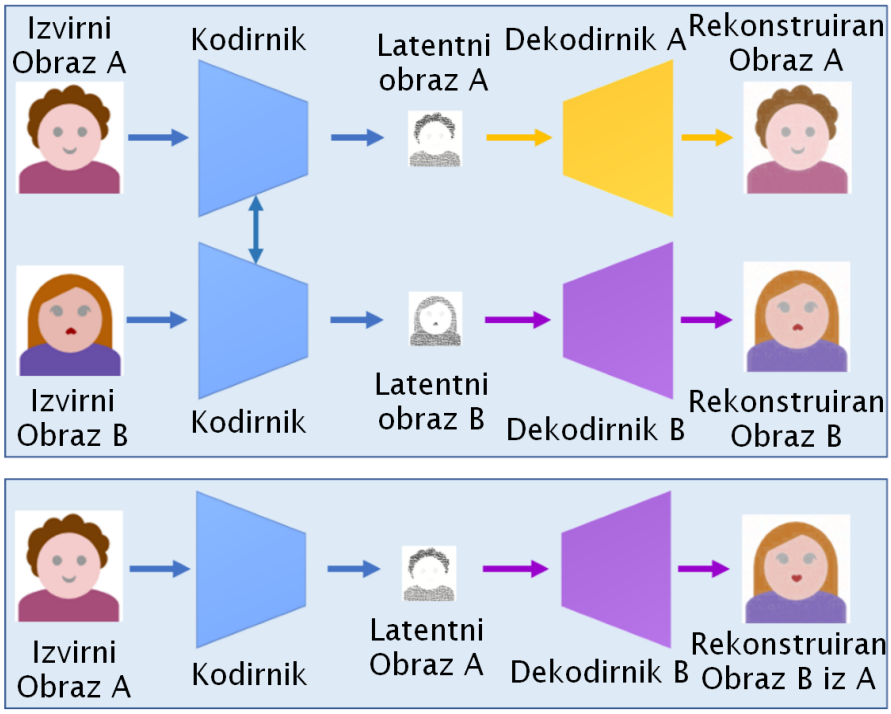
Prvi poskusi so osnovani na zaznavanju motenj in artefaktov, ki ostanejo pri sintezi videov. Dandanašnje metode pa uporabljajo globoko učenje za samodejno izločevanje in iskanje značilk, preko katerih lahko prepoznamo Deepfake ponaredke. Gre za dvorazredni klasifikacijski problem, kjer mora sistem samodejno določiti, ali je slika oziroma posnetek ponarejen ali ne.

Za preizkušanje metod sta Korushnov in Marcel sestavila podatkovno množico iz 620 videov, obdelanih z prosto dostopnim modelom Faceswap-GAN. Množica vsebuje posnetke z zamenjanimi obrazi različnih stopenj kvalitete slik in obrazno mimiko [7]. Prvi poskusi so pokazali, da metode osnovane na popularnih prepoznavalcih obrazov in metrikah kvalitete slik ne ločujejo med ponaredki in pristnimi slikami zadovoljivo. Potreba po razvoju bolj zanesljivih metod za zaznavanje ponaredkov je zato še večja.

V naslednjih dveh poglavjih bomo predstavili nekaj metod za zaznavanje ponarejenih slik in videov. Slednje so ločene v dve podkategoriji glede na način analize videov.

4.1 Prepoznava slik

Uporaba modelov globokega učenja, kot sta CNN in GAN, otežuje forenzično analizo ponarejenih slik obrazov, ker lahko ti ohranijo pozo, izraz ter osvetljenost izvirnega portreta. Z orodji za obdelavo slik se da prikriti morebitne sledove zlivanja slike, kar dodatno poveča težavnost analize.



Slika 2: Model za ustvarjanje Deepfake ponaredkov z dvema paroma kodirnik-dekodirnik. Dve nevralni mreži pri učenju uporabljata isti kodirnik in različna dekodirnika (zgoraj). Za ustvarjanje ponaredka v kodirnik spustimo obraz A, ki ga nato dekodiramo z dekodirnikom B (spodaj).

Hsu in drugi [5] so pred kratkim predstavili dvostopenjsko metodo za klasificiranje deepfake slik z uporabo globokega učenja. V prvi fazi se njihova mreža za izločanje skupnih lažnih značilk (common fake feature network - CFFN) iz večih gostih slojev uči iskati fine razlike med deepfake ponaredki in pristnimi portreti. V drugi fazi se najdene značilke na slikah pošljejo skozi kratko konvolucijsko nevronsko mrežo, ki se uči klasificirati slike. Metoda je bila preizkušena na podatkovni množici 600,000 učnih pristnih in ponarejenih slik, ter 10,000 testnih slik obeh vrst. Eksperimentalni rezultati pri tej metodi so veliko boljši kot pri konkurenčnih detektorjih lažnih slik.

4.2 Prepoznava videov

Večina metod za prepoznavanje ponarejenih slik ni uporabna pri prepoznavanju videoposnetkov zaradi izgube kvalitete slike pri kompresiji videov. Poleg tega se v posnetkih nahajajo značilne časovne spremembe med posameznimi sličicami, ki predstavljajo velik izziv za metode, osnovane za negibne slike. V naslednjem poglavju bomo predstavili metode, ki iščejo značilke v časovnih spremembah, potem pa metode, ki analizirajo samo sličice v posnetku.

4.2.1 Časovne značilke v posnetkih

Pri sintezi ponarejenih posnetkov se pogosto izgubi časovna zveznost med sličicami v posnetku, na kar se osredotoča prva podmnožica metod. Sinteza obrazov se izvaja na posameznih sličicah, kar pusti drobne motnje, ki pa postanejo bolj opazne kot vizualne neskladnosti v posnetku skozi čas. Pri zaznavanju teh uspevajo rekurenčne in konvolucijske nevronske mreže. Sabir in drugi [10] predstavljajo model, ki s temi metodami primerja izrez obraza v prvi in n-ti sličici posnetka. Variacija tega algoritma doseže 96-odstotno točnost pri prepoznavanju deepfake videov.

Guera in Delp [3] uporabljata slednje za iskanje značilk v sličicah, ki jih pošljeta v LSTM (long short-term-memory) mrežo, ki oceni verjetnost, da je posnetek ponarejen ali ne. V eksperimentu z podatkovno množico 600 video posnetkov, od katerih je bila polovica ponarejenih, ta metoda prepozna Deepfake ponaredke s 97-odstotno točnostjo.

Po drugi strani imamo predlog za metodo, ki temelji na spoznanju, da ljudje na ponarejenih posnetkih mežikajo pogosteje kot v izvirnih. Algoritmi za ustvarjanje Deepfake ponaredkov za učenje pogosto uporabljajo portrete ljudi z interneta, kjer so ljudje prikazani z odprtimi očmi, oz. kjer je zelo malo slik ljudi z zaprtimi očmi. Posledično ti algoritmi nimajo sposobnosti ustvarjati lažnih obrazov z zaprtimi očmi, in osebe v ponarejenih posnetkih veliko manj mežikajo. Li in drugi [8] predstavljajo metodo, ki najprej poišče oči oseb v posnetku in jih primerno obdela za analizo. Te regije slik nato pošlje v rekurenčno konvolucijsko nevralno mrežo, ki za vsako sličico napove verjetnost, da se bodo oči zaprle. Naravno mežikanje ima značilen časovni vzorec, na podlagi katerega lahko metoda napove, ali je posnetek ponarejen ali ne. Metoda je preizkušena na podatkovni množici iz 49 posnetkov intervjujev, in kaže obetajoče rezultate.

4.2.2 Vizualne motnje v sličicah posnetka

Drugi pristop loči cel posnetek v posamezne sličice, in se osredotoči samo na vizualne motnje v njih, da pridobi značilke za klasifikacijo. Značilke lahko pošljemo v globoke ali plitke klasifikatorje, ki določijo pristnost posnetka.

Med globoke klasifikatorje štejemo konvolucijske nevronske mreže, ki zaznavajo vizualne motnje. Deepfake ponaredki so po navadi narejeni iz posnetkov z omejeno ali neskladno resolucijo, zaradi česar obdelava za seboj pusti značilne artefakte v sličicah. Metoda Li in Lyu [9] deluje na tak način, njena prednost pa je tudi, da za uspešno učenje ne potrebuje podatkovne množice dejanskih deepfake posnetkov. Ustvarjanje velike množice ponarejenih deepfake posnetkov je zamudno, tukaj pa se ponarejanje lahko simulira s preprostimi funkcijami za obdelavo slik, kot je gaussovo megljenje na naključnih točkah obraza. Podobne motnje se namreč pojavljajo v dejanskih ponaredkih iz drugih podatkovnih množic.

Yang in drugi [11] predlagajo metodo, ki se osredotoča na neskladnosti v položaju glave osebe v 3D prostoru, ki je ocenjen na podlagi 68 določenih točk na obrazu. Cevovod za generacijo Deepfake ponaredkov ima pogosto pomanjkljivosti, ki vodijo do neskladnih položajev ali manjkajočih detajlov na glavi, ki jih lahko prepozna tudi metoda podpornih vektorjev.

Uporaba analize neuniformnosti odziva fotografij (photo response non-uniformity - PRNU) tudi obeta. PRNU je šumni vzorec ki nastane zaradi tovarniških defektov v proizvodnji fotoreceptorjev za digitalne kamere, in je različen za vsako snemalno napravo. Orodje se pogosto uporablja pri forenziki digitalnih slik, ker zamenjan obraz spremeni vzorec šuma na tem delu slike. Koopman, Rodriguez in Gerardts [6] predstavljajo metodo, ki posamezne sličice iz posnetka razreže v osem delov, za katere potem izračuna povprečni PRNU vzorec. Z merjenjem korelacije med deli slike lahko oceni, ali je slika ponarejena ali ne. V poskusih z 10 izvirnimi in 16 ponarejenimi posnetki, ustvarjenimi z orodjem DeepFaceLab, je bila povprečna statistična razlika v korelaciji veliko višja pri ponarejenih posnetkih kot pri izvirnih. Metoda obeta, a jo je treba še preizkusiti na večji podatkovni množici.

Hasan and Salah [4] po drugi strani ponujata pristop, ki se povsem izogiba strojnemu učenju. Pri preverjanju ponaredkov nas zanima izvirni posnetek, a trenutno še ni orodja, s katerim bi ga poiskali. Predpostavka, da je posnetek izviren samo, ko je njegov vir izsledljiv, omogoča vpeljavo tehnologij Blockchain in pametnih pogodb v postopek prepoznavanja lažnih posnetkov. Vsak posnetek bi bil povezan z pametno pogodbo, ki ga povezuje z starševskim posnetkom, in vsak starš ima povezave do svojih otrok v hirearhični strukturi. V tej hierarhiji lahko uporabniki izsledijo izvor ponarejenega posnetka. Unikatne razpršitvene funkcije datotečnega sistema, v katere shranimo videoposnetke in njihove metapodatke, omogočajo varnost. Ta pristop se lahko razširi tudi na druge tipe datotek, kot so slike, besedilni dokumenti, tabele, itd.

5. SKLEPI

Deepfake tehnologija je omajala zaupanje v verodostojnost medijev. Prosta razširjenost metod za ponarejanje slik in hitro širjenje, ki ga omogočajo družbena omrežja, vzbujata zahtevo po razvoju dobrih metod za prepoznavanje ponaredkov. V našem prispevku smo predstavili nekaj sodobnih metod, ki slonijo na različnih pristopih. Vredno je opozoriti, da različne metode razkrivanja Deepfake-ov delujejo bolj

ali manj zanesljivo tudi glede na samo metodo, ki je bila uporabljena za spreminjanje slik ali videoposnetkov. Zato bomo verjetno morali določiti več različnih metod testiranja, preden bomo lahko zanesljivo ugotovili ali gre za verodostojen dokaz.

Pričakujemo lahko tudi, da se bodo Deepfake tehnologije v prihodnosti še izboljšale, zato se morajo metode odkrivanja ponaredkov razvijati in izboljšati skupaj z njimi. Smeri, v katere naj ta stroka raste, je precej; potrebujemo podatkovne množice za robustno preizkušanje novih metod, razumeti je treba pristope, ki jih zlonamerni agenti uporabljajo pri ustvarjanju ponaredkov, ter integrirati metode teh raziskav direktno v portale za deljenje vsebin, kot so družabna omrežja. V tej smeri so vse rešitve še vedno v povojih.

Ob tehničnem prepoznavanju prepoznavanj ponaredkov pa je pomembno tudi razumevanje namena posameznikov, ki oblikujejo in širijo Deepfake vsebine. Video in avdio posnetki ter fotografije se pogosto uporabljajo kot dokazi v sodnih procesih, zato je ključno razumevanje samega postopka za ponarejanje in tudi družbenega konteksta, v katerem je nastala Deepfake vsebina.

6. LITERATURA

[1] Full body deepfakes are the next step in ai-based human mimicry. `https://www.fastcompany.com/90407145/youve-been-warned-full-body-deepfakes-are-the-next-step-in-ai-based-human-mimicry`. Accessed: 2020-05-10.

[2] C. Bregler, M. Covell, and M. Slaney. Video rewrite: Driving visual speech with audio. pages 353–360, 1997.

[3] D. Güera and E. J. Delp. Deepfake video detection using recurrent neural networks. In *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 1–6, 2018.

[4] H. R. Hasan and K. Salah. Combating deepfake videos using blockchain and smart contracts. *IEEE Access*, 7:41596–41606, 2019.

[5] C.-C. Hsu, Y.-X. Zhuang, and C.-Y. Lee. Deep fake image detection based on pairwise learning. *Applied Sciences*, 10:370, 01 2020.

[6] M. Koopman, A. Macarulla Rodriguez, and Z. Geradts. Detection of deepfake video manipulation. 08 2018.

[7] P. Korshunov and S. Marcel. Vulnerability assessment and detection of deepfake videos. In *The 12th IAPR International Conference on Biometrics (ICB)*, pages 1–6, 2019.

[8] Y. Li, M. Chang, and S. Lyu. In ictu oculi: Exposing ai created fake videos by detecting eye blinking. In *2018 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–7, 2018.

[9] Y. Li and S. Lyu. Exposing deepfake videos by detecting face warping artifacts. *arXiv preprint arXiv:1811.00656*, 2018.

[10] E. Sabir, J. Cheng, A. Jaiswal, W. AbdAlmageed, I. Masi, and P. Natarajan. Recurrent convolutional strategies for face manipulation detection in videos. *Interfaces (GUI)*, 3:1, 2019.

[11] X. Yang, Y. Li, and S. Lyu. Exposing deep fakes using inconsistent head poses. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8261–8265. IEEE, 2019.

Learning Rich Features for Image Manipulation Detection review

Nermin Jukan
Fakulteta za računalništvo in informatiko
Večna pot 113
Ljubljana, Slovenija
nj2245@student.uni-lj.si

Gregor Ažbe
Fakulteta za računalništvo in informatiko
Večna pot 113
Ljubljana, Slovenija
ga4588@student.uni-lj.si

ABSTRACT

This document describes a novel method for image manipulation detection proposed by Zhou et al. [1]. The described method uses a faster region-based convolutional neural network (R-CNN) for learning rich features from images that have clues of image manipulation. To achieve better performance it uses two streams. The first is an RGB stream and the second is a noise stream, which is derived from the RGB stream with the help of steganalysis rich model (SRM) filters. The RGB stream is used to find strong contrast differences, unnatural boundaries and so on. At the end of these streams, bilinear pooling is used for joining the two streams together. After presenting and describing this novel method, the paper describes experiments made by the authors of the novel method. These experiments show that using both streams gives better results than just using an individual stream.

Keywords

image forensics, image manipulation detection, faster R-CNN, deep learning

1. INTRODUCTION

Because of free, user friendly image manipulation editors, image manipulation became very widespread and it poses a very big issue to public security. It has become a huge challenge for image forensics to provide good image manipulation detection software. The most common image tampering techniques are copy-move, splicing and removal.

Copy-move is copying one part of an image and moving it to another part of an image. Splicing is moving a part of an image to another image. Removal is removing a part of an image followed by inpainting the region of removal. These techniques are represented in Figure 1.

Current methods usually do not detect all of these techniques. To address this problem, authors of the reviewing article [1] proposed a new method for image manipulation detection. It uses both RGB and noise streams to detect manipulated regions and classify the manipulation technique. The RGB stream is used to detect tampered edges with capturing clues like visual inconsistencies at tampered boundaries and contrast effect. But this is not enough for example in case of the removal tampering process. For this reason the noise stream has been added. With the noise stream, we can detect some inconsistencies in the local noise of an image. Every image has its specific noise and when an image is tampered, the noise of that image is disrupted. With noise analysis we can detect tampered parts of an image.

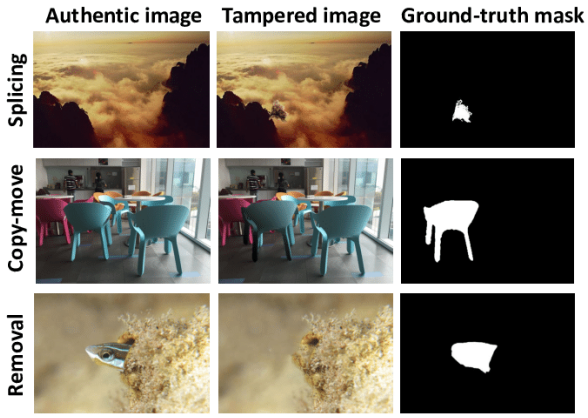


Figure 1: Tampering techniques splicing, copy-move and removal.

2. RELATED WORK

In the field of digital forensics, there are many various techniques to detect tampered regions of an image. One of the most commonly used techniques is the double JPEG comparison [2]. Unlike the method described in the article, most of other methods are not so robust and are limited to specific tampering manipulations and techniques.

Other methods are commonly based on colour filter array (CFA) analysis. One of these methods is Gaussian mixture model (GMM), which classifies tampered and untampered regions by finding regions that disturb or do not disturb CFA patterns.

Local noise analysis is frequently used for discovering tampered regions. The method which is described in this article uses SRM kernels for local noise analysis. SRM features can be combined by including quantization and truncation operations with a convolutional neural network (CNN). Rao et al. [3] proposed the use of SRM filter kernel to boost CNN detection accuracy.

Many techniques today use deep learning for detecting tampered regions on an image. Most of them use a CNN. The method described in this article uses a faster R-CNN. Salloum et al. [4] use a fully convolutional network (FCN) to directly predict the tempering mask.

A long short term memory (LSTM) network based architecture is applied to small image patches to find the tampering artifacts on the boundaries between tampered patches and the image patch. It is trained with pixel level segmentation.

3. PROPOSED METHOD ANALYSIS

The authors propose a multi-task framework, that can simultaneously detect manipulated regions and classify the type of manipulation for a certain region. Two separate streams are implemented. Features from the two streams are then combined using bilinear pooling. The fully connected layer is then capable of classifying manipulated regions, while also using the RGB stream and passing it trough an Region Proposal Network (RPN), allowing it to detect local tampered regions.

3.1 RGB stream

The RGB stream in the described method is used for detecting mid- and high-level visual artifact of tampered image. It is a single Faster R-CNN network. Authors of this method use ResNet 101 network for learning features from the RGB stream.

With this method we can get bounding boxes of manipulated regions and classify the manipulation technique. The last layer of the CNN is used for manipulation classification. RPN is a component of the Faster R-CNN network and is used for detecting Region of interest (RoI). Unlike in the case of object detection, our RoI is a region where an image is potentially tampered. It is not necessarily an object, e. g. in case of object removal.

3.2 Noise stream

Local noise distributions are added as a separate stream to overcome the inability of the RGB stream to detect tampered images that were carefully post processed to conceal the splicing boundary and reduce contrast differences.

Because the noise stream is designed to pay more attention to noise rater than semantic image features, it can introduce new evidence or features into the detection network. This is a novel approach which has not been used before in any type of image detection problems. Popular steganalysis rich model (SRM) filters are used to extract local noise features.

Noise modeling is performed by comparing a pixel's value with the result of interpolating the same pixel only to it's neighboring pixels. Therefore we can compare the two results and determine the noise levels between the actual pixel values and the interpolated values. The process is initialized with 30 basic filters including nonlinear operations such as minimum and maximum of the nearby outputs after filtering. The outputs of these filters are then quantified and truncated by SRM and the final features are gathered by extracting the nearby co-occurrence information.

$$\frac{1}{4}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 2 & -4 & 2 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \frac{1}{12}\begin{bmatrix} -1 & 2 & -2 & 2 & -1 \\ 2 & -6 & 8 & -6 & 2 \\ -2 & 8 & -12 & 8 & -2 \\ 2 & -6 & 8 & -6 & 2 \\ -1 & 2 & -2 & 2 & -1 \end{bmatrix} \quad \frac{1}{2}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 2: The three SRM filter kernels used to process images and extract noise features.

Different numbers of kernels were used to see whether an increase in the number of kernels provides better performance. The findings showed that the optimum number of kernels is 3 and that higher numbers of kernels do not provide any significant performance gain. The kernel sizes were set to $5 \times 5 \times 3$, corresponding to width, height and the number of colour channels (RGB). The kernels, shown in Figure 2, are fed directly into a pre-trained network trained on 3-channel inputs. These kernels are responsible for estimating a pixels value by interpolating the values of neighboring pixels. The output of the SRM layer is also 3-dimensional.

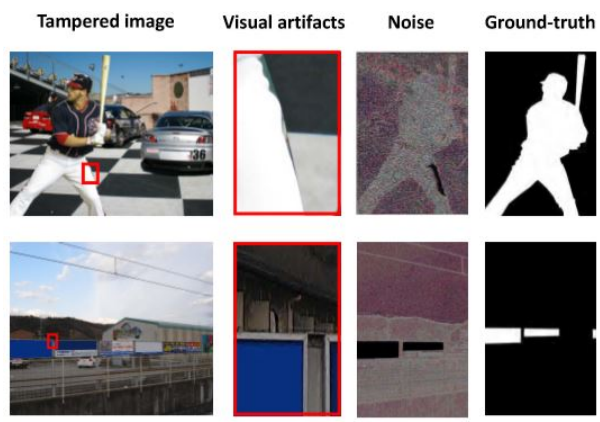


Figure 3: Visual presentation of tempering artifacts. The third column represents the artifacts, which were located using local noise features.

We can observe the results of the SRM layer in Figure 3. The second column represents the amplification of the red bounding boxes in the first column. The third column represents the emphasized local noise regions, where tempering artifacts are visible (black items). Detecting these artifacts in the RGB channels would not be possible. The last column shows the ground truths for each image. The rest of the noise stream configuration is shared with the RGB stream, this includes the convolutional network and the RoI pooling layer. Noise features are excluded from bounding box regression, because RGB stream features are more appropriate.

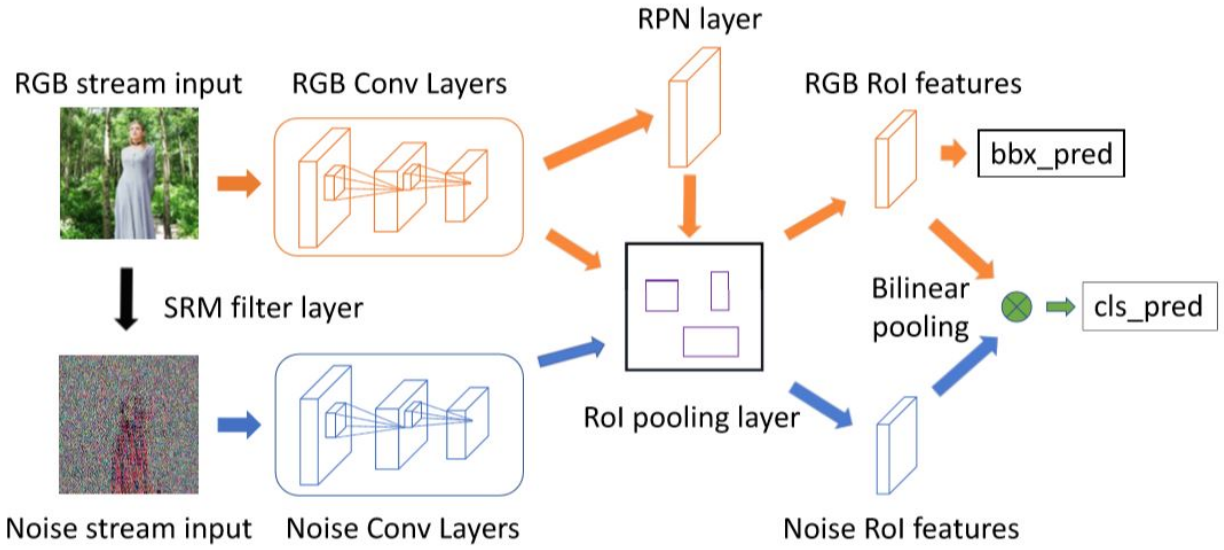


Figure 4: Illustration of the two-stream Faster R-CNN network. The blue arrow path represents the noise stream and the orange arrow path represents the RGB stream.

3.3 Stream combining

The RGB and the noise streams are combined into a common stream by performing some simple operations. Figure 4 represents the complete pipeline. It shows how the RGB stream input is passed trough a series of SRM filters to create the noise stream input. The RGB stream features coming out as a result from the RGB stream's convolutional layers are passed on one side to the RPN layer and to the RoI pooling layer. RGB features are used for predicting bounding boxes for RoI and also for classifying the type of manipulation. In more detail, bilinear pooling combines streams in a two-stream CNN network while preserving spatial information to improve the detection confidence. The output of this operation is a result of multiplying RoI both of the RGB and the noise stream. Sum pooling is then performed to squeeze the spatial features before classification. Signed square root and L2 normalization are applied and the result is forwarded to the fully connected layer. Compact bilinear pooling is used to save memory and speed up training. Following the fully connected and softmax layers, prediction classes for the RoI are obtained as depicted in Figure 4. Noise stream features are only used for the manipulation type classification. Cross entropy loss is used for manipulation classification and smooth L1 loss for bounding box regression. The final loss function is calculated by summing up the individual losses.

3.4 Implementation details

The original paper does not provide any details on what type of programming language and additional libraries were used to implement the network, however, we were able to find an existing implementation online [5]. This project used *Python 3.6* and *TensorFlow 1.8.0*. The results of this project can be seen in Figure 5.

Figure 5: Results obtained with a proposed implementation [5] of the method described in the original paper [1].

End-to-end training is done on images and extracted noise features. Both of these are re-sized so that the shorter length equals 600 px. Four anchor scales of sizes 8^2, 16^2, 32^2 and 64^2 are used alongside aspect ratios of 2:1, 1:1 and 1:2. Feature sizes of $7 \times 7 \times 1024$ are the same after RoI pooling for both RGB and noise streams. Batch sizes equal 64 for training and 300 for testing. Data augmentation uses image flipping. The intersection-over-union threshold is set to 0.7 for positive examples and to 0.3 for negative examples. The learning rate is dynamic and is initially set to 0.001. After 40,000 steps the learning rate is decreased to 0.0001. The model is trained for 110,000 steps and applies non-maximum suppression with a threshold of 0.2 to reduce the redundancy of proposed overlapping regions.

4. EXPERIMENTS ANALYSIS

The two streamed network was tested on standard datasets and compared to state of the art technologies. Different data augmentation methods and robustness towards resizing and JPEG compression were also testes.

4.1 Pre-trained model details

Because standard datasets did not have enough data for deep neural network training, the authors created their own synthetic dataset. The dataset was automatically created using the images and annotations from COCO database [6]. Items from images in the dataset were randomly selected and pasted onto other images. A split of 90% for training and 10% for testing is formed to assure that no tampered object is assigned to the same background in both the train and test set. The resulting set contained 42,000 tampered and authentic images.

AP	Synthetic test
RGB Net	0.445
Noise Net	0.461
RGB-N noise RPN	0.472
Noise + RGB RPN	0.620
RGB-N	**0.627**

Table 1: Different model variations and their performance scores on the Synthetic dataset.

The model outputs bounding boxes with confidence intervals describing whether the detected objects were manipulated or not. A margin of 20 pixels was added to the default bounding boxes upon training, allowing both the RGB and noise streams to better detect these anomalies. A pre-trained ResNet 101 was used for the Faster R-CNN. Different variations of the proposed network were then analysed and the results are shown in Table 1. As seen from the last three rows, the model with both RGB and noise features as input of the RPN network leads to weaker performance than just using RGB features.

4.2 Testing on standard datasets

Several different datasets were used to test the implemented model. Not all three datasets contained all image manipulation types, however all databases contain ground truth bounding boxes for their respective samples or original images, from which the bounding boxes could be obtained using different extraction methods. NIST16 database [7] contains all three types of manipulations. On top of that, all of the samples are also post-processed to conceal manipulation traces. The CASIA database [8] provides spliced and copy-moved image samples, which are also post-processed with blurring filters. The COVER database [9] is a smaller dataset containing only copy-move samples of different objects. [10] is the final testing dataset. It is composed of uncompressed images containing splicing manipulations.

The evaluation metrics that were chosen to compare the results are *pixel level F_1 score* and *area under the receiver operating characteristic curve (AUC)*. The performance of the newly developed model was compared to a series of state of the art models ([11], [12] [13], [14], [15]), which were used as a baseline for comparison.

Tables 2 and 3 show that the newly designed model outperforms all other solutions on almost every dataset that was used for testing. This is because other models only rely on one type of features, mainly on RGB-like features, while the new model incorporates noise stream features as well. Other approaches for improving the model were also investigated. On the part of data augmentation, several methods were analyzed. The results of data augmentation methods are shown in Tables 4 and 5. The authors also tested the robustness to JPEG and resizing attacks. The tests on the NIST16 database confirmed that the proposed solution is better at detecting these attacks than other models.

	NIST16	Columbia	COVER	CASIA
ELA	0.236	0.470	0.222	0.214
NOI1	0.285	0.574	0.269	0.263
CFA1	0.174	0.467	0.190	0.207
MFCN	0.571	0.612	-	**0.541**
RGB Net	0.567	0.585	0.391	0.392
Noise Net	0.521	**0.705**	0.355	0.283
Late Fusion	0.625	0.681	0.371	0.397
RGB-N	**0.722**	0.697	**0.437**	0.408

Table 2: F_1 score results the four standard datasets.

	NIST16	Columbia	COVER	CASIA
ELA	0.429	0.581	0.583	0.613
NOI1	0.487	0.546	0.587	0.612
CFA1	0.501	0.720	0.485	0.522
J-LSTM	0.764	-	0.614	-
RGB Net	0.857	0.796	0.789	0.768
Noise Net	0.881	0.851	0.753	0.693
Late Fusion	0.924	0.856	0.793	0.777
RGB-N	**0.937**	**0.858**	**0.817**	**0.795**

Table 3: *Pixel level AUC* comparison on four standard datasets.

F_1	NIST16	COVER	CASIA
Flipping + JPEG	0.712	0.425	**0.413**
Flipping + noise	0.717	0.412	0.396
Flipping	**0.722**	**0.437**	0.408
No flipping	0.716	0.312	0.361

Table 4: Data augmentation comparison using F_1 score.

AUC	NIST16	COVER	CASIA
Flipping + JPEG	**0.950**	0.810	0.785
Flipping + noise	0.947	0.801	0.776
Flipping	0.937	**0.817**	**0.795**
No flipping	0.940	0.793	0.766

Table 5: Data augmentation comparison using *AUC* score.

The new network is capable of also distinguishing between different manipulation types. The results yield that the easiest manipulation to detect is splicing, because it is likely to produce both RGB and noise artifacts. Removal is the second easiest manipulation, because it leaves behind a high amount of noise features. Copy-move is the hardest manipulation to detect, mostly because the element that is being copied is from the same region, which yields similar noise levels. On the other hand, the two regions have the same contrast levels.

5. CONCLUSION

The authors of the reviewed paper presented a novel approach towards finding and classifying manipulated regions in images. The two-stream approach allowed them to build a model which uses RGB and noise features to detect regions of interest. They tested the model on four different datasets and compared the results to state of the art solutions. The results proved that the proposed model outperformed other models on all dataset tests. The authors also analyzed the classification problem for classifying manipulation types. The results showed that the proposed model is robust against all manipulation types and is also robust towards image compression and resize attacks. In addition to the conducted tests and the obtained results, we would also suggest experimenting on using the model not only for detecting static manipulation techniques, but perhaps also test the model on dynamic inputs like deepfakes.

6. REFERENCES

[1] Peng Zhou, Xintong Han, Vlad I Morariu, and Larry S Davis. Learning rich features for image manipulation detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1053–1061, 2018.

[2] Tiziano Bianchi, Alessia De Rosa, and Alessandro Piva. Improved dct coefficient analysis for forgery localization in jpeg images. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2444–2447. IEEE, 2011.

[3] Yuan Rao and Jiangqun Ni. A deep learning approach to detection of splicing and copy-move forgeries in images. In *2016 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–6. IEEE, 2016.

[4] Ronald Salloum, Yuzhuo Ren, and C-C Jay Kuo. Image splicing localization using a multi-task fully convolutional network (mfcn). *Journal of Visual Communication and Image Representation*, 51:201–209, 2018.

[5] Hangyan Jiang. Image manipulation detection. https://github.com/LarryJiang134/Image_manipulation_detection, 2018.

[6] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[7] Nist nimble 2016 datasets. https://www.nist.gov/itl/iad/mig/nimble-challenge-2017-evaluation/. Accessed: 2020-04-26.

[8] J. Dong, W. Wang, and T. Tan. Casia image tampering detection evaluation database. In *2013 IEEE China Summit and International Conference on Signal and Information Processing*, pages 422–426, 2013.

[9] B. Wen, Y. Zhu, R. Subramanian, T. Ng, X. Shen, and S. Winkler. Coverage — a novel database for copy-move forgery detection. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 161–165, 2016.

[10] Yu-Feng Hsu and Shih-Fu Chang. Detecting image splicing using geometry invariants and camera characteristics consistency. *2006 IEEE International Conference on Multimedia and Expo*, pages 549–552, 2006.

[11] Neal Krawetz. A picture ' s worth . . . digital image analysis and forensics. 2007.

[12] P. Ferrara, T. Bianchi, A. De Rosa, and A. Piva. Image forgery localization via fine-grained analysis of cfa artifacts. *IEEE Transactions on Information Forensics and Security*, 7(5):1566–1577, 2012.

[13] Md Jawadul Bappy, Amit Roy-Chowdhury, Jason Bunk, Lakshmanan Nataraj, and B. Manjunath. Exploiting spatial structure for localizing manipulated image regions. 10 2017.

[14] Babak Mahdian and Stanislav Saic. Using noise inconsistencies for blind image forensics. *Image Vision Comput.*, 27(10):1497–1503, September 2009.

[15] Ronald Salloum, Yuzhuo Ren, and C.-C. Jay Kuo. Image splicing localization using A multi-task fully convolutional network (MFCN). *CoRR*, abs/1709.02016, 2017.

6 Razno / Miscellaneous

Leveraging Electromagnetic Side-Channel Analysis for the Investigation of IoT Devices [5]

Jovan Toroman
Faculty of Computer and Information Science
University of Ljubljana
Ljubljana, Slovenia
jt3485@student.uni-lj.si

Uroš Hercog
Faculty of Computer and Information Science
University of Ljubljana
Ljubljana, Slovenia
uh8243@student.uni-lj.si

ABSTRACT
The area of IoT (Internet of Things) devices is growing by day and they are becoming a vital part of our lives. Consequently, we are more likely to find them at a crime scene, and analysing them could provide insight into the events that unfolded. The article proposes a novel method of analysing such devices and extracting relevant data without destroying the device. The authors suggest that these methods might be part of existing digital investigation flows. [4].

Keywords
IoT, Side-Channel, Electromagnetic Analysis

1. INTRODUCTION
Usage of IoT devices is gaining traction in almost all aspects of our lives. These devices provide interesting functions to their users, such as health monitoring, personal assistant, smart light bulbs, smart household devices, among others. But besides being useful, these devices can also represent an invaluable source of data in a criminal investigation. Besides devices themselves, we can also get data from devices connected to IoT devices (e.g. smartphones) or from remote cloud services which the device communicates with. But both sources of data have their challenges. Gathering data from the devices themselves is challenging because of their non-standard interfaces. Most times, the only way to getting this data is by dismantling the device to reach the flash memory, or even using chemicals to expose the silicon wafer of the device. Obtaining data from the devices which gather data from the IoT device (e.g. smartphones) or its cloud service can be unreliable because of different factors. Namely, we have no guarantee that no one has tampered with the software on the IoT device. If someone altered the software, that would mean that the data on remote devices or cloud cannot be considered completely relevant. It would be useful if we had a way of determining if the devices code was changed, without accessing the device's storage directly. An interesting way of doing this is by using unintentional electromagnetic (EM) radiation from IoT devices. EM radiation schemes from CPUs of IoT devices sufficiently correlate to their software activities. Using Raspberry Pi and Arduino Leonardo devices as representative IoT target devices, authors of this paper show that we can observe various forensically useful software behaviours. Authors show that cryptographic algorithms running on an IoT device can be detected with 82% accuracy while modifications of software can be detected with 90% accuracy through a combination of EM-SCA (Electromagnetic side-channel analysis) techniques and machine learning in a real-world environment.

Authors of the original paper present three contributions of their work:

- **EM-SCA** This approach has great potential for gathering forensically useful information from IoT devices in an uninvasive way
- **Machine learning** Due to the complex nature of this electromagnetic radiation, authors utilise machine learning to determine if a device's software was altered. With this approach, they achieve 80% accuracy
- **Integration** Besides developing the techniques themselves, authors also propose a way to integrate their technique with existing procedures in digital forensics.

2. RELATED WORK
Electromagnetic radiation is everywhere around us. It's generated by more or less every device powered by electricity. There is more than one component in every such device that generates electromagnetic radiation. These components range from processors to memory cards. All electromagnetic waves generated by these components differ in frequency, amplitude and phase. The subtle variations in these properties stem from the nature of a component and also the amount of current this device consumes in a given time period. In the work *Side-channel Vulnerability Factor: A Metric for Measuring Information Leakage* by Demme et al., [2] the authors present a new metric called Side-channel Vulnerability Factor. It describes and quantifies the amount of information a given either device or a component leaks. They base this metric on the notion that many operations, a component performs, are executed in sequences called patterns. This means the sequences of for example CPU instructions are not random, but executed in specific orders.

An example of such a pattern can be the operation of establishing a secure connection with a remote server. The session key might not be the same every time, but the sequence of instructions a device performs is constant and repeatable. This metric represents a correlation between the actual pattern and the pattern the attacker reconstructed based on the collected electromagnetic radiation emitted by the victim's device. How he did this is out of scope. The lower the correlation, the less likely it is that the attacker reconstructed the actual pattern. A device leaking a lot of information could become dangerous to its user because once the attacker manages to reconstruct the pattern he can focus his attention to the slight variations of the radiation. From this, he could technically deduce the session key and exploit the user.

But this metric by itself does not provide a lot of insight into the quantity of information an attacker extracted by analysing the electromagnetic radiation emitted by the victim's device. Every component is different, so a general method is difficult to devise. But if we limit the analysis to CPUs such a method exists. Yilmaz defines it in a paper *Capacity of the EM Covert/Side-Channel Created by the Execution of Instructions in a Processor* [6]. The method is based on statistical analysis of electromagnetic radiation emitted during the execution of specific single instructions or sequences of them. Such an analysis is executable in a real-word scenario. Bhe problem arises before that, on data collection. There is a precondition that we as an attacker can install purpose-made software on the victim's device. This helps us because we know which instructions this program will execute. We have to then wait for it to run and collect the data. Based on the collected data points we can then, using statistical methods, deduct the signature or the shape of a specific wave caused by a specific instruction. Once we know this we can figure out, again using statistical methods, what is running on the victim's device and potentially steal sensitive information. But this method is based on a lot of very highly unlikely preconditions and is deemed impractical. I<t provides a glimpse on how electromagnetic radiation could interfere with a by day more important technologies, cryptography and IoT. Many of these small always-on and connected devices utilise encryption to safely move the data to the back-end. Incorrect implementations of encryption procedures can lead to information leakage and consequently stealing of private keys, given these devices use asymmetric encryption [3]. To execute such an attack, we would only need to be in a radius of a couple of meters from the victim's device.

We can use these concepts for good, such as using them in digital forensics. Classical procedures don't allow us to analyse the devices without getting direct access to them. Once we have the device, getting to the relevant information can cause irrecoverable damage which might sometimes not be desirable. Contrary to these classical approaches, EM-SCA can tell us what is going on a device without direct access and without turning the device off to analyse it.

3. EM-SCA ANALYSIS OF IOT DEVICES

Although many components of an IoT device emit EM radiation, the most significant amongst them is the device's processor. Because of the similarity between processors on most of these devices, patterns identified on one device can easily be applied to others. Authors conduct their study on representative devices, *Raspberry Pi 3 B* and an *Arduino Leonardo* (a high-end and a low-end IoT device). Authors present three aspects of their work: a hardware setup, experiments, and proving feasibility of their invention.

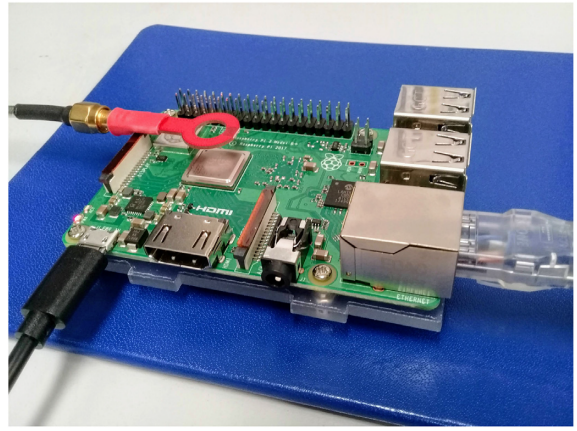


Figure 1: Setup for recording EM radiation [5]

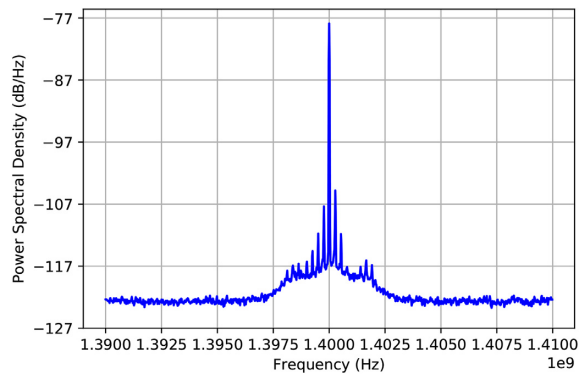


Figure 2: EMR intensity distribution over different frequencies in a sample of 0.01s length [5]

3.1 Observing EM emissions

For obtaining EM emissions, authors utilise a software-defined radio device called *HackRF*. We can observe it attached to a Raspberry Pi in Figure 1. To observe EM radiation, they had to determine the frequency of this radiation. This frequency mostly depends on the processor working frequency, with some lower-power harmonics of higher frequencies caused by other components like wireless and cable network chips, and baseline offsets caused by e.g. unstable electric supply. The most reliable frequency the authors found in the device was its base clock frequency of the processor which is 1.4 GHz. This frequency is represented as the highest spike in Figure 2. This means that power spectral density is at its highest for processor base frequency of 1.4 GHz.

Relying on the finding from previous work [1, 7] by Callan et al., which proves that operations being run on a processor can be inferred from the frequency of its EMR (Electroma-

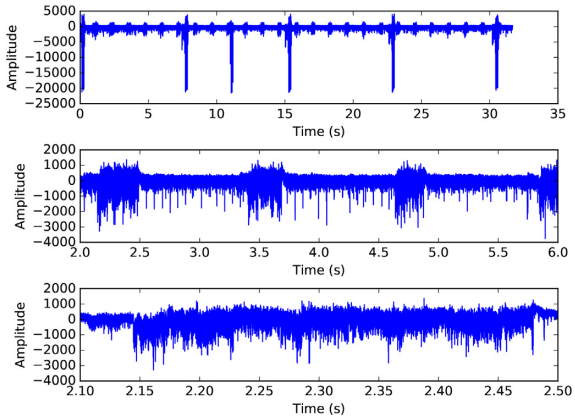


Figure 3: EMR at different time granularity. Higher frequency and smaller amplitude peaks represent AES computations, whereas larger peaks come from an unknown interference. [5]

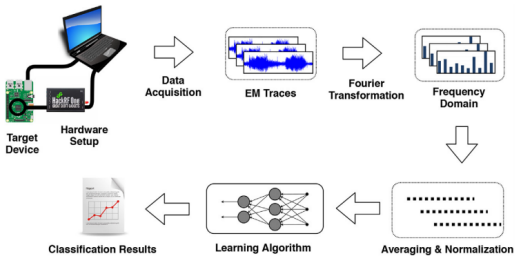


Figure 4: EMR acquisitions and preprocessing stages [5]

gnetic Response), authors perform an experiment to confirm this hypothesis applies to the devices in question. They run compute-intensive AES encryption on their Raspberry Pi in 1s intervals, simultaneously monitoring its EMR. Results are shown on a chart in Figure 3. Here the same sample is shown at different levels of granularity. The large peaks belong to an unknown external noise source, whereas the smaller, higher frequency peaks between them present actual peak in EMR because of AES computations.

3.2 Discriminating cryptographic activities

In this section authors try to infer the type of cryptographic activities based on EMR, since cryptography is usually of most interest in an investigation. Authors create a setup on which they run most widely used encryption algorithms, as well as some other operations for reference. Their setup is presented in Figure 4.

Authors require a labeled EM trace set to train the machine learning model. They establish a UDP communication channel between the Raspberry Pi and the host computer. To ignore all noise, the host computer is notified by the Raspberry Pi whenever it performs a cryptographic operation. In this way, the host computer can identify the time frame when the cryptographic operation happens.

Activity	Precision	Recall	F1-Score
Other	0.93	0.85	0.89
AES-256	0.78	0.86	0.82
AES-128	0.99	0.92	0.95
3DES	0.81	0.85	0.83

Figure 5: Classification accuracy of cryptographic algorithms

To account for variable lengths in the time-domain and improperly enclosed cryptographic operation within its boundary, authors perform the following pre-processing. Every trace is transferred into the frequency-domain by using a Fourier Transformation; more concretely Fast Fourier Transform with usage of a segment length of 0.1 s from the beginning of each EM trace. They also use only frequencies starting from 1/4 to 3/4 of the frequency spectrum, to get best quality samples.

3.2.1 Classification

Authors implement a neural network with 4 layers: input, output, and two hidden layers, having 10 and 5 neurons, respectively. Input layer has 500 neurons (number of features), whereas output layer has 4 (number of classes to predict). It outputs one of the widely used encryption algorithms (AES-256, AES-128, or 3DES) or "Other". There was 2400 training samples, equally distributed among all classes. After training, the network was evaluated with unknown inputs, and in Figure 5 we see that it performed with accuracy above 80% for all sorts or encryption algorithms, and other tasks.

3.3 Detection of software behaviour

Another application of this technology is detecting behaviour on IoT devices which is targeted at compromising forensic evidence, e.g. deleting data from the device, initiated from a remote location, or spying on the area of investigation using peripheral sensors like microphones or video camera.

Authors use a similar approach to solving this problem using machine learning. They achieve mean classification accuracy of over 90%, which can be observed in more detail in the confusion matrix in Figure 6. It contains classification results for programs subject to the experiment, which are labelled from 0 to 9 in the figure. As seen, most of the Arduino programs were detected by the classifier accurately.

3.4 Detecting modifications to firmware

The simplicity of IoT devices' software is what makes them susceptible to malicious modifications. It would be quite useful if we can detect these modifications through changes in their EMR.

There are two approaches to detecting these changes. First is an unsupervised learning approach where samples of both legitimate and compromised code are provided to the ML system. Another way is called *novelty detection*, which is trained only with legitimate data and ML system is then used to detect malicious code running based on how different it is from legitimate code. Authors choose the latter, because there is a very large number of possible code modifications

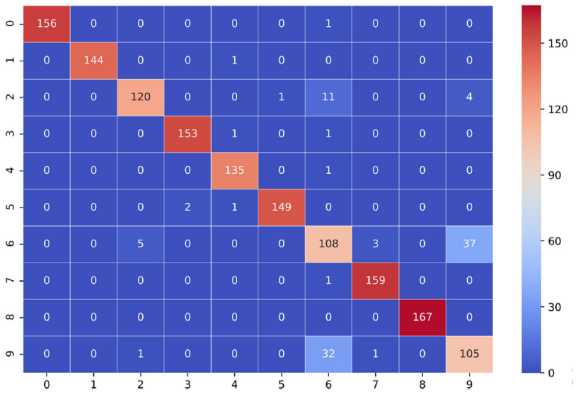


Figure 6: Confusion matrix of the neural net for detecting linear operations on an Arduino device [5]

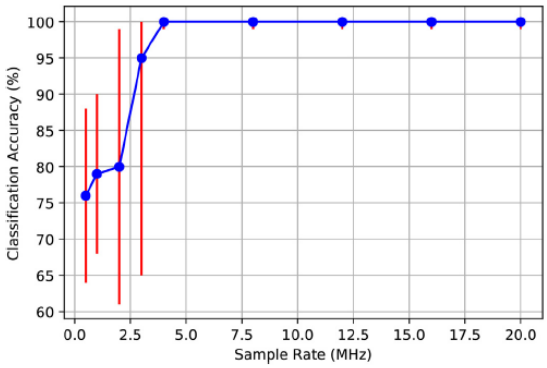


Figure 7: Accuracy of a 4 class classifier in relation to sampling rate of EMR [5]

of an IoT device and it would be hard to create enough test data to make this classifier feasible.

They also tackle problems of storage-intensive sampling process, by selecting an optimal sampling frequency in relation to accuracy. We can observe this in Figure 7.

Authors do not provide results they have obtained for this use case.

4. USING THE METHOD IN DIGITAL FORENSICS

The Internet of Things is a highly evolving area of small, always-on and connected to the Internet, which we can find everywhere around us. This also means we can find them on crime scenes and with this they fall under the domain of digital forensics. The experts in this area are dealing with new unknown scenarios, which make their work difficult. Most of these devices store no historical data on non-volatile drives such as Flash of NVM (Non-Volatile Media). This means these devices store user data, in most cases, only while powered on. The second problem investigators face is the fact that these devices are constantly connected to the Internet sending data to servers. This makes them highly unstable because someone with remote access permissions can remotely connect to the device and wipe its internal storage before the investigators can extract relevant information for their case. To overcome these issues, investigators usually rely on investigating the servers IoT devices are communicating with. But because of the way Internet works, there might be international laws that come in play and that makes investigations even more difficult. This is one more reason the analysis of the physical device the investigators have and its investigation is in their jurisdiction can prove to be an invaluable source of evidence.

But investigating a device means we have, after securing the device, to turn it off. By doing this we immediately lose relevant information stored on the devices volatile media. Another problem investigators face was already mentioned, encryption. Analysing devices utilising this can prove next to impossible. But if investigators were to use a method such as EM-SCA they could in specific circumstances circumvent these restrictions and correctly execute their investigation. The last problem we will talk about is the changeability of the data on the device. This means the more time the device on the crime scene is running, the higher the chance that the data it has will become irrelevant. So investigators have to act as fast as possible to recover the data from the volatile media from the device before turning it off.

The procedures we've talked about highlight some problems investigators face when investigating IoT devices, but also provide some insight into how these problems could be solved. But these methods are new and not battle tested so they cannot be used in real-world scenarios and consequently used as evidence in court.

5. CONCLUSION

Digital forensics typically deals with an analysis of traces left behind by attackers. These traces can be anything from files on storage devices, log and audit entries, network traffic traces and so on. With computing platforms getting more complex and widely adopted, they utilise even more complex methods to protect themselves and their users. Consequently, this creates a problem for the digital investigators as they have to overcome the protections of the well-intentioned manufacturers. Knowledge required to conduct a digital forensics investigation on such devices requires a sheer amount of specialised knowledge and experience. Cryptographically protected storage devices are a bane of such investigations. EM-SCA method shows its usefulness with exactly such cases because it offers us a glimpse into the inner workings of these protected devices.

Authors in the article present a method which gives us such power. Using two representative IoT devices they showed with a chain of experiments it is possible to deduce the inner state of a device and the processes that live within it. Experiment of classification of cryptographic operations showed that its possible to classify whether a device is using encryption with over 82% accuracy on high-end IoT devices, and over 90% accuracy on lower-end devices. Classifying whether a devices software was altered was done using 100% accuracy. The model was trained by feeding it 500 training samples of the authentic program produced during one of

the previous experiments. For testing, 100 additional samples of the authentic program were provided. Finally, when 20 different modified Arduino programs were provided for validation, each of them were detected by the model, yielding 100% accuracy.

Even though these methods were just developed and have not undergone any formal analysis and testing and their results cannot be used as evidence in criminal proceedings, they do show us that the world of digital investigations is getting more and more interesting by day. We believe that such methods will become a standard part of the digital investigator's toolbox.

6. REFERENCES

[1] R. Callan, A. Zajić, and M. Prvulović. A practical methodology for measuring the side-channel signal available to the attacker for instruction-level events. *47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 242–254, 2014.

[2] John Demme, Robert Martin, Adam Waksman, and Simha Sethumadhavan. Side-channel vulnerability factor: A metric for measuring information leakage. In *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, pages 106–117. IEEE, 2012.

[3] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Annual International Cryptology Conference*, pages 388–397. Springer, 1999.

[4] A. Sayakkara, N. Le-Khac, and Scanlon M. Leveraging electromagnetic side-channel analysis for the investigation of iot devices. *Digital Forensic Research Conference*, 2019.

[5] Asanka Sayakkara, Nhien-An Le-Khac, and Mark Scanlon. Leveraging electromagnetic side-channel analysis for the investigation of iot devices. *Digital Investigation*, 29:S94–S103, 2019.

[6] Baki Berkay Yilmaz, Robert L. Callan, Milos Prvulovic, and Alenka Zajić. Capacity of the EM covert/side-channel created by the execution of instructions in a processor. *IEEE Transactions on Information Forensics and Security*, 13(3):605–620, 2017.

[7] A. Zajić and M. Prvulović. Experimental demonstration of electromagnetic information leakage from modern processor-memory systems. *Transactions on Electromagnetic Compatibility*, 56(4):885–893, 2014.

Forensic analysis of water damaged mobile devices

Digital forensics, UL FRI 2019/20

Nejc Rebernik
UL FRI
nr3054@student.uni-lj.si

Jaka Kokošar
UL FRI
jk0902@student.uni-lj.si

ABSTRACT

Our lives are becoming more and more entangled with mobile electronic devices. Increased usage can lead to various types of physical damages that can occur. These can happen due to the negligence of their owners or in an effort of destroying crime evidence. In any case, we are interested in retrieving relevant data from such a device. Naturally, this is an important part of any digital forensic investigation. In an effort of destroying crime evidence, electronic devices are often destroyed by submerging in a liquid. To conduct a successful restoration of a device to an operating state it is important to understand electrochemical reactions that happen when a device is exposed to water. In this paper, we analyze the effects of moisture in combination with device electronics, discuss common repair methods and related works in this field.

Keywords

digital forensics, water damage, metal corrosion, mobile devices

1. INTRODUCTION

In this paper, we will briefly introduce the research area of water damaged devices [1]. The referenced paper mainly focuses on modern mobile devices because they arguably provide the most crucial pieces of information when conducting criminal investigations. We start by recognizing the key components of mobile devices that are subject to moisture-related damages. The next key step is knowing the process of metal corrosion that happens inside a mobile device. This includes identifying all the chemical reactions in combination with metal, moisture and constant supply of voltage. This is a relatively complicated process and needs understanding from a few different fields. Authors of the referenced paper conducted a lab experiment using working commercial smartphones to observe the types of damages caused by water. They then described the process of restoring the device and reported their success. Findings from such experiments can greatly improve the success rate of data extraction and also help the forensics investigators to properly handle water damaged devices on the field.

We will continue with a short overview of a current research state in this field. We then discuss the most common types of metal corrosion in a mobile device and look at the components and types of materials that are exposed to it. When we have a basic understanding of metal corrosion we can discuss the methods that are used in the process of restoring water damaged devices.

2. BACKGROUND

With the continued rise of mobile devices in forms of smartphones, smartwatches, other wearables, and sensors, the number of devices that incur water damage is also increasing. [2] Research is focused on evaluating the effects of water on corrosion of different metals and chip-off and chip transplantation methods for recovering data from damaged devices. Newer devices with water and dust resistance can withstand harsher conditions, which is favorable when attempting to recover data. We must take into account these properties when performing the analysis and recovery.

Research into corrosion is important for determining the ability of forensics to repair mobile devices by cleaning damaged logic boards. Even with newer, waterproofed devices, we must still consider the possibility of the seals leaking or even breaking, effectively rendering the device susceptible to water damage [3]. This leads to a need for improved materials that can resist corrosion better. Research in materials with greater corrosion resistance is being done and is an important step for mobile device durability, not only for the end-user but also for forensic analysis [4]. Major strides were made in discovering lead-free solder systems, increasing the corrosion resistance of devices using these materials.

Due to the proximity of electrical components and circuits on logic boards, sometimes the damage cannot be repaired simply by cleaning and drying the device. The water-induced corrosion damage inside the chips and short circuits on the board is sometimes too difficult to repair and requires a different approach. The water-induced corrosion might damage some chips or electrical lines on the board, rendering it unusable. In such scenarios, identifying the source of the problem is crucial. In such instances, alternative methods such as chip-off and chip transplantation have to be employed [5][6].

3. METAL CORROSION IN MOBILE DEVICES

In this section, we will summarize the basic parts of a mobile device. It is important to understand what are the key components in a device that contribute to and are exposed to the process of metal corrosion. We will continue with summing up the three major causes of metal corrosion.

3.1 Mobile device structure

Authors of the original paper describe the internals of a mobile device, which are important for understanding why the electrochemical process occurs in the first place. We will not go into the details here but will point out only the necessary key parts that are directly connected to water damages. The most important part of a modern mobile device is a printed circuit board (PCB). All of the peripheral components are connected to it. Conductive metals in the PCB are used as a channel for communication between peripheral devices in the form of electrical signals. The conductive materials inside PCBs are typically made of copper. All of the electrical components, such as resistors and capacitors, are soldered to the PCB using some mixture of metal materials. Those materials are typically tin (Sn), silver (Ag), and copper (Cu). And finally, there are electrodes that are located on the surface of a PCB. They are used to transfer electricity from a PCB to the electrical components or other devices located on the board. They are typically layered with highly conductive materials like nickel-gold (Ni/Au). To get a better understanding of this, the authors provided a nice picture (see figure 1) of a cross-sectional view of the PCB. This way we can look closely on the inside of the PCB itself. Finally, we need to consider that components on the PCB operate with different voltages. And that components on the PCB get closer with every generation. When we have this many components that are close to each other and operate with varying voltages we get different levels of electric potentials. And this is the key part when we talk about something like metal corrosion in a mobile device.

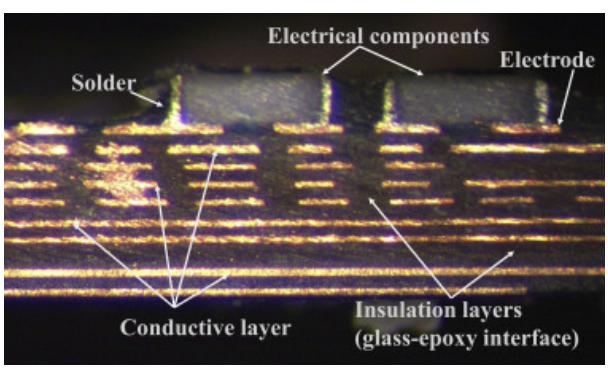


Figure 1: The cross-sectional view of the printed circuit board

3.2 Major causes of metal corrosion

The referenced paper states that manufacturers take a great interest in researching the conditions at which metal corrosion occurs. The combination of humidity, high temperature, and potentials biases have been widely researched to get a better understanding of this phenomenon.

3.2.1 Electrochemical migration

The electrochemical migration, or ECM for short, is one of the most common corrosion that is happening on the PCB in humid environments. ECM tests of solder alloys in pure water dates back to 1997 [7]. If some sort of fluid media is between a positive electrode (anode) and a negative electrode (cathode) the process of EMC can be observed. The metal on the positive electrode gets ionized. Then, because of the presence of water electrolysis, hydroxide metal is produced and deposited on the positive electrode. Because of the electrical potential, these metal ions travel to the negative electrode where they combine with electrons and migrate with the metals. Metal that is deposited on the negative electrode forms a shape of dendrites in the direction of the positive electrode. When it reaches back to the positive electrode a short circuit is formed (see figure 2). The process does not stop if all of the environmental factors are present. And while the negative electrode keeps losing solid metal even an open circuit can be formed.

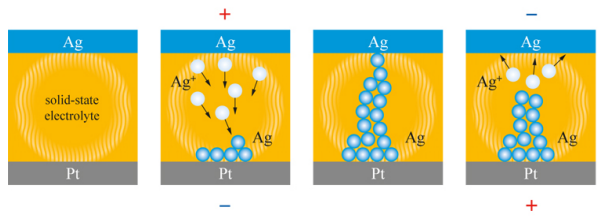


Figure 2: Visualized process of the electrochemical migration that is happening between silver (Ag) and platinum (Pt) metal materials.

3.2.2 Galvanic corrosion

Similarly, as in the previous section, the water acts as an electrolyte channel between different types of metals [8]. Galvanic corrosion [9] can occur even if there is no external supply of voltage. And because there are many types of materials on the inside of mobile devices, manufacturers have to use the right combination of metals or they need to protect them with some sort of coating. We differentiate between noble and less noble metals based on their electric potential. For example, using aluminum, which is a less noble metal, in combination with copper can lead to galvanic corrosion. Aluminum starts giving up electrons and gets ionized. These electrons then travel to the more noble metal in this case, copper. When electrons are moving between metals they make an electric current and this is how the noble metal (copper) keeps corroding. The process of galvanic corrosion is not important only in the scope of water damages in digital forensics investigations. This has great implications for example in building structures or water-cooling systems.

3.2.3 Conductive anodic filament

Conductive anodic filament (CAF) [10] happens inside on the inside of the PCB. PCBs contain multiple layers of copper (see Figure 1) that are used as a channel for electricity between components. Electricity potential is present between different copper layers. Trough the process of corrosion a short circuit can be formed inside the PCB. Naturally, these types of damages are harder to observe and can cause severe damage to the system.

4. REPAIRING WATER DAMAGE

Looking into repairing water damage in mobile devices we will talk about common approaches. We will describe two major techniques used when repairing water damage. This will also be shown in action by seeing a test of two mobile devices submerged in water (see Figure 3) and then repaired using these methods back into a working state.

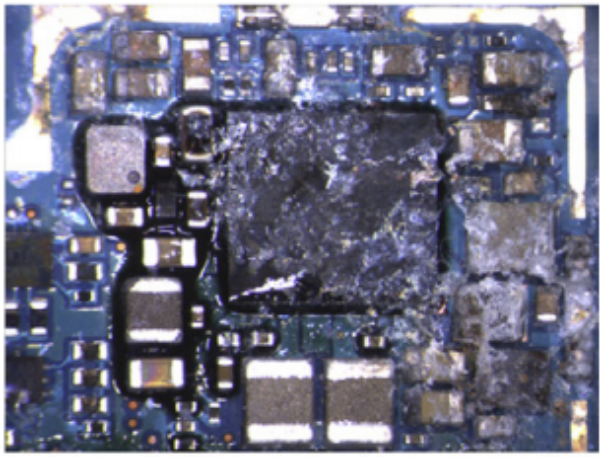


Figure 3: Water damaged PCB. Note the film created across components

4.1 Repair methods

In general, there are two approaches when repairing water damaged mobile devices. If the device has not been exposed to water for a long time or if the damage is not severe enough to damage the PCB and chips it is enough to disassemble, dry, and clean the device. If the device has been exposed to water for a longer period of time, creating short circuits and/or corrosion, chip-off has to be performed.

4.1.1 Drying and cleaning

Usually, if the device has not been exposed to water for a long period of time, it is enough to put it in a hygroscopic material such as rice or silica gel. In slightly more severe cases, the device has to be disassembled before it is dried and cleaned with ethanol and a soft brush.

4.1.2 Chip-off

Chip-off is a process of removing ICs from the PCB. It is the most difficult of the options for data extraction. The chip has to be heated to around 230 Celsius to melt the solder and adhesive. Then, the chip is extracted off the PCB with a blade, prying underneath the chip and lifting it or using a suction device (see Figure 4). Once removed, any remaining solder and glue have to be cleaned off to prevent them from sticking to and damaging the contact points. In a repair process, chip-off is used to remove existing damaged ICs (Integrated Chips) and replace them with donor parts from another device. Connecting solder and/or adhesive has to be removed before we can extract the chip. If the PCB and components are beyond repair, we must transfer the chip to another functioning device. This is then called chip transplantation. If a memory module is irreparably damaged due to internal short circuits or other reasons we cannot repair it and the data is lost. However, if the memory module is intact, we can use the chip-off and chip transplantation mentioned above to extract the data on another device.

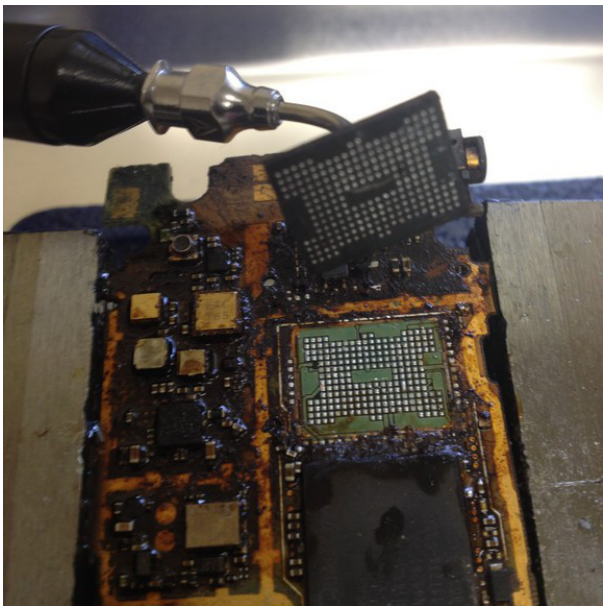


Figure 4: Chip-off process

4.1.3 Resoldering electrical paths

Excessive water damage can cause electrical paths between components on the PCB to break or become non-conductive. Due to structural damage to the PCB, it is necessary to resolder electrical paths or replace them entirely with a conductive wire such as copper if re-soldering is not possible.

4.2 Comparing test case devices

The researchers in the paper submerged 2 identical Samsung and 2 identical LG devices underwater for 72 hours. One device was turned on while the other was turned off. After 72 hours, the devices were disassembled, dried, and then attempted to get them to boot again. After disassembling, drying, and cleaning the device, only one of the four devices booted again. As highlighted before, drying and cleaning only go so far with devices that have been submerged in water for a longer period of time. Following is the account of advanced repair methods which were used to repair all three remaining devices.

The two Samsung devices in question are the Galaxy S6 Edge (SM-G925) model. Both of them have failed to boot after disassembly, drying, and cleaning, therefore a more thorough approach was required. PMICs(Power Management Integrated Chips) in mobile devices are responsible for power management tasks, such as AC to DC conversion, charging the battery and voltage regulation. The PMIC chips were found to be poorly connected to the PCB due to solder loss and short circuits and thus not working correctly. If a chip is internally damaged due to water damaged it is often impossible to repair it, as was the case here. The chip was removed and replaced with a donor part from another functioning device. After the PMIC and other damaged IC

chips were replaced with donor parts, both devices booted with all functions working.

The LG devices used in the test are the LG Nexus 5X (H790) model. The researchers were able to get the device that was submerged while turned off to boot by disassembling, drying, and cleaning the PCB. The device that was turned on when submerged did boot, however, the screen did not turn on. One could assume the LCD was damaged, but upon further inspection, it was evident that the screen was not getting the required power. A more thorough repair had to be done. Some of the display connectors were corroded, creating an open circuit, resulting in the screen not powering on. After the connectors were cleaned, parts of the PCB wiring had to be re-done. Because water damages the structure of PCBs, it was not possible to re-apply solder. The solution was to use thin copper wires to restore the electrical paths. This fixed the power issue and the device booted and the screen was working.

4.3 Contributing factors to repair difficulty

What was observed in the test was that different manufacturing practices affect water damage resistance. The LG device used underfill on the chips, which protected the IC chips from water damage. Underfill is typically used for increasing mechanical strength at the connections, but in this case, it also has a positive side effect of protecting the chip from water. But underfill is only part of the equation, as we can see multiple factors in the report, one of them being solder material, impacting the water-resistance and subsequent ability to repair.

Battery level and device state at the time of submersion play a significant part in determining the repair process. As previously mentioned current running through the circuit increases the rate of corrosion. From here we can work with parameters that determine the severity of corrosion. In the test, the batteries were fully charged before the devices were submerged in water. ECM can create a short circuit in a chip, creating a constant current flow. The more short circuits are created, the more current is flowing through, making the connections corrode faster. If the battery is fully charged and assuming it is not found before, it continues to discharge until it is empty, speeding up corrosion while it is discharging. The lower the charge of the battery, the sooner the process ends, creating less corrosion on the connections. Following from this, we see that it is favorable to have the battery as discharged as possible, to cause the least amount of ECM and corrosion [11]. Additionally, explaining why powered on devices were harder to repair, the state of the device is also an important contributing factor. Each activity on a mobile device such as playing music, using location services, downloading a file uses a certain amount of power. With every working activity, more power is drawn from the battery and more current is traveling through the circuits. With a bigger power draw, the ECM is happening faster, making the corrosion faster. If the device is recovered prior to the battery running out, it is desirable to have as little activity as possible since the power draw is lower, slowing down the corrosion.

5. CONCLUSIONS

Devices arrive at digital forensics labs in different states of disrepair. Longer exposure to water and devices being powered on increase the damage and make the repair process harder. While the forensic lab can't control what kind of devices they get, the people collecting them can. Chip-off is the last resort to repairing since it cannot always be done, due to unavailable replacement chips for rare devices, therefore we want to collect and preserve the devices as efficiently as possible to reduce possible malfunctions. Continuously improving and finding new ways of repairing devices is required in this ever-evolving field, as is communicating the big picture of the forensic analysis, especially the measures that reduce contamination and corrosion to the agents collecting these devices, so we can execute the repair as easy as possible and get the best results with data extraction.

6. REFERENCES

[1] Aya Fukami and Kazuhiro Nishimura. Forensic analysis of water damaged mobile devices. *Digital Investigation*, 29:S71 – S79, 2019.

[2] Rajan Ambat, Per Møller, and Peter Jacob Schwencke Westermann. *Corrosion and Environmental Effects on Electronic Systems*, page Abstract 800. The Electrochemical Society, 2006. Copyright The Electrochemical Society, Inc. [2006]. All rights reserved. Except as provided under U.S. copyright law, this work may not be reproduced, resold, distributed, or modified without the express permission of The Electrochemical Society (ECS).; ECS Meeting : D2 - Corrosion of Electronic Materials and Devices ; Conference date: 01-01-2006.

[3] Quanqing Yu, Rui Xiong, Chuan Li, and Michael Pecht. Water-resistant smartphone technologies. *IEEE Access*, PP:1–1, 03 2019.

[4] Feng Li, Vadimas Verdingovas, Balint Medgyes, and Rajan Ambat. Corrosion reliability of lead-free solder systems used in electronics. In *2017 40th International Spring Seminar on Electronics Technology (ISSE)*, pages 1–6. IEEE, 2017.

[5] Marcel Breeuwsma, Martien De Jongh, Coert Klaver, Ronald Van Der Knijff, and Mark Roeloffs. Forensic data recovery from flash memory. *Small Scale Digital Device Forensics Journal*, 1(1):1–17, 2007.

[6] Th Heckmann, Thomas Souvignet, S. Lepeer, and D. Naccache. Low-temperature low-cost 58 bismuth - 42 tin alloy forensic chip re-balling and re-soldering. *Digital Investigation*, 19, 10 2016.

[7] T. Takemoto, R.M. Latanision, T.W. Eagar, and A. Matsunawa. Electrochemical migration tests of solder alloys in pure water. *Corrosion Science*, 39(8):1415 – 1430, 1997.

[8] V. Novotny, G. Itnyre, A. Homola, and L. Franco. Corrosion of thin film cobalt based magnetic recording media. *IEEE Transactions on Magnetics*, 23(5):3645–3647, 1987.

[9] H.P. Hack. 2.07 - galvanic corrosion. In Bob Cottis, Michael Graham, Robert Lindsay, Stuart Lyon, Tony Richardson, David Scantlebury, and Howard Stott, editors, *Shreir's Corrosion*, pages 828 – 856. Elsevier, Oxford, 2010.

[10] Laura J Turbini and W Jud Ready. Conductive anodic filament failure: a materials perspective. In

Proceedings of the Third Pacific Rim International Conference on Advanced Materials and Processing, pages 1977–1982, 1998.

[11] Xiankang Zhong, Longjun Chen, Bálint Medgyes, Zhi Zhang, Shujun Gao, and László Jakab. Electrochemical migration of sn and sn solder alloys: A review. *RSC Adv.*, 7:28186–28206, 05 2017.

Digitalne forenzične prakse in metodologije za pametne asistente

Aljaž Nunčič
63160244
Fakulteta za računalništvo in informatiko
an2006@student.uni-lj.si

Grega Dvoršak
63160098
Fakulteta za računalništvo in informatiko
gd4667@student.uni-lj.si

POVZETEK

Pametni asistenti so lahko pomemben vir podatkov pri digitalni preiskavi, saj so običajno ves čas vklopljeni in čakajo na naš ukaz. Gre za naprave, ki so del tako oblačnih storitev, kot tudi interneta stvari. Pri njihovi preiskavi uporabimo pet metod. V vsaki izmed njih uporabljamo specifični pristop in orodja, kot rezultat pa dobimo nekatere podatke. Pri analiza paketov pametnih asistentov analiziramo HTTP pakete med asistentom ter oblakom, med tem ko pri analiza paketov Android mobilnih aplikacij analiziramo HTTPS pakete med Android mobilno aplikacijo ter oblakom. Ko analiziramo datotečni imenik, analiziramo podatke aplikacije, ki se običajno nahajajo v "/data/data". Ena izmed metod je tudi dekompilacija aplikacijskih paketov, kjer se ukvarjamo z analizo programske kode. Analiza odrezkov poteka na internem spominskem čipu asistenta. Pri analizi uporabimo več različnih programov, pri pametnem asistentu NAVER Clova pa si lahko pomagamo tudi z aplikacijo, ki so jo razvili avtorji originalnega članka.

Splošne oznake

Teorija

Ključne besede

Digitalna forenzika, pametni asistent, internet stvari (IoT), oblačne storitve, analiza

1. UVOD

Hitremu razvoju informacijske tehnologije mora v sodobnem življenju slediti tudi forenzika, saj starejši pristopi niso vedno uporabni pri novih rešitvah. Eden izmed takšnih primerov so tudi pametni asistenti z mikrofonom, kot je na primer vsem znana Alexa, Google asistent in ostali. Ker so te rešitve dokaj nove, se je v zadnjih letih pojavilo več člankov, ki predlagajo, kako naj se s temi pripomočki rokuje ob forenzičnih preiskavah, da pri tem dobimo čim več podatkov, ki jih potrebujemo, hkrati pa ne kršimo pravic uporabnikov. Eden izmed njih je članek avtorjev korejskih znanstvenikov z naslovom Digital Forensic Practices and Methodologies for AI Speaker Ecosystems [5]. Za te naprave je značilno, da združujejo dve področji v računalništvu. To sta internet stvari (IoT) in oblačno računalništvo. Avtorji so se v svojem članku osredotočili na štiri večje predstavnike pametnih asistentov v Koreji: Clova, Kakao I, NUGU in GiGA Genie. Verjetno ti predstavniki niso neposredno zanimivi za evropski prostor, a so si vsi asistenti med seboj sorodni, zato je postopek forenzične preiskave pri njih zelo podoben, razen nekaj specifik posamezne naprave. Ker gre za kompleksne naprave, so avtorji analizo pri preiskavi razdelili na pet metod. Vsako izmed njih bomo predstavili v nadaljevanju.

Za začetek pa si najprej poglejmo primer, ko je pametni asistent uspel rešiti življenje [10, 2]. Ko je fant prebral sporočilo, ki ga je prejela njegovo dekle, je pomislil, da ga vara in jo fizično napadel. Med tem jo je vprašal, ali je morda poklicala policijo, kar je naprava najverjetneje sprejela kot ukaz za klic. Med posnetki so kasneje našli tudi več ukazov, ko je žrtev želela poklicati policijo. Napadalec ji je grozil, da jo bo umoril, a ko je na svojem telefonu zagledal klic policistov, je bil tudi sam začuden.

2. PREGLED PODROČJA

V tem poglavju bomo pregledali nekaj člankov s področja digitalne forenzike pametnih asistentov. Pri tem se bomo osredotočili na najnovejše, saj so se pametni asistenti pojavili šele pred časom ter se področje še razvija. To pomeni, da mora temu slediti tudi način njihove preiskave in analize.

V prvem članku avtorji analizirajo dva trenutno najbolj prodajana asistenta: Alexo in Google Home [11]. Avtorji se ukvarjajo predvsem z analizo najbolj pogostih ukazov, kar nam kasneje pomaga odkriti, ali so pridobljeni dokazi resnični. Kriminalec lahko namreč pametnega asistenta uporabi tudi v svojo korist s tem, da ponaredi dokaze. To lahko naredi na primer tako, da spremeni ime pametnemu asistentu, ustvari samodejno ustvarjanje lažnih rutinskih aktivnosti in si s tem priskrbi alibi, ali pa razvije spretnost po meri, ki daje drugačen rezultat, kot bi preiskovalec pomislil ob samem imenu ukaza (npr.: Na ukaz "Vklopi TV" se ugasne luč na hodniku). Za preiskovalca je tako poleg tega, katere vse informacije lahko najde v pametnem asistentu pomembno vedeti razliko med resničnimi in ponarejenimi aktivnostmi.

Drugi članek opisuje potek preiskave pametnega asistenta Alexa [7]. V članku je opisana tako analiza strojne, kot pro-

gramske opreme. Avtor se pri preiskavi osredotoči predvsem na uradno Android mobilno aplikacijo ter analizo imenika, ki jo bova tudi sama predstavila v nadaljevanju. Skozi preiskavo ugotovi, da je veliko datotek šifriranih. V okviru preiskave najde tudi kar nekaj šifrirnih ključev. Kot največja ovira pri preiskavi pa se izkaže iskanje uporabniškega gesla, ki je neuspešno.

V tretjem članku se avtorja ukvarjata s pametnim asistentom Google Home Mini [8]. Do njegovega nastanka se je namreč večina člankov ukvarjala predvsem z Amazon Echo - Alexo. Ukvarjata se predvsem s tem, kje asistent hrani posamezne podatke, pri čemer analizirata samo napravo, mobilno aplikacijo in omrežje.

V četrtem članku avtorji preučujejo pametnega asistenta Alexo [1]. Pri svojem delu razvijejo tudi aplikacijo CIFT, ki omogoča pridobivanje naravnih artefaktov iz Alexe in analizo lokalnih artefaktov. Ta aplikacija je uporabna tudi pri forenzični preiskavi.

3. ANALIZA PAMETNIH ASISTENTOV

Ekosistem pametnih asistentov zajema osrednjo napravo oziroma sistem z zvočnikom in mikrofonom, ki sprejema glasovne ukaze, ter naprave interneta stvari, ki jih asistent upravlja. Zajema tudi internetni oblak, na katerem so shranjene datoteke, lahko pa je povezan tudi z zunanjimi storitvami. Osrednja naprava je lahko tudi pametni telefon, kjer asistenta upravljamo z mobilno aplikacijo. S tem je asistent lahko povezan z večino internetnih storitev in na ta način shranjuje podatke, ki so morda pomembni za forenzično preiskavo, kot so na primer podatki o uporabniku in stanju datotek v oblaku. V članku ima mobilna naprava nameščen operacijski sistem Android. Ker je ekosistem sestavljen iz mobilnih naprav, asistentov, oblaka z zunanjimi storitvami in IoT napravami, moramo pri preiskavi analizirati vsa področja. Vizualni prikaz ekosistema pametnih asistentov je viden tudi na sliki 1. V nadaljevanju so navedene podrobnosti za analizo paketov asistentov in mobilnih aplikacij v omrežju, za analizo imenika in dekompilacije aplikacijskih paketov na mobilnih napravah z operacijskim sistemom Android ter analiza t. i. Chip-off-ov na asistentih.

3.1 Analiza paketov pametnih asistentov

Analiza paketov pri pametnih asistentih se osredotoča na pošiljanje paketov od asistenta do oblaka preko dostopne točke. Ker je na asistentih težko nastaviti certifikate ter s tem vzpostaviti posredniški strežnik, se analiza ukvarja s HTTP komunikacijo. Od analize paketov Android mobilnih aplikacij se razlikuje v tem, da pri mobilni aplikaciji lahko nastavimo posredniški strežnik ter s tem preiskujemo tudi HTTPS pakete, kar si bomo pogledali v naslednjem podpoglavju. Ostale analitične metode so pri obeh analizah enake.

Najprej nastavimo vse potrebno in zberemo podatke o uporabi paketov asistenta v realnem času. Ker se za te pakete uporablja protokol HTTP, lahko za njihov zajem uporabimo program Wireshark. Glede na uradno dokumentacijo pridobimo podatke za vse podprte funkcije pametnega asistenta. Pri tem analiziramo glavni komunikacijski strežnik, funkcije za strežnik in HTTP pakete.

Tipični podatki, ki jih dobimo kot rezultat analize so: informacije o uporabniškem računu, dostopni žeton pametnega asistenta za komunikacijo z oblakom, informacije o povezanih napravah, zgodovina ukazov za vsako povezano napravo, čas uporabe povezanih naprav, informacije o podatkih, shranjenih v oblaku ter notranjo strukturo oblaka.

3.2 Analiza paketov Android mobilnih aplikacij

Za razliko od analize paketov pametnih asistentov, se analiza paketov mobilnih naprav ukvarja s pošiljanjem paketov med mobilnimi aplikacijami in oblakom preko dostopne točke. V tem primeru je možno nastaviti posredniški strežnik s pristopom "man in the middle", kar omogoča analizo HTTPS paketov. Večinoma so podatki, ki so pomembni za preiskavo, v JSON formatu. Kot je opisano v prejšnjem razdelku, se analiza paketov Android mobilnih aplikacij razlikuje od analize paketov asistenta v tem, da pri mobilnih aplikacijah analiziramo pakete HTTPS, pri asistentu pa HTTP. Ostale analitične metode so enake.

Enako kot pri analizi paketov asistenta, začnemo z zbiranjem paketov v realnem času, pri čemer uporabimo posredniški strežnik. V članku je uporabljen program Fiddler Web Proxy, ki je varen dekoder HTTP protokola in uporablja pristop "man in the middle". Pri tem mora imeti mobilna naprava nameščene ustrezne certifikate. S tem postopkom pridobimo pomembne podatke za preiskavo, kot so uporabnikov profil, podatki o prijavi ter druge. Podobno kot pri analizi paketov asistenta, analiziramo glavni komunikacijski strežnik, funkcije ter v tem primeru HTTPS pakete.

Tipični rezultati analize paketov mobilnih aplikacij vsebujejo podobne podatke, kot rezultati analize paketov za asistenta.

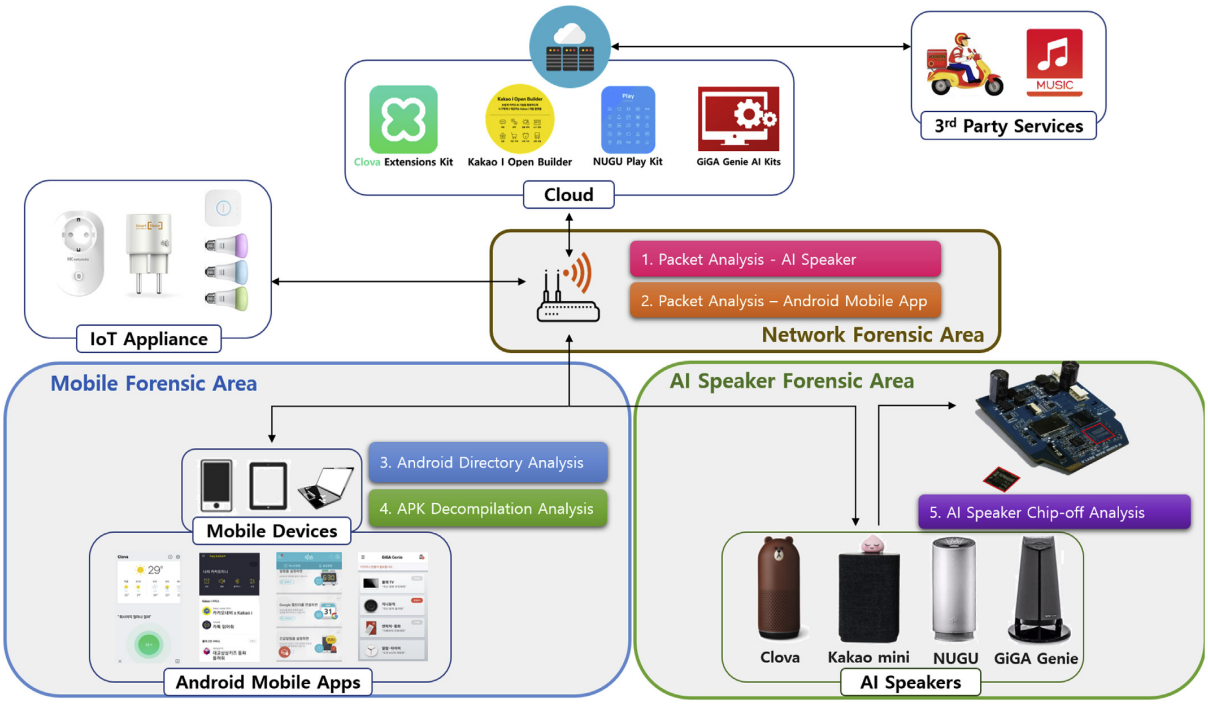
Na sliki 2 je prikazano okolje, s katerim zajamemo pakete pri pametnem asistentu NAVER Clova in povezanih napravah. Povezave med napravami so prikazane z modrimi črtami. Vsa komunikacija poteka preko dostopne točke. Rumene puščice prikazujejo smer komunikacije podatkov, kjer paketi med aplikacijo in oblakom tečejo obojesmerno. Komunikacija z dostopno točko poteka v realnem času.

3.3 Analiza datotečnega imenika mobilne naprave Android

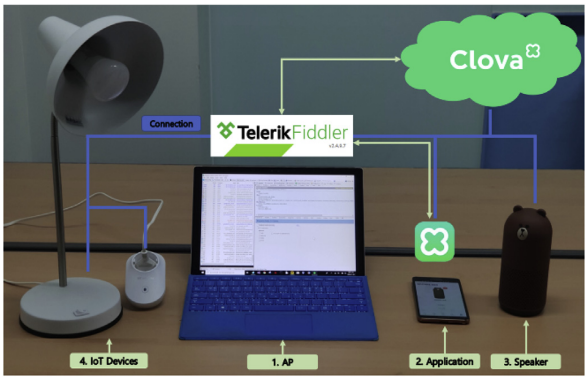
Podatki, ki so shranjeni v mobilni napravi, so zbrani in analizirani skupaj z mobilno aplikacijo. Ta komunicira z oblakom, poleg tega pa uporablja funkcije, kot so nastavitve asistenta in glasovni ukazi. Nekateri podatki so pri tem shranjeni v datotečnem imeniku aplikacije. Z analizo imenika pridobimo podatke o uporabniku, podatke o asistentu in informacije o glasovnih ukazih.

Analiza imenika pregleduje shranjene datoteke in informacije, specifične za povezane naprave. Poteka v treh fazah: preliminarno zbiranje podatkov, primarno zbiranje podatkov in analiza podatkov. V članku je predstavljen postopek analize za mobilno aplikacijo asistenta NAVER Clova.

Najprej je potrebno pridobiti pot do imenika aplikacije, da lahko identificiramo ime paketa aplikacije. Ta se običajno nahaja na lokaciji "/data/data", ki pa je zaščitena, zato za



Slika 1: Prikaz ekosistema pametnih asistentov



Slika 2: Testno okolje za ekosistem Clova

njen dostop potrebujemo administratorske (root) pravice. Te pridobimo z orodjem Odin, ki je Androidovo orodje za popravljanje in reševanje težav. Nato lahko uporabimo identifikacijo imena paketa, da dobimo pot do imenika, ki ga kasneje analiziramo. Identifikacija imena paketa uporablja orodje ADB. Pri tem pridobimo podatke s časovno oznako, ko je bila aplikacija nameščena, s časom po uporabi in s časom po odjavi iz aplikacije. S tem lahko opazujemo spremembe v podatkih skozi čas uporabe pametnega asistenta. Uporabimo vse funkcije asistenta, da lahko kasneje analiziramo vse možne podatke. Podatki, ki so ustvarjeni po odjavi uporabnika, se lahko uporabijo za ugotavljanje zgodovine preteklih ukazov.

Tehnike, ki so uporabljene pri analizi pridobljenih podatkov, torej zadnji fazi analize datotečnega imenika, so: analiza strukture imenika, analiza digitalnih podpisov datotek in analiza podrobnosti. Pri prvi je namen ugotoviti globino imenika, število in velikosti datotek v njem ter namen imenika. Pri drugi lahko določimo podpis datotek z uporabo hex urejevalnika. To pride v poštev predvsem pri datotekah, ki so iz različnih razlogov izgubile končnico. Tipično lahko tako določimo datoteke formata JPEG, PNG, SQLite in xml. S tem pridobimo originalne datoteke. Analiza podrobnosti zajema natančni pregled datotek, ki smo jih v predhodnih fazah pridobili s pripadajočimi programi za ogled vsakega formata datoteke. V datotekah najdemo pomembne podatke za preiskavo vključno z nekaterimi podatki uporabnika, ki je bil prijavljen v sistem asistenta, nekaterimi zvočnimi posnetki ukazov ter podatki o nastavitvah.

3.4 Analiza dekompilacije APK

Mobilna aplikacija komunicira z oblakom in za procesiranje glasovnih ukazov uporablja REST API. Z analizo podatkov lahko dobimo naslove ukazov REST API, ki so bili poslani na strežnik, ter druge informacije. Če se z aplikacijo povežemo na oblak, lahko analiziramo podatke, brez da bi spreminjali nastavitve ekosistema asistenta. Informacije, poslane v oblak natančno analiziramo z dekompilacijo aplikacijskih paketov (APK), in dobljene podatke uporabimo za validacijo naših rezultatov iz drugih analiz. Uporabljeni so statični in dinamični procesi analiz. Statična procesa, navedeni v članku, sta uporaba JaDX za preverjanje kode in apktool za proces imenovan baksmaling, ki je preliminarni proces za analizo in modifikacijo smali kode. Aplikacijski paket je za pridobivanje želenih rezultatov treba spreminjati,

kar lahko delamo z orodjem apktool. Po modifikaciji aplikacijski paket pripravimo za ponovno namestitev, po njej pa lahko izvajamo dinamično analizo aplikacijskega paketa. Za dinamično analizo uporabljamo Android Debug Bridge (ADB), s katerim analiziramo dnevniške zapise aplikacije. Z njimi ugotovimo, kakšen je operacijski mehanizem sistema asistenta.

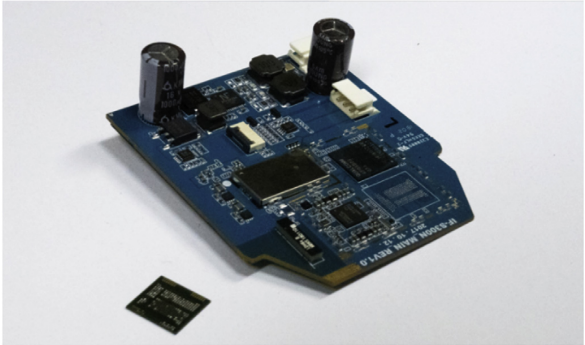
Tehnike za dekompilacijo so razdeljene v šest področji. Analiza klicne hierarhije nam da funkcijo uporabljenih klicev v IDE in jo dobimo z JaDX, ki vstavi dekompilirano izvorno kodo v Java IDE. Analiza iskanja v nizih je prav tako opravljena s pomočjo JaDX. Pri njej je potrebno spremeniti podpis sheme pridobivanja glede na tip artefakta. Rekonstrukcija metod in razredov v prvotno obliko je potrebna, da lahko razumemo njihovo delovanje, saj dekompilacija lahko prvotno vrne manj razumljivo kodo. Pri tem si pomagamo z že obstoječimi odprtokodnimi knjižnicami za Android. Analiza in modifikacija kode Dalvik, ki je uporabljena, ko nam JaDX ne da pričakovanih rezultatov v smislu binarnih napak ali nejasnosti, ali ko zaradi drugih faktorjev dekompilacija ni uspešna. V takih primerih se mora koda ročno ali s pomočjo IDE popraviti, kar je podobno, kot pri rekonstrukciji metod in razredov. Če analiza Java kode ne uspe, potem metode rekonstruiramo ročno vrstico po vrstico. Analiza dnevniških zapisov je opravljena po analizi in modifikaciji Dalvik kode. Dnevniški zapisi so pridobljeni v Android Logcat formatu.

3.5 Analiza odrezkov

Analiza odrezkov (angl. chip-off) pomeni analiza slike asistenta na internem spominskem čipu, ki se začne, ko asistenta fizično razstavimo in dobimo podatke s čipa. Primer spominskega čipa, ki ga analiziramo na sistemu NAVER Clova je prikazan na sliki 3. Ker je asistent v našem primeru zasnovan kot Android sistem, taka analiza vpliva tudi na dekompilacijo APK-jev, saj ima večina Androidovih naprav sistemske particije in so na njih prisotne APK. Na čipu najdemo identifikacijske podatke o uporabniku za asistentovo prepoznavanje, druge podatke o uporabniku ter zgodovino uporabe. Tehnike pri preiskavi so zbiranje podatkov, pridobivanje izrezka, prepoznavanje datotečnega sistema in strukture imenikov, pregled digitalnih podpisov datotek, analiza ključnih besed ter rekonstrukcija izbrisanih datotek.

Pri zbiranju podatkov uporabimo vse možne funkcije, ki jih omogoča pametni asistent, saj ne moremo vedeti, v katerem okolju je bil uporabljen, hkrati pa želimo pridobiti vse pomembne podatke. Ostale naštete tehnike so izvedene, ko dobimo slike bliskovnih (flash) pomnilnikov, ki nam jih priskrbi specializirano podjetje. Ko so slike pridobljene, prejšnje zbiranje podatkov ni več mogoče.

V procesu identifikacije datotečnega sistema najprej analiziramo metapodatke o slikah odrezkov, s tem pa ugotovimo, kateri datotečni sistem se uporablja in ali je v dobljeni sliki več particij. V članku so imeli testirani asistenti sistem, ki je temeljil na Androidu z EXT4 Linux datotečnim sistemom. Pri prepoznavanju strukture imenikov želimo dobiti več informacij o datotečnem sistemu in particijah. Namen je določitev globine imenikov ter velikost in količino datotek. V pregledu digitalnih podpisov datotek določimo formate datotek, kjer so med najpomembnejšimi formati tisti, ki predstavljajo zvočne datoteke. Te morda vsebujejo uporabnikov glas in odziv asistenta. Tudi nastavitvene (konfiguracijske) datoteke in datoteke podatkovnih baz imajo pomembno vlogo. Uporablja se tudi iskanje ključnih besed, kot so "voice", "log", "version", "release", "time" in "journal". Rekonstrukcija izbrisanih besed je opravljena predvsem zaradi tega, ker se lahko ob tem, ko pridobivamo izrezke in ugasnemo sistem, izbrišejo pomembne datoteke. Ker so uporabljeni EXT4 datotečni sistemi, z rekonstrukcijo ni težav.



Slika 3: Bliskovni spominski čip po razstavitvi sistema NAVER Clova

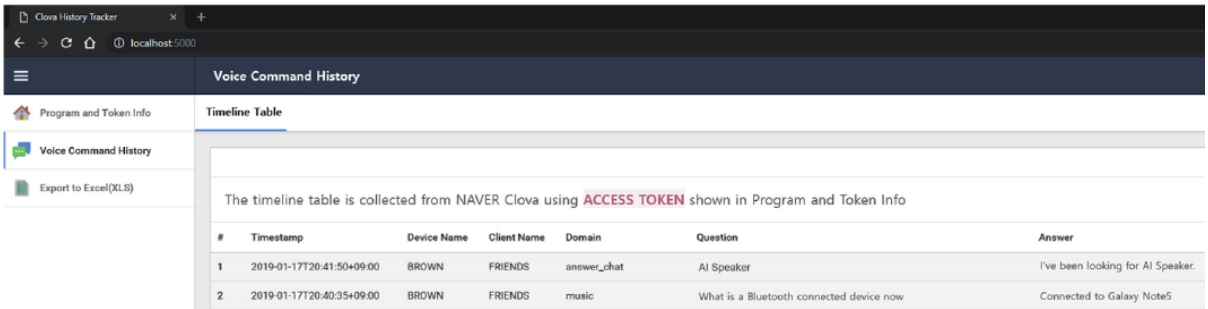
4. FORENZIČNO ORODJE ZA PAMETNEGA ASISTENTA NAVER CLOVA

V tem poglavju je predstavljeno forenzično orodje, ki so ga avtorji razvili z namenom, da bi vključili metode analiz, ki so naštete v prejšnjem poglavju v eno orodje, ki omogoča pridobitev rezultatov večine metod. Orodje je optimizirano za delovanje s pametnim asistentom NAVER Clova in med drugim nudi informacije o programu in žetonu ter zgodovino ukazov, pridobljeno s pomočjo podatkov uporabnika, ki so bili zajeti v analizi paketov (Slika 4). Podatke o zgodovini ukazov lahko pridobimo direktno iz oblaka, brez uporabe mobilne aplikacije. Uporabniški vmesnik omogoča pregled zgodovine ukazov, ki prikazuje glasovne ali tekstovne zahteve in odgovore. Vidimo lahko največ 100 zapisov. Prikazan je tudi datum zapisa. Žeton aplikacije lahko pridobimo s posredniškim strežnikom, kar je učinkovitejše od porabe pomnilnika na napravi Android, saj bi lahko izgubili integriteto podatkov ob pridobivanju pravic administratorja. Žeton je lahko uporabljen v napadu, kjer storilec žeton ponovno uporabi in so mu s tem dodeljene vse pravice brez preverjanja identitete. Z istega razloga lahko pridobimo zgodovino ukazov tudi brez administratorskih pravic, če smo pridobili žeton, ki je pripadal prijavljenemu uporabniku. Orodje podpira tudi izvoz podatkov v Excel.

Poleg zgodovine ukazov torej orodje prikazuje še časovno oznako, ime naprave. ime klienta, domeno, zahtevo in odgovor. Ker je NAVER Clova korejski model asistenta, je tudi orodje namenjeno korejskem trgu in je za potrebe članka prevedeno v angleščino.

5. RAZPRAVA

V tem poglavju je predstavljeno najino osebno mišljenje in videnje področja digitalne preiskave pametnih asistentov.



Slika 4: Orodje za forenzično preiskavo asistenta NAVER Clova

Glede na večanje števila pametnih asistentov ter količine podatkov, ki jih le ti vsebujejo, so že in bodo zagotovo pomemben vir informacij pri forenzičnih preiskavah. Po svetu je bilo že kar nekaj primerov, kjer so ravno s pomočjo analize pametnih asistentov rešili primer, a se pogosto tudi zgodi, da kljub zahtevku sodišča za pridobitev podatkov iz pametnega asistenta, jih proizvajalec noče posredovati [6, 4, 3]. Na tem področju bo v bližnji prihodnosti zagotovo potrebno natančneje določiti pogoje za dostop do podatkov ter s tem zagotoviti enakost v vseh kazenskih postopkih. A s pripravo zakonodaje ne smemo prehitevati, saj te naprave vsebujejo mikrofon, ki običajno posluša 24 ur na dan ter tako ve marsikaj, česar z zunanjim svetom ne želimo deliti. Težava pri teh naprav je namreč, da če želijo sprejeti ukaz, morajo imeti vedno vklopljen mikrofon. Če morda na računalniku lahko fizično izklopimo mikrofon in kamero, ko ju ne potrebujemo, bi le to pri pametnih asistentih zmanjšalo njihovo uporabnost, saj bi v tem času lahko velikokrat že sami izvedli aktivnost (npr: "Vklopi TV").

Preiskovalne metode, ki so razložene v originalnem članku ter sva jih predstavila tudi sama, se nama zdijo dobre. Metode namreč zajamejo vsa področja, ki jih pokrivajo pametni asistenti od oblačnih storitev, IoT, mobilnih aplikacij do samih čipov v pametnem asistentu. Kljub temu, da so metode napisane za štiri korejske predstavnike, so uporabne tudi pri ostalih, pri čemer so med njimi le majhne razlike. Enakega mnenja so tudi pri svetovni organizaciji Interpol, kjer so te metode priporočili tudi na lastnem simpoziju v okviru primerne literature za digitalne dokaze v letih 2016 - 2019 [9].

Ob vsem naštetem pa ne smemo pozabiti, da so lahko tudi pametni asistenti izrabljeni v napačne namene. Kot že omenjeno, si lahko osumljenec z njegovo pomočjo zagotovi lažni alibi ali pa ustvari lažne dokaze, zato bo v prihodnosti zagotovo potrebno posvetiti pozornost tudi prepoznavanju neresničnih podatkov.

6. ZAKLJUČEK

Pametni asistenti so vedno bolj prisotni v našem vsakdanjem življenju ter so tako vir veliko informacij. Ker gre za kompleksne naprave, ki združujejo tako internet stvari, kot oblačne storitve, obstaja za njihovo preiskovanje pet metod, in sicer: analiza paketov pametnih asistentov, analiza paketov Android mobilnih aplikacij, analiza datotečnega imenika mobilne naprave Android, analiza dekompilacije APK in analiza odrezkov. V vsaki izmed njih dobimo določene podatkov, zato je pomembno, katero metodo izberemo, da pri tem dobimo željene podatke. Nekatere podatke lahko pridobimo tudi v različnih metodah preiskovanja. To nam omogoča, da lahko z večjo verjetnostjo trdimo, da so ti podatki resnični. V tem članku smo predstavili podrobnosti pri izvedbi vsake od zgoraj naštetih metod, ter orodje, ki omogoča izvedbo teh metod na korejskem pametnem sistemu NAVER Clova.

Možna nadaljnja dela za tak forenzični pristop bi bila raziskava seje in enkripcijskih algoritmov pri dekompilaciji APK ter razširitev in nadgradnja predstavljenega orodja za NAVER Clova. Zanimivo bi bilo tudi preveriti, kako metode, ki so vezane na mobilne naprave z operacijskim sistemom Android, izvedemo na tistih z IOS. Na to temo namreč nisva zasledila nobenega kakovostnega članka.

7. LITERATURA

[1] H. Chunga, J. Parka, and S. Leea. Digital forensic approaches for amazon alexa ecosystem, 2017.

[2] L. M. F. Dailymail.com. Man, 28, arrested for allegedly beating girlfriend after an amazon alexa device calls 911, Jul 2019. Dostopano: 4.6.2020.

[3] K. Epstein. Police think amazon's alexa may have information on a fatal stabbing case, Nov 2019. Dostopano: 5.6.2020.

[4] M. Flynn. Police think alexa may have witnessed a new hampshire double homicide. now they want amazon to turn her over., Nov 2018. Dostopano: 5.6.2020.

[5] W. Jo, Y. Shin, H. Kim, D. Yoo, D. Kim, C. Kang, J. Jin, J. Oh, B. Na, and T. Shon. Digital forensic practices and methodologies for ai speaker ecosystems. *Digital Investigation*, 29:S80 – S93, 2019.

[6] D. Moser. The role of home devices in police investigations, Aug 2019. Dostopano: 5.6.2020.

[7] T. Olufohunsi. Forensic investigation of artificial intelligence virtual/personal assistant: Amazon alexa as a case study. 01 2020.

[8] M. Park and J. I. James. Preliminary study of a google home mini. *Journal of Digital Forensics*, 12(1), 06 2018.

[9] P. Reedy. Interpol review of digital evidence 2016 - 2019. *Forensic Science International: Synergy*, 2020.

[10] A. Sulleyman. Man who threatened to murder

girlfriend arrested after smart gadget alerted police, Jul 2017. Dostopano: 4.6.2020.

[11] I. Yıldırım, E. Bostanci, and M. Guzel. Forensic analysis with anti-forensic case studies on amazon alexa and google assistant build-in smart home speakers. pages 1–3, 09 2019.

Nedovoljeni digitalni dokazi in zaščita zasebnosti v sistemih verige blokov

Andrej Burja
Fakulteta za računalništvo in informatiko
Univerza v Ljubljani
ab9501@student.uni-lj.si

Lovro Vražič
Fakulteta za računalništvo in informatiko
Univerza v Ljubljani
lv0811@student.uni-lj.si

POVZETEK
Članek, ki sva ga prebrala se osredotoča na velikokrat pozabljen aspekt rokovanja z dokazi, to je ko sodišče zavrže dokaze. Več razlogov je zaradi katerih so dokazi lahko zavrženi. Lahko so zavrženi že med preiskavo, pred sodiščem, ali pa jih opusti tožilec. Ker v generični implementaciji verige blokov kasnejše spremembe niso mogoče, je v članku opisana rešitev katera uporablja dve verigi blokov za vodenje verige skrbništva. S tem je omogočeno razveljevanje transakcij, ki se navezujejo na dokaze.

Ključne besede
Veriga skrbništva, dokazi, veriga blokov, transakcije

1. UVOD
Različne države imajo različne pravne sisteme. V nadaljevanju je opisan primer za katerega lahko rečemo, da bi veljal v večini držav. Bojan je osumljen posedovanja posnetkov spolne zlorabe otrok. Izdan je sodni nalog kateri je podlaga za preiskavo njegove hiše. Med preiskavo je najden trdi disk, ki je po preiskovalnih postopkih registriran, in lahko se prične voditi skrbniška veriga. Ker je sodobna policija že opremljena z novimi tehnologijami je trdi disk registriran tudi v evidenco, ki deluje na podlagi sistema verige blokov.

Digitalni forenziki nato pregledajo disk in na njem najdejo dokaze, ki kažejo, da je tudi Anja povezana s spolno zlorabo otrok. Na podlagi tega je izdan sodni nalog za preiskavo tudi njene hiše. Med preiskavo najdejo USB pomnilnik, na katerem je več posnetkov spolne zlorabe otrok. Kot zahtevajo postopki, je tudi ta pomnilnik dodan v evidenco. Kasneje v postopku odvetnik obtoženega (Bojana) izpodbija legalnost pridobivanja dokazov v prvi policijski preiskavi. Sodišče ugodi, in dokazi so zavrženi. Ker je druga preiskava rezultat prve, so tudi dokazi iz druge preiskave zavrženi.

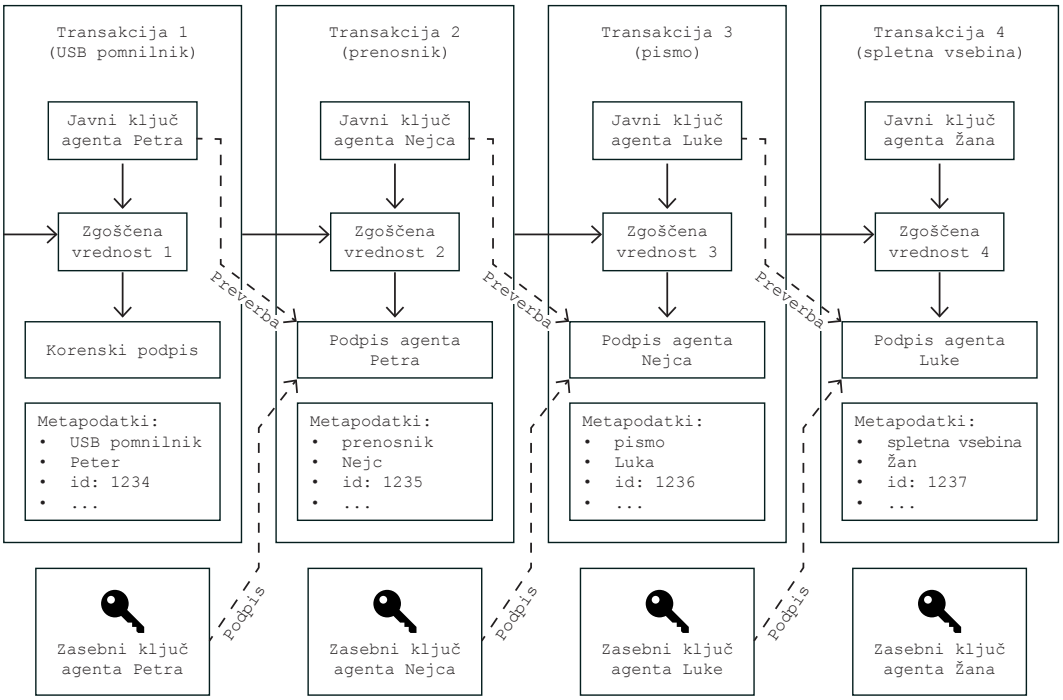
1.1 VERIGA BLOKOV
Veriga blokov ali angl. *blockchain*, so med seboj povezani bloki, v katerih je seznam zapisov (transakcije). Vsak blok vsebuje podatke, hash bloka in hash bloka pred njim. Hash, slovensko zgoščena vrednost, je podpis podatkov (*prstni odtis*), kjer zgoščevalna funkcija podatke poljubne velikosti preslika v besedilo konstantne dolžine. V verigi blokov so vse zgoščene vrednosti blokov unikatne. Izračunajo se, ko se blok ustvari. Če v bloku podatke spremenimo, se spremeni tudi zgoščena vrednost bloka. Kar daje verigi blokov varnost, je to, da vsi (z izjemo prvega - *Genesis block*) bloki vsebujejo zgoščeno vrednost prejšnjega bloka, s čimer dobimo verigo med seboj povezanih blokov.

Podatki v bloku so lahko različni, odvisni pa so od tipa verige blokov. Če vzamemo npr. Bitcoin, potem ti podatki predstavljajo transakcije, ki povedo iz katerega naslova je znesek prišel (*from*), na kateri naslov je namenjen (*to*) in količino v Bitcoinih (*amount*). V našem primeru je podatek eden v katerem so *metapodatki* dokaza.

V primeru, da nekdo v nekem bloku spremeni podatke, se zgoščena vrednost bloka spremeni, kar pomeni, da so vsi bloki po tem bloku neveljavni, saj se z zgoščeno vrednostjo ne povezujejo več med seboj. Če na novo izračunamo vse zgoščene vrednosti po tem bloku, je veriga spet veljavna. Da ne pride do takšne zlorabe, nekatere verige blokov uporabljajo *proof-of-work* (dokaz o delu) med njimi je tudi Bitcoin [5].

1.2 POSTOPEK V VERIGI BLOKOV
Poglejmo si sedaj isti primer, ko je za implementacijo skrbniške verige uporabljen sistem veriženja blokov. Metapodatki dokazov iz preiskav so shranjeni v verigo blokov, ta pa nima mehanizma, ki bi omogočal odstranjevanje dokazov iz verige blokov, ker je ta namenoma zasnovana tako, da kasnejše spremembe niso mogoče. Zaradi te pomanjkljivosti je potrebna drugačna rešitev, ki bo omogočala kasnejše spremembe in s tem odstranjevanje, zavračanje ali razveljavljenje transakcij.

Predlagani sta dve rešitvi. Prva je, da se izbriše celotna veriga blokov in se potem naredi novo, v kateri so izpuščeni zavrženi dokazi. V praksi bi to pomenilo, da se na novo generira veljavna veriga blokov v katerih ni zavrženih dokazov. To pomeni, da se generiranje začne pri korenskem bloku in izda vse naslednje transakcije, razen transakcij povezanih z zavrženimi dokazi. Čeprav je sistem tehnično izvedljiv, bi ta zahteval veliko zahtevnost pri preverjanju transakcij in blokov, vključujoč glasovalni algoritmi in vodenje vseh blokovnih referenc. V tem primeru je to nemogoče avtomatsko generirat, če program nima vseh zasebnih ključev sodnikov in preiskovalnih agentov. Če pa jih ima, je varnost takega sistema vprašljiva. Opazimo tudi, da bi tak sistem zahteval veliko računsko kompleksnost.



Slika 1: Primer verige blokov InventoryTX.

Druga rešitev je, da se uporabi veriga blokov, ki vsebuje razveljavitvene transakcije, katerih namen je kazanje na transakcije, ki niso več pravno veljavne in ne morejo biti več uporabljene kot referenca za dokaz. To pomeni, da ima veriga blokov dve kategoriji transakcij: (1) transakcija za registracijo dokaza in (2) transakcija za razveljavitev dokazov.

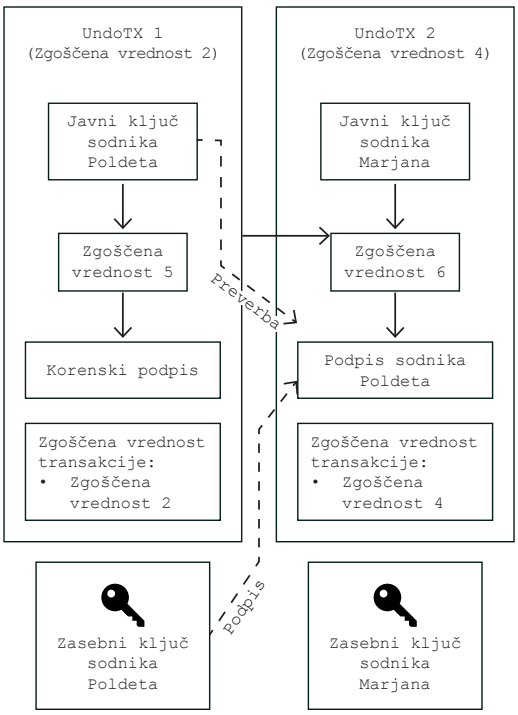
Ta način je že dolgo in široko uporabljen pri sistemih za upravljanje s podatkovnimi zbirkami (Database management system, DBMS). Vendar kljub uporabnosti v takih sistemih, ta prinese težave pri sistemih verige blokov.

Največjo skrb povzroča verifikacija potrjevanja transakcije. Da lahko uporabnik preveri pravilnost transakcije, mora ta preveriti ponavadi (celotno verigo) od začetka verige, da ta ni bila prekinjena od neke točke v času. Ta preverba potrdi, da je bila transakcija pravilno dodana v sistem in da je preverjena po pravilih.

Ta preverba pa kljub temu ne potrdi, da je transakcija pravilna iz pravnega vidika. Dokaz povezan z transakcijo je bil lahko završen kasneje. Potemtakem se mora preverba nadaljevati dokler ne najde: (1) razveljavitveno transakcijo, potem transakcija ni pravno veljavna ali (2) postopek iskanja doseže konec verige, kar pomeni, da je transakcija pravno veljavna. Opazimo lahko, da je računska zahtevnost postopka iskanj veliko višja, kot pri običajni preverbi transakcije v verigah blokov[6].

2. PREDLAGANA REŠITEV

Obe rešitvi sta nezadovoljivi, zato se predlaga rešitev v kateri dodatna plast programske rešitve, AccessTX, omogoča



Slika 2: Primer verige blokov InvalidatedTX.

dostop do verige blokov. Poimenujmo verigo skrbništva v verigi blokov InventoryTX. Dodatna plast AccessTX, ki omogoča dostop do InventoryTX bo najprej preverila, če je transakcija pravno zavržena in ali je pravilna. Po tej preverbi bo transakcija preverjena še po normalni poti verige blokov InventoryTX.

Nadzor dostopa bo prav tako za preverjanje uporabil verigo blokov imenovano InvalidatedTX. Vsaka transakcija v tej verigi blokov se preko zgoščene vrednosti transakcije navezuje na nedovoljene dokaze. Zaželeno je, da je vsaka transakcija v InvalidatedTX podpisana s strani pristojne entitete, ki je izdala nalog za razveljavitev nedovoljenih dokazov.

Potrditev vsake razveljavitvene transakcije je procesirana, kot pri standardni verigi blokov, od korenskega bloka InvalidatedTX. Sama narava razveljavitvenih transakcij jo ločuje od standardne verige blokov.

Primer je verjetno najboljši za prikaz delovanja predlagane rešitve. Recimo da policija preišče hišo Ane. Ta ženska je osumljena skrivanja moškega, ki ga išče policija. Trije dokazni predmeti so najdeni v njeni hiši:

- Agent Peter je našel USB pomnilnik na katerem so osebni podatki iskanega moškega;
- Agent Nejc je našel prenosnik s pornografsko vsebino in povezavo do spletnega strežnika;
- Agent Luka je našel ljubezensko pismo poslano s strani osumljenca gospe Ani.

Kasneje je agent Žan ob pregledu spletne strani našel vsebino, ki opisuje postopke za izdelavo prepovedanih drog. Veriga blokov InventoryTX, na sliki 1, je primer izgradnje verige blokov na podlagi teh dokazov. Opazimo lahko, da izgleda kot generična veriga blokov.

Recimo, da se obtoženi zagovarja, da pornografsko gradivo in recepti za droge niso predmet preiskave, in morajo zato biti zavrženi. Sodišče se s tem strinja, in sodnik Polde in Marjan posodobita verigo InvalidateTX, prikazano na sliki 2. Ko bo oseba, ki ima dostop do sistema AccessTX, posredovala poizvedbo za pridobitev shranjenega dokaza, bo sistem najprej pogledal verigo InvalidatedTX. V te verigi bo preveril, če je transakcija pravno pravilna. Poizvedba lahko vrne tri tipe odgovorov:

- Če zgoščena vrednost transakcije ni prisotna v InvalidatedTX, je pa prisotna v InventoryTX, potem bo sistem uporabniku vrnil podatke o dokazu. Te podatki ponavadi vsebujejo mesto hrambe na kateri se nahaja dokazno gradivo. Podatki lahko vsebujejo tudi samo metapodatke dokaza.
- Če zgoščena vrednost transakcije ni prisotna v InvalidatedTX in prav tako ne v InventoryTX, potem bo sistem uporabniku vrnil izjemo: `"Transakcija ni najdena."`
- Če zgoščena vrednost transakcije je prisotna v InvalidatedTX, pa bo sistem uporabniku vrnil izjemo: `"Transakcija razveljavljena s sodno odločbo xxx."`[6]

3. DRUGA DELA

Na internetu se je v zadnjih letih pojavilo kar nekaj člankov, ki se navezujejo na verigo skrbništva katere delujejo na podlagi verige blokov. V nekaterih člankih celo opisujejo shranjevanje dokazov, vendar v ločeni podatkovni bazi, iz katere se dokaze lahko izbriše, saj to pri navadni verigi blokov ne bi bilo mogoče.[2] Večina opisanih rešitev, ki jih najdemo na internetu za delovanje uporabljajo predelano zasebno verigo blokov *Ethereum*. Ta nam ponuja platformo, na kateri lahko uporabniki razvijejo svoje pametne pogodbe (decentralizirane aplikacije). Preko teh pogodb so avtorjih drugih člankov izdelali rešitev, ki omogoča izdelavo enostavne verige skrbništva.[2, 4]

Primeri drugih rešitv so za implementacijo verige blokov vzeli bolj univerzalno rešitev, kot je *Hyperledger Composer*. Gre za odprtokodno razvojno okolje za izdelavo verige blokov. Ta nam omogoča, da lahko na enostaven način napišemo kodo, ki prilagodi verigo blokov za točno določen namen, kot je v tem primeru veriga skrbništva [3].

Nikjer ni zaslediti, da bi se za implementacijo verige skrbništva uporabljali dve verigi blokov, kot predlaga avtor. V vseh ostalih pregledanih člankih se uporabi le ena veriga blokov.

4. DRUGA PODROČJA

Ob pregledu področja uporabe verige blokov za vodenje različnih verig lastništva najdemo kar nekaj primerov. Na spletu se vse več pojavlja uporaba verige blokov za vodenje različnih verig, kot na primer vodenje živilske verige [7]. V tem primeru se na ta način sledi transportu, pogojem med transportom, verigi lastništva in se tako zagotovi potrošnikom pravilno rokovanje živil in s tem tudi podatke o rokovanju specifičnega proizvoda.

Vodenje verige lastništva je prisotno tudi v industriji dragih kamnov, lastništvu strelnega orožja, farmacevtski industriji in tako naprej [1]. Še posebej je lahko tehnologija blokov zanimiva za področje umetništva, ker omogoča vodenje lastnikov vse od prvotnega lastnika, ki je tudi avtor dela [8].

5. ZAKLJUČEK

Prednost tega sistema je, da je zelo sistemsko nezahteven. V primeru, da ni razveljavljenih transakcij v InvalidatedTX je časovna ocena iskanja O(1). V primeru, da je bilo iskano dokazno gradivo zavrženo s strani sodišča, pa je časovna ocena iskanja O(m), kjer m predstavlja število vseh transakcij v InvalidatedTX.

Ta rešitev za zavračanje nedovoljenih dokazov ne odpravlja dejstva, da so bili dokazi nekoč del dokaznega gradiva. Preprečila pa bi njegovo uporabo. Ta algoritem bi lahko pomagal pri sprejemanju tehnologije verige blokov kateri zagotavlja večjo prožnost pri upravljanju z dokazi.

Rešitev deluje z večino implementacij verig blokov, ker ne spreminja strukture verige. Prav tako so dokazi ločeni od verige blokov, ta vsebuje le metapodatke dokazov in verigo skrbništva.

Zdi se nama, da se je avtor članka kar precej osredotočil na izboljšavo časovne ocene iskanja. Res je, da je izbran članek le del članka oz. le obsežnejši pregled prihajajočega članka, vendar pa sva pogrešala nekaj podatkov, kot so: podatek kako se potrdijo novi bloki, podrobnejši opis delovanja AccessTX itn. V tem članku nisva zasledila nobenega tehničnega podatka, ki bi nama lahko povedal kaj več o tipu verige blokov ali pa kaj več o komponenti AccessTX.

Prav tako avtor v svoji rešitvi ne predlaga kako se rešuje problem avtentikacije. Najin predlog je, da se za dostop uporabijo certifikati, katere bi uporabnik predložil ob dostopu do aplikacije. Dodatno predlagava, da se v dodatni programski plasti AccessTX za preverjanje avtorizacije uporabi podatkovno bazo, v kateri so podatki o dovoljenjih posameznega uporabnika na podlagi njegove vloge. Na primer sodniki lahko dodajajo razveljavitvene transakcije, med tem, ko preiskovalci smejo dodajati le transakcije za vnos novega dokaza.

6. REFERENCE

[1] Blockchain for 2018 and Beyond: A (growing) list of blockchain use cases, Kevin Doubleday . `https://medium.com/fluree/blockchain-for-2018-and-beyond-a-growing\-list-of-blockchain-use-cases-37db7c19fb99`. [Dostopano: 01. 06. 2020].

[2] B-CoC: A Blockchain-Based Chain of Custody for Evidences Management in Digital Forensics, Silvia Bonomi & Marco Casini & Claudio Ciccotelli . `https://drops.dagstuhl.de/opus/volltexte/2019/11402/pdf/OASIcs-Tokenomics-2019-12.pdf`. [Dostopano: 01. 06. 2020].

[3] Forensic-chain: Blockchain based digital forensics chain of custody with PoC in Hyperledger Composer, Auqib Hamid Lone & Roohie Naaz Mir . `https://www.sciencedirect.com/science/article/abs/pii/S174228761830344X?via%3Dihub`. [Dostopano: 01. 06. 2020].

[4] Forensic-chain: Ethereum blockchain based digital forensics chain of custody, Auqib Hamid Lone . `https://www.researchgate.net/publication/321746762_Forensic-chain_Ethereum_blockchain_based_digital_forensics_chain_of_custody`. [Dostopano: 01. 06. 2020].

[5] Aplikacija za trgovanje na kriptoborzah, Andrej Burja. `http://eprints.fri.uni-lj.si/4437/1/63140024-ANDREJ_BURJA-Aplikacija_za_trgovanje_na_kriptoborzah.pdf`. [Dostopano: 02. 05. 2020].

[6] D. Billard. Tainted digital evidence and privacy protection in blockchain-based systems, forensic science international: Digital investigation. *Elsevier*, X(X):2, January 2020.

[7] Blockchain, Provenance, Traceability & Chain of Custody, John G. Keogh .

[8] Blockchain, Chain of Custody and Trace Elements: An Overview of Tracking and Traceability Opportunities in the Gem Industry, Laurent E Cartier & Saleem Ali & Michael S. Krzemnicki . `https://www.researchgate.net/publication/327555908_Blockchain_Chain_of_Custody_and_Trace_Elements_An_Overview_of_Tracking_and_Traceability_Opportunities_in_the_Gem_Industry`. [Dostopano: 01. 06. 2020].

Detecting Cyberbullying "Hotspots" on Twitter: A Predictive Analytics Approach

Jaka Stavanja
Fakulteta za računalništvo in informatiko
Večna pot 113
Ljubljana, Slovenija
js8927@student.uni-lj.si

Andraž Povše
Fakulteta za računalništvo in informatiko
Večna pot 113
Ljubljana, Slovenija
ap6932@student.uni-lj.si

Blaž Rupnik
Fakulteta za računalništvo in informatiko
Večna pot 113
Ljubljana, Slovenija
br8839@student.uni-lj.si

ABSTRACT
In this article, we describe the work done in the field of detecting cyberbullying from online texts from the article by Ho et al [1], [2]. Furthermore we introduce methods and available software by explaining what cyberbullying hotspots are and how we can computationally find them. We also describe other approaches and work in the field of detecting cyberbullying. Lastly, we implement our own classifiers and try to computationally infer whether tweets are cyberbullying orientated ourselves and discuss the results and possible future applications of these methods.

Categories and Subject Descriptors
H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms
Cyberbullying, classification

Keywords
cyberbullying, field review, predictive analysis, logistic regression, text mining, social media, Formspring, Twitter

1. INTRODUCTION
With the widespread usage of information technology in social interaction, physical bullying has been partly replaced with cyberbullying. Cyberbullying is charged language towards people, found online. We will describe the way we think of cyberbullying in this article more in depth later in the article, along with some examples. These kind of attacks, can inflict mental and sometimes result in physical damage to their victims. Since the victims often already suffer from low self esteem, such attacks can push them into a vicious downward spiral, sometimes resulting in extreme escalations such as attempted suicide [3]. Furthermore, the anonimity of platforms like Twitter enables bullies to hide their true identity from their victim while simple registration allows for creation of disposable accounts.

Study [4] on the rise of cyberbullying was performed in 2015. In examination of 28,104 adolescents attending 58 high schools in United States, approximately 23 percent of the youth reported being victims of any form of bullying and approximately 25 percent of these victims mentioned cyberbullying as one of the forms.

The cyberbullying research center posted data about how much middle and high schools students in the United States experienced cyberbullying from years 2007 to 2019 [5] and stated that the cyberbullying cases are on the rise. In 2007, approximately 20% of students experienced it and in 2019 that percentage has risen to around 37%.

Hotspots are areas that have a greater than average number of events or in our case are an area with more crime than usual [6]. For example, if a Twitter account seems to post a lot of offensive and bullying tweets, that could be some kind of a hotspot for cyberbullying on Twitter. In this case, we would like to find the cyberbullying hotspots on Twitter by identifying the probability that individual user's messages are cyberbullying first and then trying to identify users with many of them. Then we might raise some flag for such users.

The articles we're primarily reviewing [1], [2] and our research will be more focused towards the part of finding the probabilities that individual tweets are cyberbullying, since the remaining part of identifying which users post many tweets like these is a lot easier to compute and is relatively straightforward.

Thus, we talk about the computational approaches on how we can detect charged language from text or in other words, how we detect how much tendency a person has to be a cyberbully from his messages and posts using a regression approach.

2. FIELD OVERVIEW
In the field of detecting cyberbullying online, there seem to be a few approaches that the researchers use to extract the possibilities of cyberbullying from text.

Researchers were trying different text mining techniques on different scenarios. Starting with vandalism, spam, internet

abuse and cyberterrorism in the late 2010's and later starting with explorations on cyberbullying in 2011 (Dinakar, Reichar, Lieberman [7]). That particular research was conducted on Youtube comment data, where it was manually labeled and studied with different binary and multiclass classifiers. The study concluded that binary classifiers performed better of the two. However, that research didn't include a lot of data which would include context of the discussion which some of the later work also includes.

Studies by Yin et al. [8] trained a support vector machine to be able to detect harrasment and also used contextual data. However, they did not include the characteristics of the user posting these texts and solely used the context from each of their message separately.

Then, the users' information was added to the context by Dadvar and De Jong [9] and had a better result.

Reynolds et. al [10] also used user context and used a supervised learning approach, but on data from the web portal Formspring.me. They concluded that the density of bad words in a post had a great impact on the accuracy in detecting whether the messages were cyberbullying.

The main work in the field was done on cyberbullying, however our article was more revolving around tweets. The work that it is focusing on is the Masters Thesis from Kasture [11] where he has developed a predictive model to detect online cyberbullying.

Other studies, conducted on text classification for cyberbullying are also described shortly in that work and are perhaps the most important in this field. In Kasture's work, psychometrics were also introduced to the studies. Psychometrics is thought of as a field of study which quantifies characteristic differences of humans [11]. To compute the psychometric features, he used a paid tool called LIWC.

The articles we are primarily reviewing [1], [2] are based on Kasture's dataset which they used for creating a baseline model. It uses 10 categories amongst 90 features generated by the LIWC tool. They used a logistic regression model and compared results with Kasture's.

3. CLASSIFYING TWEETS AS CYBERBULLYING

In this chapter, we present the logical framework and the idea behind the research, conducted in the article, we are reviewing [1]. In the next chapter, we use the same concepts to implement our own classifiers, that we later compare to the one from the original article.

3.1 Cyberbullying

In our case cyberbullying is thought as charged language towards someone in an online message, more specifically, in a tweet. That doesn't mean that any bad word is directed towards someone or is there to insult as it could only be a reference to genitals. There are many proposed definitions for cyberbullying and they point out repetition as some sort of critical element, that seperates bullying from jokes or simply teasing.

3.2 Classification framework

To classify tweets we need to construct a framework, seen in Figure 1.

To try and classify whether a tweet contains cyberbullying or not, we will have to take a dataset of tweets, that would potentially include charged language. In our primarily reviewed article, the researchers collected the data using a script, that was fetching tweets from the Twitter API for a period of a couple of months, but that data is not publicly available. The work from Kasture also used a similar dataset of tweets whereas this dataset is publicly available and linked inside of the thesis.

However, the framework for cyberbullying classification will work on any dataset. The only thing we need to ensure is that we filter the initial tweets well. The framework will take in a dataset of tweets that are potentially cyberbullying. We treat tweets including bad words (more particularly words: "die", "faggot", "fat", "fuck", "kill", "loser", "shit", "slut", "suck", "whore", "bitch", "cunt", "dick", "pussy") as candidates as stated in the article by Ho et al [1], [2]. We go through all the tweets and only keep those with the bad words in them.

Then, we compute feature vectors from the dataset. Since Kasture showed that the data processed inside LIWC produced good results, Ho et al. [1], [2] also computed the feature vectors using that dataset. The features in vectors are psychometric properties, computed for the tweets such as the amount of negative emotion, sexual content, death related words, etc. This form of data (feature vectors and cyberbullying class) will also be used to train our experimental implementation.

We should also mention, that the preprocessing of the text is all done inside LIWC, where the tool computes sentimental and contextual values from the text itself. We don't manually fix typos or take only stems of words, instead all of that is done through the tool. By looking at the examples from the dataset, that we will present later, we can also see that some of the symbols and characters were manually removed by the creator of the dataset.

3.3 Classification

The next piece of the framework is the actual classifier that will decide whether a tweet contains cyberbullying or not. For that, the researchers propose a logistic regression model, that will output 0 if a tweet is not cyberbullying and 1 otherwise. A logistic regression classifier in general takes multidimensional inputs such as feature vectors and tries to fit a model to the training data. Then, when we get a new input, we can put the input vector in the same space and take a look at what value the fitted model predicts for that input. Our logistic regression classifier will in the end output whether our new tweet (transformed to fit in our vector space) contains cyberbullying or not.

Using that framework we could then also compute the amount of charged tweets for each person that one follows and give out preventative information of people's cyberbully tendencies to the user which would help identify which accounts are bad to follow.

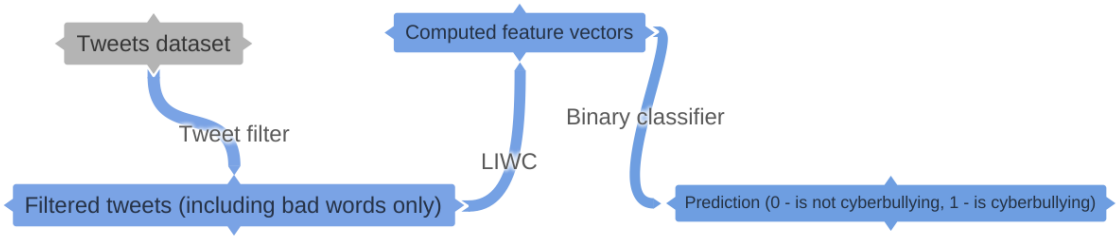


Figure 1: Study framework inspired by the article from Ho et al [1], [2].

We can also use any other binary classifier, some of which we will also test ourselves.

4. EXPERIMENTAL IMPLEMENTATION

To test how the approach, described in the article by Ho et al. [1], [2] works, we implement our own logistic regression model (since the original authors used the same approach) and test it out on the dataset, which was used to train the classifier in the original article. Along with that, we try some other models, easily accessible from the SKLearn library for Python, using their default parameters unless stated otherwise. We also implement a model manually using the precalculated parameters from the article. Lastly, we compute the probabilities that a certain word in a tweet is likely to make the tweet contain cyberbullying, similarly to what the authors present in the article.

4.1 Dataset preparation and collection

To perform our own training and calculate probabilities, mentioned before, we need to get the data, that has the same structure as the data Ho et al. [1], [2] used for their research. The feature vectors need to contain features, calculated by the tool LIWC. Since that tool is not open-source or free, we will use the same dataset, used as the baseline for training the model in the research from the original article we're reviewing. That dataset is found through a link in Kasture's MSc thesis work. It contains 1314 annotated tweets, where 376 of them are classified as ones containing cyberbullying. All of the annotations (classifying the tweets as cyberbullying or not) are decided subjectively and manually by people. Examples from the dataset can be seen in table 1.

What that means is that our model will be trained in the same way as in that article, however all the statistics for cyberbullying tweets might not be that accurate since we won't be able to collect many tweets on our own and process them through LIWC. We should however get a basic idea of the statistics and cyberbullying probabilities by just looking at the preannotated dataset.

For computing the statistics later on, Kasture also includes a dataset, which is not processed through LIWC and contains the original tweets and their cyberbullying classifications in the rows. That dataset will be used, when we compute probability statistics for bad words.

4.2 Logistic Regresion Implementation

4.2.1 Custom classifier

Table 1: Tweet examples from the dataset and their annotations.

Tweet content	Is cyberbullying
Are you sure I just received an email from Carly Haycock she said it has to be a US card as there is no other country	no
Just washed my car and she is looking sexy now my hands are like ice the things you go through to please a lady	no
Shut the fuck up stop pretending you re from the fucking hood because you live up pant and listen to fucking wannabe ra	yes
Bill DeMott can suck my Jordanian zibby Get colon cancer you fat fuck	yes

Logistic regression is one of the machine learning techniques, that fits a polynomial to the training data. Then, it can infer results by looking at the fitted polynomial and decides (upon setting a fixed threshold) whether some result is classified as a 0 (no or negative) or a 1 (yes or positive). In our example, if the polynomial equation for some point in the vector space outputs more than 0.5, we classify the point (representing a tweet) as 1 (is cyberbullying) and 0 otherwise.

To implement our own logistic regression model and use that as a binary classifier to detect whether a tweet contains cyberbullying, we will implement our own classifier in Python. We use the library sklearn and the module LogisticRegression to implement the classifier. We split the preannotated data into a 70/30 dataset, which has 70% of training feature vectors and 30% of testing feature vectors.

4.2.2 Recreating the articles classifier

To implement the exact same model that the authors of our reviewed article used, we implement a few functions manually, a prediction function and some utility functions to calculate the accuracy for our train test split data. We define a prediction function, that takes in the input vector of the 10 mentioned LIWC features, uses the weights defined in the original article and calculates the weighted sum for the current feature vector that is the input.

The prediction function is mathematically defined as:

$$y = -4.270 + 0.007 * you + 0.127 * negative\_emotion + 0.006 * anger - 0.235 * bio + 0.281 * body + 0.296 * health + 0.428 * sexual + 0.248 * ingest + 0.496 * death + 0.169 * swear \quad (1)$$

where the words are names of the features in the input vector. These weights were calculated by the authors training their Logistic Regression model in R.

To make a decision whether a tweet is cyberbullying based on the output of the prediction function, we use the same threshold value as the original article, which is 0.5. That means that when the y output from the prediction function is bigger than 0.5, we can say that the tweet contains cyberbullying and vice-versa.

4.3 Support vector machine classifier

Support vector machine classifiers embed all the feature vectors in a vector space and then try to find a hyperplane that best divides the space to be able to correctly classify new points, added to the space.

Along with the implementation of our own logistic regression classifier to compare with the authors original one, we also use SKLearn's *svm* module as the model for classification. For our particular case, we set the number of iterations of the algorithm in Python to 100000 for it to converge.

4.4 Random forest classifier

A random forest classifiers uses an ensemble of decision trees, where each tree judges the characteristics of the feature vector and outputs a prediction. Then, the most frequent output from the trees in the forest is taken as the output of the classifier.

To implement it in Python, we use SKLearn's *ensemble* module.

4.5 Naïve Bayesian classifier

Naive Bayesian classifiers use the Bayes theorem and the assumption that features in feature vectors are independent to find which output class is the most probable for a certain input.

To implement the classifier in Python, we use SKLearn's *naive_bayes* module.

4.6 Statistic Graphs Implementation

In the predecessor article to the one we are currently reviewing (also written by Ho et al. [1], [2]), we can see a graph where on the x axis there are bad words that we are using to extract potential cyberbullying tweets. We take the values from that graph and plot them next to our computed values, as is shown in the results section, to avoid having to draw two graphs (the original one and our one). On the y axis, there are probabilities. Those probabilities tell us what the probability that the current tweet is cyberbullying is, if it contains a certain bad word.

We read the article's values from the graph for each word and plot them as one line and then we also compute the same statistic on the LIWC dataset, found in Kasture's work and plot the line on the same graph. In that way we are going to be able to see how closely Kasture's small dataset resembles the big dataset that Ho et al. [1], [2] collected for their research, directly from the Twitter API.

To compute our probabilities, we go through each row of the dataset and look for any bad word from our bad word corpus. If the row contains a bad word, we create a new row in our fresh dataset, that is a vector containing features $[tweet, is\_cyberbullying, bad\_word\_found\_in\_tweet]$.

In that way, we can then, for each word, count the number of times that the bad word found in the tweet actually makes the tweet classified as cyberbullying and mark that as n_{cb}. Then we take that number and divide it by the total amount of tweets that contain the bad word n_{bw}, resulting in the following formula:

$$P_{cyberbullying}(word) = \frac{n_{cb}(word)}{n_{bw}(word)}.$$

The result is the probability that a certain bad word means that the tweet will contain cyberbullying.

5. RESULTS

In this chapter, we describe some methodologies how we obtained results and present the results along with the model accuracies.

5.1 Model accuracies

We measure the accuracy of our models simply by looking at the fraction of the tweets correctly classified as bullying or non-bullying versus the length of the testing dataset. It might be worth mentioning again at this point that we have trained our classifiers on the training data and computed the accuracy on the testing data, which does not overlap with the training data.

Our own best two trained models using sklearn (logistic regression and SVM) achieve the accuracy of 0.944. The naïve Bayesian classifier achieves the accuracy of 0.924, where the random forest classifier achieves the lowest score of 0.912. The pretrained model with hardcoded parameters from the article achieves the accuracy of 0.911. The best confusion matrix comes from our logistic regression model. The SVM confusion matrix shows more false negatives and false positives, so we will omit it from the article for clarity. We cannot really say whether the poorer performance is some sort of error since there is no information about the accuracy of the model, that the authors of the original article have computed.

We have also noticed that the dataset is imbalanced, which means that more than 70% of the tweets were classified as non-cyberbullying. We also tried a balancing algorithm SMOTE [12] to get a fully balanced dataset. The accuracy of our logistic model trained on this data is 0.926, on our SVM model it is 0.929, on the random forest model it is

Table 2: Confusion matrix for our own logistic regression classifier (without SMOTE).

Prediction / Ground truth	Positive	Negative
Positive	279	13
Negative	9	93

Table 3: Confusion matrix for the classifier with hardcoded parameters from the original article.

Prediction / Ground truth	Positive	Negative
Positive	276	16
Negative	19	83

0.901, and on the naïve Bayesian model it stays the same as before oversampling. Due to the poorer performance, we decided to omit this approach. If we had a bigger dataset, it might be worth revisiting this technique.

The confusion matrix for our logistic regression model is shown in Table 2 and the confusion matrix for the authors' model is shown in Table 3.

5.2 Cyberbully probabilities per word

The plots for per-word probabilities on both Kasture's annotated dataset (the one we used for training) and the authors' big dataset can be seen on Figure 2.

6. CONCLUSION

Even though our model seems to be highly accurate, we have to acknowledge the fact that it has been trained and tested on a very small dataset. The authors' pretrained model parameters seem to perform worse on this dataset, however, that does not mean that their model is any worse, since we do not know how it works on much bigger datasets from real life as we do not have LIWC to test that out.

However, even in our case, it seems that we can very well classify tweets as cyberbullying using those linguistic features as stated in the article.

The statistics, computed for the probabilities are also very different when we compare our results to the article's results, since we have performed our calculations on a much smaller dataset. Some of the values however differ a lot (e.g. the probabilites for word loser). Even though our dataset is significantly smaller, there should not be that big of a difference in those cases. Since the calculation approaches for these probabilites were not explained in depth by the authors, we might have even misunderstood the formula for computing our values.

Nevertheless, our own Logistic Regression model seems to perform very well and is also very fast since it follows a very traditional and simple machine learning approach. If we have the option to use LIWC to compute feature vectors very quickly, this could be a fantastic tool to not only detect cyberbullying inside tweets but also to process many tweets for different users and ultimately find cyberbullying hotspots very easily. It could also serve as baseground to potentially alert such toxic behaviour of users.

In conclusion of the original article the authors suggest that Twitter could ban the use of highly charged language but we think such automatic actions could be potentially dangerous. Highly charged language does not always immediately imply that cyberbullying happened and wrongful bans of users would hurt the integrity of the platform.

We feel like key aspects of cyberbullying were not fully represented in the article, especially not repetiton, considering that tweets that were used are all dated in a span of one week and no user information is used in predictive model. The issue here is of course also Twitter search API that only allows searches over the past seven days.

Systematic review of automatic cyberbullying detection [13] suggests that a lot of cyberbullying classification studies are actually classifying isolated online aggression cases and this also seems the case in original article. Main issue is the quality of datasets (i.e. single messages as opposed to a history of posts). The authors also used the previous dataset of only 1313 tweets for annotating their data. The annotation itself is a problematic subject because it requires experts in the field to properly decide what is cyberbullying and what isn't.

There are also many possibilities on how to use the same approaches and tools for other applications as well, not just for statistical analysis of words, like the authors seem to have focused more on in the original article.

For example, we could create many software tools or plugins that detect potential cyberbullying not only on Twitter but also on other chatting platforms such as Discord, Skype, Twitch.tv chat, etc.

The code for our experiments can be found on Github in the repository `https://github.com/andrazpovse/hate_speech`.

7. REFERENCES

[1] M. H. Shuyuan, K. Dayu, C.-H. Ming-Jung, L. Wenyi, and C.-J. Lai, "Detecting cyberbullying "hotspots" on twitter: A predictive analytics approach," 2020.

[2] M. H. Shuyuan, K. Dayu, C.-H. Ming-Jung, L. Wenyi, C.-J. Lai, and A. Bismark, "Charged language on twitter: A predictive model for cyberbullying to prevent victimization," 2020.

[3] S. Hinduja and J. W. Patchin, "Bullying, cyberbullying, and suicide," *Archives of suicide research*, vol. 14, no. 3, pp. 206–221, 2010.

[4] T. Waasdorp and C. Bradshaw, "The overlap between cyberbullying and traditional bullying," *The Journal of adolescent health : official publication of the Society for Adolescent Medicine*, vol. 56, 01 2015.

[5] "Summary of Our Cyberbullying Research (2007-2019)." `https://cyberbullying.org/summary-of-our-cyberbullying-research`. [Online; accessed 26-April-2020].

[6] J. Eck, S. Chainey, J. Cameron, and R. Wilson, "Mapping crime: Understanding hotspots," 2005.

[7] K. Dinakar, R. Reichart, and H. Lieberman, "Modeling the detection of textual cyberbullying," 01 2011.

[8] D. Yin, Z. Xue, L. Hong, B. D. Davison,

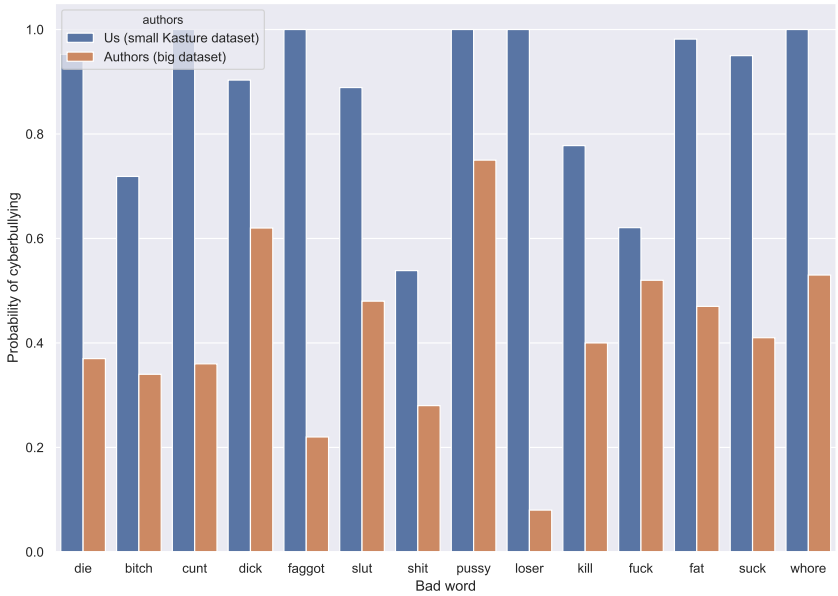


Figure 2: Probabilities of tweets containing cyberbullying per word.

A. Kontostathis, and L. Edwards, "Detection of harassment on web 2.0," *Proceedings of the Content Analysis in the WEB*, vol. 2, pp. 1–7, 2009.

[9] M. Dadvar, D. Trieschnigg, R. Ordelman, and F. de Jong, "Improving cyberbullying detection with user context," in *European Conference on Information Retrieval*, pp. 693–696, Springer, 2013.

[10] A. Edwards, K. Reynolds, A. Garron, and L. Edwards, "Detecting cyberbullying: Query terms and techniques," pp. 195–204, 05 2013.

[11] A. S. Kasture, *A predictive model to detect online cyberbullying*. PhD thesis, Auckland University of Technology, 2015.

[12] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[13] H. Rosa, N. Salgado Pereira, R. Ribeiro, P. Ferreira, J. Carvalho, S. Oliveira, L. Coheur, P. Paulino, A. M. Veiga Simão, and I. Trancoso, "Automatic cyberbullying detection: A systematic review," *Computers in Human Behavior*, vol. 93, pp. 333–345, 04 2019.