Univerza v Ljubljani

Fakulteta za računalništvo in informatiko

University of Ljubljana

Faculty of Computer and Information Science

Zbornik

Digitalna forenzika

Seminarske naloge, 2021/2022

Ljubljana, 2022

Zbornik

Digitalna forenzika, Seminarske naloge 2021/2022

Editors: Andrej Brodnik, Tom Fiette, študenti

Template author: David Klemenc

Ljubljana : Univerza v Ljubljani, Fakulteta za računalništvo in informatiko 2022.

Kazalo / Contents

1 Uvod / Introduction

Digital forensics is a branch of forensic science that covers the recovery and investigation of material found in digital devices and is often associated with computer crime. The term digital forensics was originally used as a synonym for computer forensics, but has expanded to include the investigation of all devices, which can store digital data. The roots can be traced to the personal computer revolution of late the seventies and early eighties. In the 1990s, the development of the discipline took place without real organization until national guidelines emerged in the 21st century.

Digital forensic investigations have different tasks. The most common are to support or refute the hypothesis before criminal or civil courts. Criminal cases involve an alleged violation of the laws it establishes legislation such as murder, theft and assault on the person. Civil cases deal with the protection of rights and property of individuals (often related to family disputes), but they can also deal with contractual ones disputes between economic entities, in which a form of digital forensics, which is called electronic, is involved discovery.

The collection contains the seminar papers of master's students at the Faculty of Computer Science and Information, University of Ljubljana 2021/2022. Within the Digital Forensics course, each group of students chose one article, which served as a starting point for the seminar work. The articles were selected from five research areas: malware, digital forensics of images and sound, network and service forensics, memory forensics, forensics tools and process, and followed by five articles that do not fall into any of the aforementioned categories.

In the field of malware, the group worked on the detection of adversarial attacking malware while the second dealt with the same idea but using neural network.

Since manipulation of images and sound has increased those last years, the assignment of digital forensics on this topic speaks about DeepFakes and Media forensics.

In network and service forensics, the seminar explains us how to make digital investigation using WhatsApp since this platform is widely used because of its encryption.

Memory forensics is becoming more and more relevant, as more and more methods are appearing that are able to obtain data from computer memory. The seminar assignments in this set cover on one hand the USB Device attacks and on the other hand the recovery of the master key from RAM to break Android's file-based encryption.

The tools are really important to improve the accuracy of digital forensics and spread its use. That's why the seminar deals with Generating digital forensic test images.

Finally, the miscellaneous section, there are seminar assignments that do not fall under any of the mentioned sections : they dealt with the Analysis of the car camera system using metadata, Review of An Enhanced Blockchain-Based IoT Digital Forensics Architecture Using Fuzzy Hash, and Challenges of education and standardization in the field of mobile forensics. That shows that Digital Forensics is an increasing field of search.

This collection brings together all the final seminar assignments that were prepared in the academic year 2021/2022. It is intended for everyone who is interested in the field of digital forensics or just one or more areas presented.

Ljubljana, 2022 Tom Fiette

2 Povzetki / Summaries

2.1 Detecting Adversarial Attacking Malware

The number of smartphone users has been growing exponentially in the past decade and with them the volume and variety of malware. Currently malware detection engines are having trouble catching up to all the different malware. Because of that researchers have started to use machine learning and deep learning to develop malware detection models. Because machine learning and deep learning models are vulnerable to adversarial attacks, a group of researchers has proposed a framework to construct malware detection models against such adversarial attacks. First they constructed twelve different malware detection models and attacked them with a GAAN based attack, which modifies malware samples to fool the malware detection models. The attack was very successful in fooling the malware detection models so the researchers proposed three defense strategies, which were very effective against a GAAN based attack. They also identified a list of Android permissions and intents, which were used most of the time to force misclassifications in detection models.

Ključne besede / Keywords : Malware detection, Android, machine learning

2.2 Detekcija zlonamerne programske opreme z uporabo konvolucijskih nevronskih mrež na podlagi kratkih binarnih segmentov

Nobena programska oprema ni popolnoma varna. V praksi se pogosto srečamo z novimi napadi, ki poskušajo izkoriščati ranljivost tarčne programske opreme. V delu bomo raziskali področje detektiranja zlonamernih programskih oprem še preden te škodujejo sistemu. Osredotočili smo se na prepoznavanje takih programov na podlagi kratkih binarnih segmentov s pomočjo nevronskih mrež. Članek predstavlja model strojnega učenja, ki na simulacijah realnih primerov dosega do 86,92% točnost pri zaznavanju tovrstnih virusov.

Ključne besede / Keywords : detekcija zlonamerne programske opreme, detekcija segmentov zlonamerne programske opreme, napadi ničtega dne, konvolucijske neuronske mreže

2.3 Media Forensics and DeepFakes: An Overview

In recent years, techniques of generating and manipulating visual media have developed to a point where forged images are often indistinguishable from real ones. There is significant potential for abuse and conventional methods of detecting manipulation are no longer powerful enough. For that reason, more advanced methods based on deep learning have been developed to tackle the problem of detecting deepfakes.

Ključne besede / Keywords : media forensics, deepfakes, deep learning, image analysis

2.4 Analiza sistema avto kamer s pomočjo metapodatkov

Glavna tema dela so avto kamere. Avto kamera je lahko uporaben vir za dokazovanja resnice pri prometnih nesrečah na sodišču, prav tako pa lahko posnetek tudi zajame kakšno kriminalno dejanje na ulici. Zajema veliko uporabnih podatkov, ki se jih pri forenzični analizi lahko uporabi, na primer, čas, lokacija, hitrost in podobno. V delu se dotaknemo osnov delovanja avto kamer, njihove uporabe, obdelave posnetkov in forenzične analize posnetkov, še posebej s pomočjo metapodatkov, ki jih kamere med svojim delovanjem generirajo. Dotaknemo se tudi legalnosti uporabe avto kamer v Združenih državah Amerike in v Sloveniji. Poleg tega se tudi sami lotimo pridobivanja meta podatkov iz posnetka avto kamere, pridobljenega iz interneta. Analiziramo tudi posnetek, ki ga pridobimo iz mobilne aplikacije DroidDashcam, ki deluje kot avto kamera. Iz posnetka katerega pridobimo iz interneta uspešno pridobimo meta podatke s pomočjo orodja exiftools, medtem ko smo pri pridobivanju podatkov iz posnetka, katerega je generirala aplikacija DroidDascham neuspešni. Iz raziskovanega članka pridemo do zaključkov, da se meta podatki glede na različne modele avto kamer precej razlikujejo, hkrati pa so zelo uporabni za rekonstrukcijo dogodkov na primer ob prometni nesreči.

Ključne besede / Keywords : Avto kamere, videoposnetki, analiza, pridobivanje podatkov

2.5 Review of An Enhanced Blockchain-Based IoT Digital Forensics Architecture Using Fuzzy Hash

A recently proposed way to achieve better IoT security using blockchain technology includes using an immutable ledger, a decentralization architecture and a well-established low-level cryptographic algorithm. An issue that presents itself is that not all IoT devices are compatible with existing implementations of this idea, as some of them use conventional hash, which is useless for comparing similar files (as even the smallest change causes the hash to be completely different). The authors of the reviewed paper [5] propose an approach using fuzzy hash (in addition to conventional hash for authentication) to construct the Blockchain's Merkle tree, which allows the discovery of altered files by comparing its hash to the one in the Blockchain network. We propose and implement an improved approach, which could be beneficial to all types of digital forensics.

Ključne besede / Keywords : Blockchain, Internet of Things, Fuzzy hash, IoT forensics, digital examination

2.6 Izzivi izobraževanja in standardizacije na področju mobilne forenzike

Usposabljanja na področju mobilne forenzike so v današnjihčasih vedno bolj pomembna. Seminarska naloga predstavlja nekatere izmed izzivov s katerimi se zaradi pomanjkanja leteh srečujejo udeleženci forenzične preiskave. To področje se vedno bolj razvija, tudi kriminalci vedno bolj izkoriščajo mobilne naprave za izvajanje raznih zločinov. Z namenom zagotavljanja boljšega usposabljanja je bilo narejenih tudi že veliko raziskav. V članku, ki sva ga uporabili za zgled, je predstavljenih tudi nekaj rezultatov glede kategorizacije usposabljanj ter mnenj o pridobivanju posameznih spretnosti oz. pomanjkljivostih. Naredili sva še kratko analizo predmetnikov na slovenskih fakultetah, ki vsaj delno pokrivajo področje povezano z mobilno forenziko.

Ključne besede / Keywords : mobilna forenzika, standardizacija

2.7 Digital investigation using WhatsApp

WhatsApp is a messaging app used worldwide by more than a billion people and has become part of everyday life for many as a free alternative to SMS. Therefore, it could potentially give investigators useful information. However, obtaining those information does not come without hurdles because WhatsApp started to have end-to-end encryption in 2016, making real-time insight much harder to obtain. Before 2016, WhatsApp was in plain text XMPP and was not nearly as problematic for digital investigators and analysts. WhatsApp is a double-edged sword for digital forensics: it could be a great source of information (text messages, audio ones, calls, photos files, etc.) but many elements can hinder the access to them due to deletion and end-to-end encryption. This report gives an overview of the challenges specific of WhatsApp now that it is end-to-end encrypted and explains methods of how one can go about getting real-time as well as non real-time information through WhatsApp. As each approach to get information has both benefits and drawbacks, we will discuss their use and provide conditions that would make some techniques more relevant.

Ključne besede / Keywords : WhatsApp forensics, Mobile forensics, Investigation, Encryption

2.8 Forenzika pomnilnika napadov z USB napravami

USB naprave ne predstavljajo le fizične shrambe datotek, ki jo lahko imamo vedno pri roki in jo brez težav priklapljamo v različne operacijske sisteme, ampak predstavljajo tudi vektor napada, če ima napadalec fizični dostop do računalnika oz. sistema, ki ga želi napasti. V članku se bomo osredotočili predvsem na forenziko pomnilnika, ki je posledica napada naprave Hak5 Rubber Ducky in Bash Bunny [6]. Napad je bil izveden na računalnik Windows 10, osredotočili pa se bomo na artefakte slik iz sistemskega pomnilnika, ki so bile izvečene s pomočjo orodij usbhunt in dhcphunt. Za potrebe forenzike je bilo potrebno preveriti tudi DHCP izpise, ki so omogočili vpogled v dejavnost omrežja.

Ključne besede / Keywords : Digitalna forenzika, pomnilnik, USB naprave, Rubber Ducky, Bash Bunny

2.9 Recovering the master key from RAM to break Android's file-based encryption

One of the major challenges in today's digital forensics is how to acquire data from an encrypted storage device without knowing the right decryption key. One possible way to acquire the key, without breaking the cypher, is by extracting a key from the systems memory which might be possible if we have physical access to a powered on device and we are able

to make a copy of the memory's content. In this essay, we review and discuss how it might be possible to acquire a master key from a modern Android system that uses File-based encryption.

Ključne besede / Keywords : Digital forensics, Computer security, Mobile phone security, Cold boot attack

2.10 Generating digital forensic test images

With the increasingly diverse landscape of digital evidence, both the experts and tools of digital forensics must be properly certified to deal with it effectively. In order to achieve this goal, quality datasets of digital data must be made available. Manual generation of test images can be very time consuming, and usage of second-hand drives presents privacy concerns. As such, emulating user and system actions presents itself as a valid solution. Several tools have been developed to address this need. In this seminar paper, we present a few existing approaches for generating synthetic digital forensic datasets, with focus on TraceGen. TraceGen is a novel framework for automating user-generated stories through virtualization technologies. Through the use of Python scripts, it can reproduce various user behaviour. Additionally, we cover select mobile and network forensic dataset generation approaches. We find that each approach has its advantages and drawbacks, and that a unified overview of the field would be of great benefit.

Ključne besede / Keywords : data generation, user emulation, forensic images

3 - Slabogramje / Malware

Detecting Adversarial Attacking Malware

Jurij Kolenik
Faculty of Computer and Information Science
University of Ljubljana
jk8120@student.uni-lj.si

Robert Modic
Faculty of Computer and Information Science
University of Ljubljana
rm1052@student.uni-lj.si

Bine Stančič
Faculty of Computer and Information Science
University of Ljubljana
bs1690@student.uni-lj.si

ABSTRACT

The number of smartphone users has been growing exponentially in the past decade and with them the volume and variety of malware. Currently malware detection engines are having trouble catching up to all the different malware. Because of that researchers have started to use machine learning and deep learning to develop malware detection models. Because machine learning and deep learning models are vulnerable to adversarial attacks, a group of researchers has proposed a framework to construct malware detection models against such adversarial attacks. [6] First they constructed twelve different malware detection models and attacked them with a GAAN based attack, which modifies malware samples to fool the malware detection models. The attack was very successful in fooling the malware detection models so the researchers proposed three defense strategies, which were very effective against a GAAN based attack. They also identified a list of Android permissions and intents, which were used most of the time to force misclassifications in detection models.

Keywords

Malware detection, Android, machine learning

1. INTRODUCTION

Smartphones have become an integral part of our modern way of living. They store various user information like videos, photos, social media, e-mail, bank accounts, etc. Android applications themselves also store certain amount of data on the device. All this information has become a lucrative target for malware designers who wish to exploit it for monetary gain. Android has the biggest market share (71.6%) in terms of mobile communication devices. The application can also be installed directly on the device without using stores, meaning checks designed to prevent malicious software distribution from Google, Samsung and other store providers are not made.

1.1 Article structure

In this article we study a proposed approach of Android malware detection by training models using an adversarial attacking malware. In section 2 we discuss why malware detection on Android devices is important and challenging. In section 3 we discuss the proposed model for training the malware detection model, the way data was collected and features extracted. We then look at the effectiveness of the 12 basic models followed by the improvements of 3 models trained on the adversarial attacking malware in section 4.

2. BACKGROUND

The primary defenses against malware attacks are designed and developed by anti-malware community and anti-virus companies. These solutions depend upon signature heuristics and behaviour based detection engines to tackle the problem of detecting new and old malware. These mechanisms are considered to be slow, un-scalable and in many cases reactive driven, meaning they detect the infection after it has already occurred. This is why another approach using machine or deep learning might be worth looking at. [2]

The building of a malware detection model is a two stage process: (1) Feature extraction and (2) Classification. Extracted features from Android applications consist of extracting application permissions and based on them build a model that would detect malware. However research in other machine learning domains, such as image classification show that these models can be easily fooled [4]. This is achieved by slightly modifying an image in such a way that humans cannot see the difference and similar approach might be used to fool a malware detection model.

Threat modeling of adversarial attacks is defined based on adversary's goal, knowledge and capabilities for the target system. In this work we present the process of modeling where authors of original work first performed feature extraction from Android applications (malware and benign) and built several baseline malware detection models. Next they acted as an attacker and performed evasion attack on these baseline models. They proposed Gradient Adversarial Attack Network (GAAN) to perform these attacks and find vulnerabilities and reduce the performance of beforementioned models. This attack network is designed to perform minimal modifications on malware samples. The authors also ensured that they did not break application's functionality and behavioural aspects.

Authors performed GAAN based attacks on two feature vectors, one based on permissions and one based on intent. This way they tried to validate the generalisability of the solution against different detection models. Later on they proposed three defenses, *Adversarial Retraining, GAN* and *Hybrid Distillation*. Next they compared the performance of newly acquired knowledge and fed it back into detection models.

3. METHODS

3.1 Problem definition

The authors define problem as where dataset (D) can be represented as:

$$D = \{\{(x_j, y_j)\}, \forall j \in (1, 2, ..., n)\} \in (X, Y) \quad (1)$$

Dataset (D) contains sets of benign (B) and malicious (M) Android applications. X represents Android applications and Y their designation (malware/benign). The applications X can be represented with features like permissions, intent, system call. The authors represent these as binary feature vectors, meaning that using this notation they can be used in construction of many different kinds of classification models. The performance of said models can be evaluated using different success metrics, such as accuracy, precision, recall, AUC, etc.

The attack on malware detection models aims to force of fool them into misclassifying samples. The defence strategy aims to improve the robustness of malware detection. This is achieved by learning from successful attacks on malware detection models.

3.2 Framework

Figure 1 shows the framework architecture. The first step is data collection (gathering of malicious and benign samples), the second step consists of feature extraction and building feature vectors. The authors extracted permissions and intent to build two separate feature vectors representing Android applications. Using these vectors in third step authors build twelve malware detection models using different classification methods. In fourth step they validate the performance using GAAN-based attack and then use what they learned in fifth and final step by using three different defence mechanisms to improve malware detection models and make them more robust.

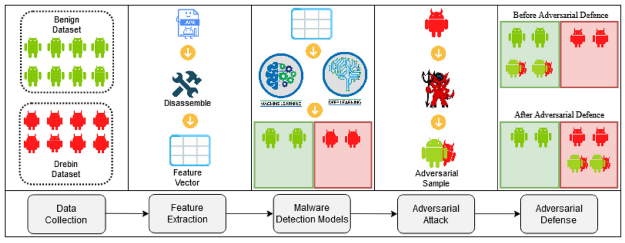


Figure 1: Proposed framework for constructing robust malware detection models.

3.3 Attack strategy

The attack strategy is designed for compromising and reducing the performance of detection models. The authors propose GAAN based attack for grey-box scenario, where attacker has knowledge about feature vectors and dataset used to construct malware detection models, but no knowledge about algorithms and methods used to train them. This strategy aims to modify malware samples to force detection models to misclassify them as benign.

The proposed GAAN is a deep neural network consisting of three hidden layers. First GAAN is trained on D. It then calculates whitch features from feature vectors are most likely to increase the loss. Using this the feature vectors are then modified so that the malware detection models misclassify them. The algorithm used to modify these vectors takes as an input GAAN model, malware sample to be converted into adversarial sample, targeted malware detection model and a maximum number of features to be modified. The algorithm then determines the features in the sample to modify based on GAAN model and its input and returns an adversarial sample and list of features added to it. The pseudo-code of this algorithm can be seen in Figure 2.

Algorithm 1 Algorithm for GAAN based attack

Input:
GAAN: a deep neural network trained on original dataset
m: malware sample to be converted into adversarial sample
target_model: targeted malware detection model
max_mod: maximum number of modifications allowed in m
Function:
predict: prediction function of target_model
get_feature: returns list of features not used by m
gradient_sort: performs sorting on the list (descending order)
modify: add the feature in malware sample m
get_name: get the name of the feature
Output: (m', $modified\_list$)

$pred \leftarrow predict(target\_model, m)$
if $(pred == 1)$ **then**
  $features \leftarrow get\_features(m)$
  for i in $len(features)$ **do**
    $feature\_gradient \leftarrow feature\_gradient \cup (\{\frac{\partial Loss}{\partial features[i]}\}, i)$
  end for
  $feature\_gradient \leftarrow gradient\_sort(feature\_gradient)$
  for $i \leq max\_mod$ **do**
    $m' \leftarrow modify(feature\_gradient[i][1], m)$
    $modified\_list = modified\_list \cup get\_name(feature\_gradient[i][1])$
    if $(predict(target\_model, m') == 0)$ **then**
      break
    end if
  end for
end if
return (m', $modified\_list$)

Figure 2: Pseudo-code of attack strategy algorithm.

3.4 Defence strategy

The authors propose three defence strategies to counter GAAN based attack.

The first strategy they describe is *Adversarial Training*. They retrain all 12 models using original samples and samples from GAAN attack that managed to fool them. Doing so helps newly trained models to learn new patterns and make them more robust in the future.

The second defensive strategy involves retraining malware detection models using GAN (Generative Adversarial Network). Using this neural network they construct adversarial samples and use them in training. Their proposed solution consists of two neural networks, *Generator* G and *Discriminator* D. They try to optimize their actions by undermining each other and minimizing each others loss. During training the generator adds noise to a malware sample to produce an adversarial sample that could fool the discriminator. There is also no restrictions on the number of features that generator is allowed to modify. The set of this adversarial samples is later added to original dataset for the purpose of retraining.

The third strategy is *Hybrid Distillation*, which is an updated version of distillation. Distillation is a compression method where knowledge is transferred from a larger to a smaller model. Authors used this method as a defence against adversarial attacks. On the first step they train detection model to learn the class probability distribution of dataset D. Later the second model is trained on the same dataset D but mapped to the class probabilities of the first model (rather than to the classification labels). This method is used to construct larger dataset containing adversarial samples and samples from original dataset.

3.5 Data and feature extraction

Google Play Store is home to 3.48 million apps. Google uses a built in application Google Play Protect to check the safety of these apps, but it is unable to detect all unsafe apps. Another security measure in Android devices is the permission system, where the user must grant each application permission for certain tasks such as sending SMS, accessing the camera, managing documents etc. However this was shown to be ineffective at preventing malware. [3]

The group took 5,560 malicious applications from the DREBIN dataset [1]. The DREBIN dataset was made by combining applications from GooglePlay Store, applications from alternative Chinese and Russian markets as well as applications from different Android websites. All the applications were then run through 10 anti-virus scanners from VirusTotal, in order to classify as malware at least 2 of the 10 scanners had to classify them as malware. These 5,560 malicious applications were combined with 5,721 safe applications from the Google Play Store, which were confirmed to be safe by all of 50 antivirus scanners in VirusTotal.

In order to build a feature set for the classification models the group extracted permissions and intents from every one of the applications. Android applications need permissions to use different features and resources of the device and intents to launch different activities. [5] These were obtained from the manifest file (*AndroidManifest.xml*) which must be present in every Android application. Two feature vectors were constructed for 195 permissions and 273 intents for every one of the applications. The result being a less complex version of the feature set constructed in the DREBIN paper [1] where they looked at both the manifest file as well as disassembled bytecode to get additional features such as restricted API calls which surpass the permission system, suspicious API calls for accessing sensitive data (`getDeviceId()`, `getSubscriberId()`), communicating over network (`setWifiEnabled()`, `execHttpRequest()`), SMS messaging (`sendTextMessage()`) as well as API calls used for hiding actions (`Cipher.getInstance()`). They also looked at all IP addresses, hostnames and URLs found in the dissasembled code which could have been used by malware to retreive commands or data from the device. All these features were then combined into a joint vector space, to give a geometric representation of the apps features.

3.6 Classification Models

The group decided to go for a broad approach by building 12 different classification algorithms: Decision Tree (DT), Support Vector Classifier (SVC), Random Forest (RF), Extra Trees Classifier (ET), Bagging based SVC (Bagged SVC), Gradient Boosting (GB), Adaptive Boosting (AB), eXtreme Gradient Boosting (XGB) and DNN with (1,3,4,5) hidden layer(s).

We are unsure why the group decided to test 12 different algorithms instead on focusing on a handful. We also cannot find any information on the 'weak learner' algorithms resulting in high variance or bias that would call for the use of ensemble methods. At the same time using DNN on a relatively small dataset of 11,281 applications with already defined features is also inefficient. A better use of these algorithms would be to take the entire DREBIN dataset with 123,453 benign applications and 5,560 malware samples. And from each application extract the disassembled code and let the DNN identify the important features. However that would be a much more computationally demanding task. We also cannot find any reason for the different numbers of hidden layers used in the DNN.

Instead of just focusing on binary classification of applications the DREBIN paper [1] decided to combine all 545,000 features in a vector space. Just by using a linear SVM they were able to train the model to determine a hyperplane which separates benign applications from malware with maximal margin. By looking at how changing a particular feature of and app changes the position of that app with relation to the hyperplane, they were able to get insight on why an app was classified one way or the other. It gives user a detection score on how how confident the classification is along with explanations on what API calls or permissions caused the result. Once the SMV model is trained it can be transferred onto mobile devices where each application can then be categorised as malware or safe. However an issue with this method is that with such a large number of features the model can suffer from curse of dimensionality.

4. EXPERIMENTAL RESULTS

4.1 Baseline performance of malware detection models

With the acquired dataset of applications, the authors constructed the twelve models, where 70% of the dataset is used for training and 30% for testing. Each detection model is evaluated based on accuracy, precision, recall and AUC.

The first set of experiments tested the performance of malware detection models using Android permissions. The average malware detection accuracy was 93.35%, with ET being the highest with 95.27% and DNN-1L being the lowest with

89.74%. Overall the ensemble-based models were the most accurate. When measuring the AUC score, similar results are observed where again ET-based model has the highest score and DNN-1L has the lowest.

The second set of experiments tested the performance of detection models using Android intents. RF and DT based detection models were the most accurate with 81.97% accuracy and DNN-1L the least with 77.99% accuracy. Compared to permission-based models, the intent-based models were less accurate.

4.2 Adversarial attack on malware detection models

After constructing and testing the baseline malware detection models, the authors performed GAAN based adversarial attacks on them. The GAAN based attack takes the malware samples used previously and makes a small change to each of them, where the goal is that the changes will make the detection models misclassify the samples and so increase the fooling rate. However, the modified malware sample still needs to be functional, so the modification of the samples was restricted to addition only. Two GAAN models were made, one for intents and one for permissions.

4.2.1 Fooling rate against baseline detection models

The fooling rate was defined as the percentage of malicious samples successfully modified for misclassification. The strategy of testing the fooling rate was to minimize the number of modifications required in each malicious sample for misclassification while maximizing the total number of successfully modified malicious samples in the dataset. The attacks ranged from 1-bit to 10-bit, meaning 1-bit attacks made 1 change to the malware sample and 10-bits made 10 changes to the malware sample.

The fooling rate achieved by the GAAN attack against the twelve permission-based baseline malware detection models can be seen on Figure 3.

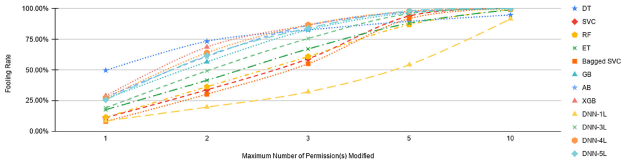


Figure 3: Performance of permission-based malware detection models against GAAN attack.

The 1-bit attack reached an average of fooling rate of 21.86% and subsequent 2-bit and 3-bit achieved the average fooling rate of 49.70% and 71.53% respectively, meaning that the first few modifications have the biggest impact in malware being misclassified.

Figure 4 shows the fooling rate of GAAN attacks on intent-based malware detection models. The initial 1-bit and 2-bit attacks achieved the average fooling rate of 32.82% and 53.91% respectively, which is higher than the initial fooling rate in permission-based malware detection models. However, the 10-bit attack reached an average fooling rate of 90.71% which is lower than the average of 99% that he permission-based models achieved. Overall the GAAN based attack was successful, because even a 10-bit change in both cases achieved an average fooling rate higher than 90%.

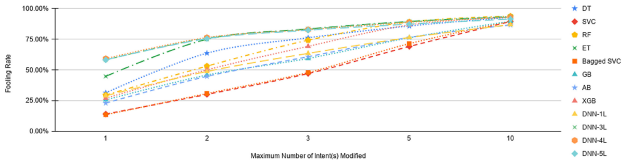


Figure 4: Performance of intent-based malware detection models against GAAN attack.

4.2.2 Vulnerability list

When performing the GAAN based adversarial attack, the authors monitored which permissions and intents were added and created a list of most frequent characteristics changed in 1-bit and 10-bit attacks. In the permission based attack, both the 1-bit and 10-bit attack had the same 5 permissions, that were used for modification the most. The permissions were:

- android.permission.SYSTEM_ALERT_WINDOW
- android.permission.READ_CONTACTS
- android.permission.READ_CALL_LOG
- android.permission.USE_CREDENTIALS
- android.permission.CALL_PHONE

In the intents based attacks, the top four most used intents were the sam for 1-bit and 10-bit attacks. The intents were:

- android.intent.action.SEND
- android.intent.action.DEFAULT
- android.intent.action.VIEW
- android.intent.action.PACKAGE_REPLACED

4.3 Defensive strategies for malware detection models

To defend against GAAN based attacks the authors proposed three defensive strategies for detection models, adversarial retraining, GAN retraining and hybrid distillation. The adversarial retraining uses a balanced dataset, which includes the modified malware samples generated by GAAN attack, to retrain the detection models. The GAN defense strategy uses GAN (generative adversarial network) to generate adversarial samples which are then added to the dataset with their correct labels. The dataset is then used to retrain the detection models. Hybrid distillation combines both methods, where distillation is applied on the augmented dataset with adversarial samples rather then training on the original dataset.

4.3.1 10-bit adversarial attack on defense strategies

The authors performed the GAAN-based attack on malware detection models that were trained with three defense strategies. Figure 5 shows the accuracy of the permission-based models after a 10-bit attack. Malware detection models us-

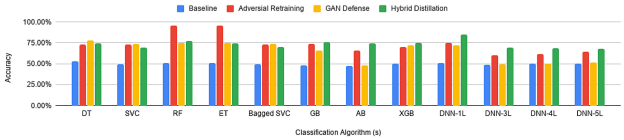


Figure 5: Performance of baseline and upgraded permission-based malware detection models against GAAN attack.

ing the defense strategies outperformed the baseline models with RF and ET based models with adversarial retraining reaching 95% accuracy on the 10-bit attack. On average adversarial retraining strategy had 55.86% accuracy improvement, hybrid distillation 54.21% and GAN retraining 36.87%.

Figure 6 shows the accuracy of the intent-based models after a 10-bit attack. The three defense strategies show improve-

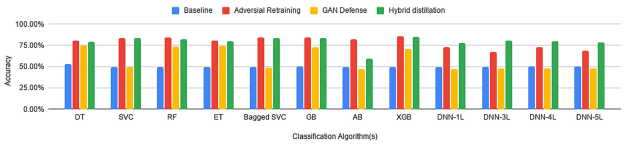


Figure 6: Performance of baseline and upgraded intent-based malware detection models against GAAN attack.

ment after 10-bit attack compared to the baseline models. Adversarial retraining achieved an average increase in accuracy of 58.18% GAN based defense 17.62% and hybrid distillation 59.14%. Overall adversarial retraining and hybrid distillation defense strategies are most effective while GAN based retraining is less efective on improving accuracy of the models.

5. CONCLUSIONS

The group proposed the GAAN model which was very effective at fooling the 12 malware detection models both for permission-based and intent-based. The 10-bit attack achieved an average of 98.68% fooling rate and a 90.71% fooling rate for the permission and intent based models. The group then proposed three new defence strategies for detecting GAAN attacks. All three models were more accurate at detecting GAAN attacks improving the accuracy by 55.86% for adversarial retraining strategy, 54.21% for hybrid distillation and 36.87% for GAN retraining. We assume this increase in accuracy is due to the models having more data used for their training.

An interesting addition would be for the GAAN model to not have any information on the data used to train the 12 detection models, as this is a more real world representation. As well as finding a way for the models to be retrained once a new application is released.

References

[1] Daniel Arp et al. “Drebin: Effective and explainable detection of android malware in your pocket.” In: *Ndss*. Vol. 14. 2014, pp. 23–26.

[2] Ömer Aslan Aslan and Refik Samet. “A comprehensive review on malware detection approaches”. In: *IEEE Access* 8 (2020), pp. 6249–6271.

[3] Kevin Benton, L Jean Camp, and Vaibhav Garg. “Studying the effectiveness of android application permissions requests”. In: *2013 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*. IEEE. 2013, pp. 291–296.

[4] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. “Explaining and harnessing adversarial examples”. In: *arXiv preprint arXiv:1412.6572* (2014).

[5] Fauzia Idrees and Muttukrishnan Rajarajan. “Investigating the android intents and permissions for malware detection”. In: *2014 IEEE 10th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. 2014, pp. 354–358. DOI: 10.1109/WiMOB.2014.6962194.

[6] Hemant Rathore et al. “Robust Malware Detection Models: Learning from Adversarial Attacks and Defenses”. In: *Forensic Science International: Digital Investigation* 37 (2021), p. 301183.

Detekcija zlonamerne programske opreme z uporabo konvolucijskih nevronskih mrež na podlagi kratkih binarnih segmentov

Miha Petrišič
Univerza v Ljubljani, Fakulteta za računalništvo in informatiko
Večna pot 113
Ljubljana, Slovenija
mp6079@student.uni-lj.si

Saša Ošlaj
Univerza v Ljubljani, Fakulteta za računačništvo in informatiko
Večna pot 113
Ljubljana, Slovenija
so8718@student.uni-lj.si

Peter Hrovat
Univerza v Ljubljani, Fakulteta za računalništvo in informatiko
Večna pot 113
Ljubljana, Slovenija
ph2355@student.uni-lj.si

POVZETEK

Nobena programska oprema ni popolnoma varna. V praksi se pogosto srečamo z novimi napadi, ki poskušajo izkoriščati ranljivost tarčne programske opreme. V delu bomo raziskali področje detektiranja zlonamernih programskih oprem še preden te škodujejo sistemu. Osredotočili smo se na prepoznavanje takih programov na podlagi kratkih binarnih segmentov s pomočjo nevronskih mrež. Članek predstavlja model strojnega učenja, ki na simulacijah realnih primerov dosega do 86,92% točnost pri zaznavanju tovrstnih virusov.

Ključne besede

detekcija zlonamerne programske opreme, detekcija segmentov zlonamerne programske opreme, napadi ničtega dne, konvolucijske neuronske mreže

1. UVOD

Z izrazom zlonamerna programska oprema opredeljujemo kakršno koli programsko opremo, razvito z namenom škodovanja računalniku ali njegovi vsebini. Z razvojem tehnologije, še posebej spletnih tehnologij, se je število zlonamerne programske opreme drastično povečalo. Ustvarjalci programske opreme so tako primorani k iskanju vedno novih varnostnih rešitev, kljub temu pa se bodo vedno našli zlonamerni iznajdljivci, ki so sposobni najti še tako majhno varnostno luknjo. Zaradi same kompleksnosti, razširjenosti in seveda finančnih nevarnosti ter varovanja osebnih podatkov, se pojavlja velika potreba po prepoznavanju take programske opreme, še preden lahko ta škoduje uporabniku. Metod za prepoznavanje zlonamernih programov je ogromno, med najuspešnejšimi pa se pogosto znajdejo tiste, ki znajo izkoristiti moč globokega učenja, kot so na primer konvolucijske nevronske mreže, katere bomo tudi podrobneje spoznali.

V začetku bomo spoznali lastnosti napadov ničtega dne. Opisali bomo različne pristope odkrivanja tovrstnih napadov, s poudarkom na metodah različnih raziskovalcev, ki temeljijo na strojnem učenju. Jedro članka obsega reševanje danega problema s pomočjo konvolucijskih mrež. V tem delu bomo predstavili metodologijo, s katero so avtorji originalnega članka [14] skušali čim natančneje posnemati dejanske napade, in arhitekturo konvolucijske mreže, ki je bila oblikovana za reševanje tovrstne problematike. Na koncu pa bomo podali in analizirali rezultate uporabljenega modela.

2. SORODNA DELA

2.1 Ranljivost ničtega dne

S terminom ranljivost ničtega dne (*angl. zero-day vulnerability*) imenujemo še neodkrito ranljivost v programski opremi, ki jo lahko vdiralci izkoristijo. Taki napadi so še posebej nevarni, saj izkoriščajo novo ranljivost, ki se je niti avtor programske opreme najverjetneje ne zaveda [12].

Vpliv napadov ničtega dne je zelo odvisen od samega načina detekcije takih programov, narave napadenega programa, kako hitro se odkrita ranljivost širi med drugimi napadalci, kako hitro ter učinkovito je ranljivost odpravljena ter mnogi drugi faktorji. Za izkoriščanje ranljivosti nekega programskega sistema je potrebno podrobno znanje o samem sistemu. Prva stopnja izdelovanja take zlonamerne programske opreme je odkrivanje ranljivih elementov v tarčnem programu. Skoraj vsaka programska oprema nosi kakšno napako ali pomanjkljivost, za katere lahko uporabljamo avtomatska ogrodja za detekcijo, kot na primer fuzzers [6], vendar nobeno ogrodje ni tako dovršeno, da lahko zazna varnostne napake, kot strokovno, ročno pregledovanje kode. S tem razlogom se mnoga večja podjetja poslužujejo tako imenovanim ”bug bounty” programom, ki ponujajo posameznikom izven podjetja nagrado za odkritja ranljivosti in varnostnim luknjam. [13].

Za zaščito pred napadi ničtega dne se uporabljajo trije protiukrepi, preventiva, ki nastopi pred napadom, detekcija, med oziroma tik pred napadom, in ublažitev po napadu. V nadaljevanju se bomo osredotočili na detekcijo zlonamernih programskih oprem.

2.2 Detekcija

Tradicionalno, antivirusni programi uporabljajo metode na osnovi podpisov za detekcijo zlonamerne programske opreme[3]. Podpisi so v tem primeru kakršne koli sledi, ki so jih zlonamerne programske opreme v preteklosti pustile. Hranjenje teh sledi, ki lahko predstavljajo same odseke zlonamernega programa ali pa tudi dejavnosti tovrstnih programov, nam lahko služijo za detekcijo podobnih programov v prihodnosti. Te metode pa so zaradi njihove narave predvsem neučinkovite za detekcijo napadov ničtega dne, saj niso zmožne prepoznati programa, ki je za njih ne poznan. Novejše metode detekcije s tem razlogom uporabljajo strojno učenje, ki se je izkazalo za učinkovito pri odkrivanju zlonamerne programske opreme.

Poznamo različne načine premagovanja virusov. Eden izmed njih je dinamični način, ki poskuša klasificirati neznano programsko opremo tako, da opazuje obnašanje programa v kontroliranem okolju. Ena od študij na tem področju se je odločila za podobno metodologijo in poskušala prepoznati tip programske opreme (škodljiva ali neškodljiva) na podlagi osnovnih algoritmov strojnega učenja [2]. Za zbiranje podatkov o obnašanju tarčnega programa med izvajanjem, se uporabljajo storitve, kot na primer Anubis [1], ki nudijo avtomatsko analizo ter beleženje pomembnih metrik in značilk na podlagi katerih se bo model strojnega učenja učil. Za najbolj uspešna so se v študiji izkazala odločitvena drevesa, s klasifikacijsko točnostjo 96,8%. Ostale uporabljene metode so bile: metoda podpornih vektorjev z 92,9% točnostjo, k-najbližjih sosedov s prav tako 92,9% točnostjo ter naivni Bayes z 92,3% točnostjo.

Čeprav se zdi dinamičen pristop obetaven glede na same rezultate, je zanj potrebno veliko režije z uporabo virtualnih okolij in ogrodij, kjer se sam program izvaja, hkrati pa nekateri pametnejši programi uporabljajo vzorce normalnih, neškodljivih programov, da zamaskirajo obnašanje. Povrh vsega pa je dinamičen pristop zaradi same izvedbe programa velikokrat časovno potraten in porablja veliko virov.

Drug način detekcije je uporaba statičnih metod. To so tiste, ki poskušajo klasificirati neznano programsko opremo na podlagi njene kode. Ponavljajoče se nevronske mreže (*angl. recurrent neural networks oz. RNN*) se pogosto uporabljajo za naravno procesiranje jezika, zato se je pojavila ideja, da bi na podoben način, s takimi mrežami poskušali prepoznati jezik zlonamerne programske opreme. Zaradi razlike med programskim jezikom in človeškim jezikom so prilagodili arhitekturo nevronske mreže za dani problem. Tako sama arhitektura mreže uporablja elemente, ki učinkovito zaznavajo vzorce, tudi če so ti različno razvrščeni, s pomočjo oken za združevanje maksimalnih vrednosti (*angl. max-pooling*). V kodi, bolj pogosto kot v naravnem jeziku, opazimo primere zelo povezanih besed, ki so lokacijsko med seboj zelo oddaljene, kar so v študiji naslovili tako, da je model sposoben dolgoročnega spomina s pomočjo prepustnih enot (*angl. leaky-units*) in uporabo dvosmerne arhitekture modela (*angl. Bi-Directional model*). Te vrste mrež so se v študiji izkazale za precej uspešne z resnično pozitivno stopnjo 98,3% in lažno pozitivno stopnjo 0,1% [7].

Zlonamerno programsko opremo pa lahko prepoznamo tudi brez same kode, ki se izvaja. Prenosno izvedljivi format (*angl. Portable Executable format oz. PE*) je datotečni format, ki nosi kodo, dinamične knjižnice in ostale podatke potrebne za izvedbo programa na računalniku. Samo z zglavnim delom formata in ostalimi značilkami izven same kode programa pa lahko izvemo s precej veliko verjetnostjo, če je program zlonameren. Primer koristnih informacij, ki jih lahko najdemo so uporabljene dinamične knjižnice (končnica .dll), kot sta na primer WINSOCK.DLL in WSOCK.DLL, ki opravljata aktivnosti povezane z mrežo. Čeprav jih najdemo tudi v neškodljivih programih, se te knjižnice pogosto pojavljajo v različnih virusih, ki poskušajo dostopati do virov na mreži. Z odločitvenimi drevesi, ki so se učili na podatkovni zbirki pridobljenih značilk, je v okviru študije iz leta 2009 [10] uspelo z 99% točnostjo predvideti zlonamerno programsko opremo.

Zlonamerno programsko opremo pa lahko zaznamo tudi na podlagi njene zbirne kode. V študiji leta 2019 [4] so poskušali razstaviti izvedljive programe v zbirno kodo in grupirati ukaze glede na njihovo funkcionalnost. Produkt pred- procesiranja programa je torej sekvenca značilk, ki predstavljajo matriko na podlagi katere se konvolucijska nevronska mreža uči. Model se je sicer izkazal za malo manj uspešnega, z 95% točnostjo detekcije, vendar je precej hitrejši pri ekstrakciji značilk in učenju.

Ena izmed naivnih metod za detekcijo je uporaba surovih bitov izvedljive kode in pustiti modelu, da sam izlušči zanj pomembne značilke. Skupini znanstvenikom je leta 2017 [8] uspelo sestaviti model z več kot 90% točnostjo odkrivanja zlonamerne programske opreme. Kasneje istega leta [9] jim je uspelo izboljšati rezultat z uporabo surovih bitov samo iz glave PE. Obstajajo torej uspešne metode prepoznavanja zlonamerne programske opreme z minimalnim poznavanjem domene, hkrati pa način s surovimi biti predstavlja manjše računsko breme, kot na primer z uporabo n-gramov.

3. MODELNA ARHITEKTURA

Za doseganje večje natančnosti in zadoščanjem različnim scenarijem med digitalno preiskavo želimo, da zastavljeni model vključuje značilnosti fragmentov zlonamernih programskih oprem. To dosežemo na naslednji način:

1. Cilj odkrivanja so manjši fragmenti zlonamerne programske opreme, torej mora model imeti boljšo univerzalnost in lahko zazna fragmente poljubnih dolžin.

2. Ker je binarni del zlonamerne kode redko časovno zaporedje, morajo biti razmerja med vsakima bajtoma učinkovito obdelana.

3.1 Pridobitev Baze Podatkov

Za bazo zlonamerne programske opreme potrebujemo tako opremo, ki je noben antivirusni program ne more učinkovito zaznati. Najprej naložimo antivirusna orodja, ki jih ne posodabljamo. Hkrati pridobimo zlonamerno programsko opremo, ki je bila zaznana pred mesecem, v katerem smo naložili antivirusna orodja. Čez nekaj mesecev ponovno pridobimo zlonamerno programsko opremo, kjer ne vzamemo zlonamerne programske opreme, ki je bila zaznana predno smo naložili antivirusno orodje in jih dodamo v bazo kot vzorce za usposabljanje.

3.2 Model Izbire

V splošnem metode, ki temeljijo na RNN boljše rešujejo probleme, ki vsebujejo časovne vrste. Pogosto v običajnih besedilih časovnih vrst vsaka beseda nosi svoj pomen in RNN lahko izve pomen povedi skozi asociacije različnih besed. To pomeni, da izvaja procese kot klasifikacije in napovedi. Za razliko od normalne časovne vrste je numerična porazdelitev dvojiške zlonamerne kode razmeroma redka. Ampak RNN ne deluje najboljše, ko ji podamo preveč redek vhod. To je prvi razlog, zakaj se je ne uporablja. Zaradi strukture RNN metode večja kot je dolžina zaporedja večja je časovna in prostorska zahtevnost za treniranje. Zlonamerno dvojiško zaporedje ima pogosto vhodno zaporedje dolžine več milijonov bajtov. Kljub temu, da je cilj zaznati majhne zlonamerne fragmente, lahko dobimo fragmente z veliko količino bajtov. Če želimo zadostiti vsestranskosti modela in karakteristiki ujemanja binarne redkosti, vzamemo na CNN bazirano metodo za treniranje modelov.

Če želimo zaznati daljše segmente, ki so bližje PE datotekam, mora biti upoštevana celostna karakteristika PE datotek. V PE datotekah veliko delov lahko premaknemo ne da bi to vplivalo na njihovo izvršljivost. Na primer PE-header hrani kazalnike na ostale vsebine, medtem ko je njegova lokacija poljubna in shranjena v konstanti MS-DOS Header. Ta prostorska lastnost ni enostavno rešljiva z uporabo trenutnih metod. Kljub temu je translacijska invarianca CNN-ja boljši način reševanja tega problema.

3.3 Nastavljanje Parametrov

Karen Simonyan in Andrew Zisserman sta leta 2015 v članku objavila VGG strukturo [11]. Ta pravi, da je za dano receptivno polje uporaba zloženih majhnih konvolucijskih jeder boljša od uporabe velikih konvolucijskih jeder, ker lahko več nelinearnih plasti poveča globino omrežja. To zagotavlja, da se bolj zapletene vzorce naučimo z nižjimi stroški (manj parametrov). Torej z uporabo manjših jeder in relativno večjim številom plasti kot pri prejšnji metodi z globokim učenjem. Na začetku se nastavi jedro na velikost 3, ker je to najmanjša velikost, ki zajame levi, desni in sredinski koncept. Med eksperimentiranjem so ugotovili, da je model bolj nagnjen k preobremenitvi, še posebej, če je fragment, ki so ga izločili, zelo kratek. Torej s povečavo jeder v zadnji plasti na velikost 5, tako da neprekinjeno izvajamo primerjalne eksperimente in dodajamo izpade. Pri večini eksperimentov so izpadi dodani popolnoma povezanim plastem. Med eksperimentiranjem so ugotovili, da so taka dodajanja imela le majhen vpliv na rezultat. Prostorski izpad je izpadna metoda, ki jo je predlagal Tompson na področju slik leta 2015. Navaden izpad naključno nastavi neke elemente na 0, medtem, ko bo prostorski izpad naključno nastavil del območja na 0. Zadnja metoda se je izkazala za učinkovito na področju prepoznavanja slik. Pri poskusu metode prostorski izpad in dodajanju le-te za konvolucijsko plastjo so učinkovito izboljšali natančnost in stabilnost modela. V zadnji plasti so izbrali sigmoidovo funkcijo namesto softmax, ker ta zavzema vrednosti med 0 in 1, da rešuje problem odkrivanja nesosednjih neurejenih fragmentov zlonamerne programske opreme.

4. MODEL IZBIRE

4.1 Baza Podatkov

Treniranje podatkov z metodami globokega učenja pogosto zahteva veliko število pozitivnih in negativnih vzorcev. Ne obstaja standardna baza, kar je tudi razlog, da različna antivirusna podjetja ali inštituti ne morejo doseči enotnosti pri območju zaznavanja zlonamerne programske opreme. Da bi zagotovili vsestranskost eksperimenta, za pozitivne vzorce uporabimo znano bazo vzorcev zlonamerne programske opreme VirusShare. Negativne vzorce so naložili iz Microsoftove spletne trgovine, kot so igrice, glasba, socialna omrežja ipd. Da bi zagotovili uravnoteženost pozitivnih in negativnih vzorcev, so jih pregledali in jih dosledno razdelili po velikosti. Na koncu so dobili 5241 pozitivnih vzorcev in 5211 negativnih vzorcev. Nato so z 1 označili pozitivne vzorce in z 0 negativne. Kot rečeno v poglavju 3.1. so pridobili tudi 10000 vzorcev zlonamerne programske opreme, ki so jih dodali med 20 primerov antivirusnih orodij. Na koncu so dobili 107 ničtih vzorcev zlonamerne programske opreme. Povprečna velikost vzorcev je 769.47KB.

4.2 Postopek Eksperimentiranja

Najprej moramo pridobiti binarne vrednosti vseh vzorcev. Nato z dvema različnima metodama, ki so ju implementirali za članek, vzorčimo. Prvi je nenehno vzorčenje, pri tem razdelku raziskujemo tudi razmerje med zaznanim rezultatom in dolžino oziroma pozicijo neprekinjenih fragmentov. Drugi način je naključno vzorčenje PE datotek, da ugotovimo učinek nekontinuirane dolžine vzorca in vrstnega reda vzorčenja na odkrivanje rezultata. Z analiziranjem eksperimentalnih rezultatov poskusimo pridobiti nekaj modelov pri različnih dolžinah. Rezultati analize so zapisani v poglavju 5.

4.3 Metode Vzorčenja

4.3.1 Nenehno Vzorčenje

Cilj tega dela je simuliranje situacije, kjer lahko pridobimo neprekinjen del PE datoteke, ker je le del datoteke uničen. Za treniranje naključno ekstrahiramo različne dele PE datoteke za vsako dolžino, ki jo želimo primerjati. Ker želimo imeti največ vzorcev, kot jih lahko imamo, upamo, da lahko nekatere dolge datoteke vzorčimo večkrat ne da bi jih dali nazaj. Prav tako, če želimo zagotoviti naključnost vzorčenja, ko potrebujemo, da je neka datoteka večkrat vzorčena, upamo da je nevzorčen del datoteke veliko večji od ciljne dolžine. Zato mora število vzorčenj za vsako datoteko ustrezati formuli $slice\_num = [file\_len/frag\_len/k]$, kjer $file\_len$ predstavlja dolžino datoteke, $frag\_len$ prestavlja dolžino vsakega fragmenta, ki ga želimo trenirati, $slice\_num$ je končno število fragmentov, ki jih pridobimo iz datoteke in k je konstanta nastavljena na 50. Po vzorčenju premešamo vse fragmente, da oblikujemo našo bazo podatkov. Dodatno še poskusimo ugotoviti učinek pozicije zaporednih fragmentov na eksperimentalni rezultat.

4.3.2 Nekontinuirano Vzorčenje

Obstaja situacija, kjer ne obstaja dolg neprekinjen fragment, ampak več majhnih fragmentov PE datotek. Zasnovali smo dva scenarija, prvi je urejen in drugi je neurejen. Urejen scenarij pomeni, da po vzorčenju vsake PE datoteke je vrstni red končnega vzorčenja konsistenten z relativno ureditvijo

fragmentov v prvotni datoteki. Nasprotno, neurejen scenarij bo motil vrstni res vzorcev. Razlog za to je, da želimo simulirati situacijo, da kar dobimo ni nujno urejeno.

Končno za eksperiment izberemo majhne fragmente velike 1024 bajtov. Dedupliciramo podatke zbrane pri zgornjih dveh primerih, da bi se izognili vplivu na rezultate testov. Razlog za to, da smo izbrali 1024 bajtov bomo pokazali v poglavju 5.

5. REZULTATI

V sledečem razdelku predstavimo rezultate testiranja modela, ki so jih pripravili v članku [14]. S testom so se poskušali čimbolj približati realnemu primeru, kjer nimamo na voljo celotnega zlonamernega programa. Bodisi je zlonamerni program že izbrisal svojo kodo, bodisi je na računalniku le delno naložen, in mora preostanek svoje kode prenesti z interneta. Prenos kode preko interneta poteka po kosih, ki lahko na računalnik prihajajo v pomešanem vrstnem redu. Protivirusni program te kose prestreza, vendar mu lahko kakšen kos vmes uide. S serijo različnih testov so v članku ugotavljali, kako se model obnaša v različnih scenarijih.

5.1 Obnašanje modela na neprekinjenem kosu datoteke

Ta scenarij predpostavlja, da je protivirusni program uspel pridobiti en neprekinjen kos kode. Zanima nas, kako uspešno model zazna zlonamerno programsko opremo glede na dolžino pridobljenega kosa in kje v datoteki se ta kos nahaja. Avtorji so model preizkusili na kosih dolžine 32 bajtov pa vse do 200 kilobajtov.

Tabela 1 prikazuje uspešnost modela pri ugotavljanju ali je dan program virus ali ne, ko so mu bili podani neprekinjeni kosi izvršljivih datotek.

Najprej osredotočimo na najbolj realen scenarij, tj. kos se nahaja na naključni lokaciji v datoteki. Razvidno je, da z velikostjo kosa narašča tudi točnost modela, dokler ne doseže vrhunca pri 60KB, nato točnost pada. Torej je 60KB optimalna velikost paketa. Vendar pa imamo zaradi dane predpostavke, da se kosi prenašajo po internetu, dodatno omejitev. Običajno je velikost internetnega paketa 1500B, zato nas zanimajo velikosti manjše od 1500B. S tem v mislih se model najboljše odreže pri 1024 bajtih z 76,75% točnostjo.

Zanimivi so tudi rezultati pri 32 bajtov velikih kosih. Kljub majhnim številom bajtov zna model z 64,39% točnostjo ločiti med virusi in neškodljivimi programi. Če predpostavljamo, da je PE datoteka sestavljena večinoma iz zbirnih ukazov in je program napisan za **x86** arhitekturo, kjer je povprečni ukaz dolg približno 4 bajte [5], je v takem kosu kode približno 8 zbirnih ukazov. Ročno ugotavljanje ali 8 ukazov predstavlja zlonamerno kodo je skoraj nemogoče.

Iz tabele lahko vidimo tudi, da model v večini primerov z večjo točnostjo zna identificirati virus, če se kos nahaja na začetku datoteke. Razlog je najverjetneje v tem, da se na začetku datoteke nahaja zaglavje iz katerega znajo klasični protivirusni programi, ki uporabljajo statično analizo, z visoko točnostjo zaznati virus.

	Lokacija kosa		
Velikost kosa [B]	Naključna [%]	Konec [%]	Začetek [%]
32	64,39	67,82	62,96
128	67,85	75,68	66,28
512	70,65	80,02	71,36
1024	76,75	82,84	72,09
4096	78,59	83,63	73,31
10K	80,78	86,75	76,18
30K	80,13	87,99	80,71
60K	86,92	85,67	77,07
100K	82,79	85,98	81,33
200K	79,47	82,29	78,73

Table 1: Rezultati modela na neprekinjenem kosu kode različnih dolžin in na različnih lokacijah v datoteki.

	Vrstni red kosov	
Skupna velikost kosov[B]	Naključen [%]	Urejen [%]
2*1024	78,29	81,17
3*1024	79,48	83,12
4*1024	83,14	82,71
5*1024	81,66	82,68
6*1024	81,60	85,27
7*1024	83,59	82,88

Table 2: Rezultati modela na več manjših kosih v naključnem ali urejenem vrstnem redu.

5.2 Obnašanje modela na več manjših kosih datoteke

Ta scenarij predpostavlja, da je protivirusni program uspel zajeti več kosov kode, med katerimi kakšen kos manjka, lahko pa so tudi v pomešanem vrstnem redu. Podobno kot v razdelku 5.1 imamo omejitev, največjo velikost paketa 1500B. Velikost enega kosa naj bo 1024B.

Ker imamo več kosov z različnih lokacij v datoteki, se nam pojavi problem, kako n kosov kot celoto podamo modelu. Tukaj avtorji predlagajo dve rešitvi. Prva je, da kose enostavno staknemo skupaj. Druga pa da vsak kos posebej podamo modelu in izračunamo neko uteženo povprečje. Utež so avtorji določili empirično. Pri kosih, ki so v urejenem vrstnem redu, sta obe rešitve enakovredni. Razlika se pojavi pri kosih, ki so v pomešanem vrstnem redu, kjer se boljše izkaže druga rešitev.

Iz tabele 2 je razvidno, da ima model v večini višjo točnost pri urejenih kosih datoteke. Pri urejenih kosih datoteke se z večanjem števila kosov točnost modela ne spreminja veliko. Pri naključno urejenih kosih lahko vidimo minimalno zvišanje točnosti do velikosti 4*1024B, nato ostaja približno enaka.

Če primerjamo točnost pri neprekinjenem kosu in pri več kosih pri isti velikosti (4096B), vidimo da se model boljše obnese pri neprekinjenih kosih. Razlog za to bi lahko bil v tem, da z vzorčenjem več manjših kosov dobimo bolj reprezentativno oz. bolj splošno sliko datoteke, kot če vzamemo le en večji kos.

Velikost kosa [B]	Avtorjev model [%]	LSTM [%]	CNN[%]	DNN[%]	HMM[%]
32	64,39	60,47	57,89	59,28	60,32
128	67,85	68,16	64,01	66,51	66,76
512	70,65	69,87	68,29	68,09	65,01
1024	76,75	71,62	70,34	69,34	68,27
4096	78,59	73,59	73,01	70,26	69,56
10K	80,78	79,48	77,95	74,29	72,84
30K	80,13	75,61	75,37	72,54	71,92
60K	86,92	76,69	75,20	70,43	69,34
100K	82,79	72,77	73,94	72,21	73,69
200K	79,47	70,56	69,42	76,23	74,07
>200K	77,78	71,32	68,30	74,06	72,58

Table 3: Rezultati modela na neprekinjenem kosu kode različnih dolžin v primerjavi z drugimi modeli strojnega učenja.

Skupna velikost kosov[B]	Avtorjev model [%]	LSTM [%]	CNN [%]
2*1024	78,29	78,69	78,84
3*1024	79,48	78,17	78,80
4*1024	83,14	80,61	82,71
5*1024	81,66	75,57	80,04
6*1024	81,60	74,47	74,68
7*1024	83,59	71,24	71,08

Table 4: Rezultati različnih modelov na več manjših kosih v naključnem vrstnem redu.

5.3 Primerjava z nekaterimi drugimi modeli strojnega učenja

V sklopu članka so bili narejeni tudi eksperimenti zaznavanja virusov z drugimi metodami strojnega učenja. Avtorji so naučili in optimizirali modele konvolucijske nevronske mreže (CNN), Long-Short Term Memory(LSTM), Globoke nevronske mreže (DNN) in skrite modele Markova (HMM).

Tabela 3 prikazuje rezultate različnih modelov strojnega učenja na neprekinjenih kosih. Model avtorjev je v večini najbolj točen izmed vseh predstavljenih. Lahko vidimo tudi, da avtorjev model, LSTM in CNN dosežejo vrhunec pri srednje velikih kosih (60KB in 10KB), nato pa točnost pada. Natančnosti DNN in HMM modelov pa do srednje velikih kosov narašča, nato pa stagnira.

V tabeli 4 so prikazani rezultati avtorjevega modela, LSTM in CNN na več manjših kosih v naključnem vrstnem redu. Vidimo, da pri avtorjevem modelu točnost narašča do velikosti 4*1024B, nato stagnira. Modela LSTM in CNN pa pri 4*1024B dosežeta vrhunec, nato točnost začne padati. Avtorji niso podrobno opisali implementacij ostalih modelov strojnega učenja. Vendar menimo, da je vzrok za padec točnosti pri večjih velikostih v načinu združevanja kosov v celoto, kot opisano v razdelku 5.2. Pri lastnem modelu avtorji v model posredujejo posamezne kose, nato izračunajo uteženo vsoto. Pri ostalih dveh modelih pa menimo, da enostavno združijo vse kose v enega in ga podajo v model.

6. ZAKLJUČEK

V članku smo predstavili področje zlonamerne programske opreme. Opisali smo, kaj je ranljivost ničtega dne, kateri so že uveljavljeni prijemi za detekcijo in kako metode strojnega učenja pomagajo pri detekciji. Predstavili smo model strojnega učenja za detekcijo tovrstnih virusov, ki so ga zasnovali v članku [14]. Predstavili smo, kako uspešno model zazna virus v situacijah, ki poskušajo simulirati realne scenarije. Na koncu predstavimo še kako se rezultati modela primerjajo z rezultati nekaterih drugih modelov strojnega učenja.

V opravljenih testih se model izkaže kot najbolj točen v večini primerov v primerjavi z ostalimi modeli strojnega učenja. Na tem mestu pa si moramo postaviti vprašanje, če bi lahko ostale predstavljene modele strojnega učenja optimizirali še bolj. Kakšni bi bili v tem primeru rezultati? Bi lahko zasnovali še kakšen drugačen realen scenarij, na katerem se model morda ne bi odrezal tako dobro?

Metode strojnega učenja na področju zaznave virusov so poleg dinamične analize ključni pri zaznavanju izkoriščevanja ranljivosti ničtega dne. Pričakujemo, da bo njihova uporaba na tem področju le rastla. Čeprav se tradicionalno in v praksi še vedno pogosto uporabljajo načini, na primer metoda s preverjanjem podpisa, ki so preprosti, zelo hitri in precej učinkoviti, so velikokrat neprimerni, še posebej, ko pride do napadov ničtega dne. Ravno tu pa se pokaže potencial nevronskih mrež, ki so kljub kompleksnosti in relativne počasnosti izvajanja še vedno precej učinkovite pri reševanju te problematike.

7. REFERENCES

[1] U. Bayer, C. Kruegel, and E. Kirda. Ttanalyze: A tool for analyzing malware. 01 2006.

[2] I. Firdausi, C. lim, A. Erwin, and A. S. Nugroho. Analysis of machine learning techniques used in behavior-based malware detection. In *2010 Second International Conference on Advances in Computing, Control, and Telecommunication Technologies*, pages 201–203, 2010.

[3] E. Gandotra, D. Bansal, and S. Sofat. Zero-day malware detection. In *2016 Sixth International*

Symposium on Embedded Computing and System Design (ISED), pages 171–175, 2016.

[4] Z. Kan, H. Wang, G. Xu, Y. Guo, and X. Chen. Towards light-weight deep learning based malware detection. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, volume 01, pages 600–609, 2018.

[5] B. Lopes, R. Auler, R. Azevedo, and E. Borin. Isa aging: An x86 case study. 07 2013.

[6] J. Metzman, L. Szekeres, L. M. R. Simon, R. T. Sprabery, and A. Arya. Fuzzbench: An open fuzzer benchmarking platform and service. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, New York, NY, USA, 2021.

[7] R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas. Malware classification with recurrent networks. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1916–1920, 2015.

[8] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. Nicholas. Malware detection by eating a whole exe. 10 2017.

[9] E. Raff, J. Sylvester, and C. Nicholas. *Learning the PE Header, Malware Detection with Minimal Domain Knowledge*, page 121–132. Association for Computing Machinery, New York, NY, USA, 2017.

[10] M. Z. Shafiq, S. M. Tabish, F. Mirza, and M. Farooq. Pe-miner: Mining structural information to detect malicious executables in realtime. In E. Kirda, S. Jha, and D. Balzarotti, editors, *Recent Advances in Intrusion Detection*, pages 121–141, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[11] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 09 2014.

[12] D. K. Vaisla. Analyzing of zero day attack and its identification techniques. *Proceedings of First International Conference on Advances in Computing Communication Engineering (ICACCE-2014)*, 1:11–13, 02 2014.

[13] T. Walshe and A. Simpson. An empirical study of bug bounty programs. pages 35–44, 02 2020.

[14] Q. Wen and K. Chow. Cnn based zero-day malware detection using small binary segments. *Forensic Science International: Digital Investigation*, 38:301128, 2021.

4 - Digitalna forenzika slik in zvoka / Digital forensics of images and sound

Media Forensics and DeepFakes: An Overview

Martin Blažek
Faculty of Computer and Information Science
Večna pot 113
Ljubljana, Slovenia
mb0928@student.uni-lj.si

Izabela Lesac
Faculty of Computer and Information Science
Večna pot 113
Ljubljana, Slovenia
il5623@student.uni-lj.si

ABSTRACT

In recent years, techniques of generating and manipulating visual media have developed to a point where forged images are often indistinguishable from real ones. There is significant potential for abuse and conventional methods of detecting manipulation are no longer powerful enough. For that reason, more advanced methods based on deep learning have been developed to tackle the problem of detecting deepfakes.

Keywords

media forensics, deepfakes, deep learning, image analysis

1. INTRODUCTION

Image manipulation is a phenomenon that has accompanied photography and other types of visual media since their inception. It can be used for legitimate purposes such as entertainment. However, malicious actors may also use image manipulation to fabricate evidence, blackmail people, etc.

Following the advent of digital image editing software, prevalence of image manipulation increased substantially. Various methods have been developed to detect traces of forgery with a good success rate.

The recent rapid development of deep-learning-based image manipulation methods has brought new challenges upon the media forensics field. When faced with deepfakes (visual media altered using deep learning), conventional methods struggle with detecting falsification. To keep up with the state of the art of image manipulation, media forensics experts have invented new detection methods that are also based on deep learning.

Chapter 2 provides the background knowledge needed for understanding deepfake detection by giving an overview of the conventional methods. Chapter 3 describes the modern deep-learning-based methods. Finally, Chapter 4 discusses deepfake detection in practice.

2. CONVENTIONAL METHODS

Conventional methods of image manipulation detection rely on information inherent to the in-camera processing chain (e.g., camera artifacts) and the subsequent out-camera processing chain (e.g., image processing). Potential irregularities and deviations from the norm in the data can serve as evidence of image tampering.

2.1 Blind Methods

These detection methods do not require any prior knowledge about the image in question, hence their name – blind methods. They rely exclusively on the image data.

Lens Distortion. Each camera contains a set of optical apparatus that causes various amounts of lens distortion, unique for a given kind of camera. It can be utilized forensically by exploiting chromatic aberrations, radial distortions of wide-angle lenses [13], etc.

CFA Artifacts. Most digital cameras utilize a color filter array (CFA), which allows sensors to only record three wavelengths (red, green, blue). To obtain the full color image, a demosaicing algorithm is used to interpolate the missing color information. This process introduces an underlying periodic pattern to the image data, which can be exploited through Fourier analysis to detect spliced regions in an image and expose computer-generated images [23].

Noise Residual Artifacts. Analysis of the noise residual of an image allows for the discovery of possible manipulations. In addition to comparing noise levels throughout an image, we can utilize a more complex method of finding deviations in noise patterns [26]. Both methods may also be used to detect inter-frame manipulations in videos.

Compression Artifacts. The exploitation of compression artifacts has been used in digital image forensics for a long time. The paper focuses on artifacts associated with the JPEG compression algorithm. An early popular approach is to detect discontinuities in the block artifact grid that is present due to JPEG's block-wise compression process [11]. Discrepancies in the grid typically indicate splicing or copy-move manipulations. The presence of double compression artifacts is another hint of image tampering. Some methods focus on exploiting the subtle differences between various implementations of the JPEG standard [3].

(a) Manipulated photo (b) Detected manipulations (c) Suggested "undo" (d) Original photo

Figure 1: The process of detecting manipulation, and an attempt to return the image to the original state [30].

Editing Artifacts. The editing of an image often leaves behind a set of traces which can be utilized in image forensics. Duplicating or moving objects in images is a very common manipulation. The presence of identical regions is strong evidence of tampering. Clones are sometimes disguised using rotation, scaling or geometric distortion to prevent detection. However, resampling is always performed when scaling or rotating an object, which produces detectable periodic artifacts [24].

2.2 One-Class Methods

Due to manufacturing imperfections, different cameras produce photographs with slightly different noise profiles, called photo-response non-uniformity (PRNU) noise. To detect forgery using PRNU noise, two steps need to be performed [17]. First, the camera's noise profile is extracted from a large number of photographs taken using that camera. Second, an estimation of the target image's PRNU is obtained. We can then calculate the level of similarity between the image's PRNU and the reference noise model. The described method is extremely powerful as it can detect all kinds of image forgery. However, the PRNU of an individual camera is a relatively weak signal. To tackle this problem, an alternative method is proposed in [28] in which a reference noise profile is obtained from many cameras of the same model instead of one individual camera.

2.3 Supervised Methods

These methods are based on machine learning of specific handcrafted features of an image. They belong to the conventional class of methods because they rely on a thorough technical understanding of a specific image feature (e.g., double compression artifacts, noise residual artifacts, etc.).

2.4 Shortcomings

In ideal conditions, the conventional methods described work equally well as deep-learning based methods. However, when compression and resizing are introduced, the accuracy of conventional methods falls below that of deep-learning based methods [27]. As both compression and resizing are used routinely in real-life scenarios, it is desirable to use deep-learning based methods instead of conventional ones.

3. DEEP LEARNING BASED METHODS

Deep learning methods have been implemented in a number of different fields, with great success. In recent years, we have seen them applied in the multimedia forensics field as well. Which architectures and techniques will be the most useful depends on the type of manipulation. Convolutional neural networks (CNNs) have been proposed for generic image and video manipulations. A CNN is essentially a deep learning algorithm which takes input, assigns importance to different elements of the input and can then differentiate them from each other. This paper presents three different techniques based on CNNs and supervised learning: CNN looking for specific clues, generic CNN and one-class training. On the other hand, GAN-generated media require a different approach which is discussed in the later chapterA GAN is a neural network made up of two parts - a generator and discriminator. Generator learns to generate data while discriminator learns to distinguish generated content from the real content. The discriminator "rates" how plausible generator's content is.s.

3.1 Searching for Specific Artifacts Using Supervised CNNs

3.1.1 Double Compression

CNNs can be used to detect traces of double compression in images and videos. These methods rely on discrete cosine transform (DCT) histograms, features and noise residuals. Q. Wang and R. Zhang proposed an algorithm capable of detection and localization by using DCT coefficients as input. DCT presents an image as a sum of sinusoids. It is used in compression of JPEG images. An input image is divided into 8x8 blocks and the DCT is computed for each block. After double JPEG compression, histograms are missing values and no longer follow a Gaussian distribution. A CNN was designed to learn features of these histograms and classify them by single and double compression. The output is a probability pair, showing how likely it is that a block has

been doubly compressed [29]. Park et al propose a method that can handle mixed JPEG quality, which is more suitable for real life application [22]. When it comes to videos, authors in [5] analyze the intra-coding process to detect traces of double compression within frames.

3.1.2 Composition, Copy-Move and Inpainting

Composition, copy-move and inpainting are some of the most common forgeries. Their detection by CNNs simulates the steps of conventional methods, but there are clear advantages. Learning based approaches are capable of performing these steps on a larger scale, more easily adapt to different scenarios and have better performance on low quality images than traditional techniques. Furthermore, CNN models can even be designed to detect specific artifacts left by individual tools in Photoshop, demonstrated in [30] (see Figure 1).

3.1.3 Generic CNN Detectors

In contrast to the previous solutions, these algorithms do not look for specific artifacts. They are based on filtering, feature extraction and noise residuals. Some authors implement an SVM for decision making [25], while others rely solely on CNNs. Zhou et al implement a two stream network, combining the detection of visual manipulation artifacts and irregularities in noise features [6], significantly improving performance compared to other modern methods. Still, limited resources and computational power means deep learning models can only accept small images as input. For this reason images are usually resized, which can destroy evidence of manipulation. A possible solution is presented in [20], with the use of gradient checkpointing and joint optimization.

3.2 One-Class Training

This approach attempts to minimize the training set and detect all types of manipulations. This can be done by extraction of expressive features and iterative feature labeling [10]. A different method includes the use of generative adversarial networks in the learning phase [32]. Some methods use camera model dependent features, metadata and even noiseprint in order to detect differences between pristine and spliced areas of an image.

4. DEEPFAKE DETECTION

Detection of deepfakes depends on which method was used to generate them. This paper describes techniques for fake images created by GANs and video-face manipulations. A GAN is a neural network made up of two parts - a generator and a discriminator. The generator learns to generate data while the discriminator learns to distinguish generated content from the real content. The discriminator "rates" how plausible the generator's content is.

4.1 GAN-Generated Images

Fake images can have artifacts seen by the naked eye. Some examples include different eye color, different ears, badly modeled teeth or asymmetry (see Figure 2). Artifacts can also come in the form of color disparities or underexposed pixels. A different method involves GAN specific fingerprints. The work of Marra et al. [19] suggests every GAN leaves a unique fingerprint in the image it created, much like traditional cameras. This is a fast developing branch of research. Authors in [4] have tried to reverse the generation process using a whitebox scenario, while [14] managed to correctly attribute high quality generated images to their network. However, certain issues exist. Performance is decreased when using compressed images and most of the current models overfit the training set. New solutions should strive for good generalization through the use of multiple sets of data and varied types of image alterations. There are a couple of possible solutions currently: use of an autoencoder-based architecture, deeper networks and incremental learning. It is also possible to improve results by removing low level artifacts via pre- and post-processing steps.

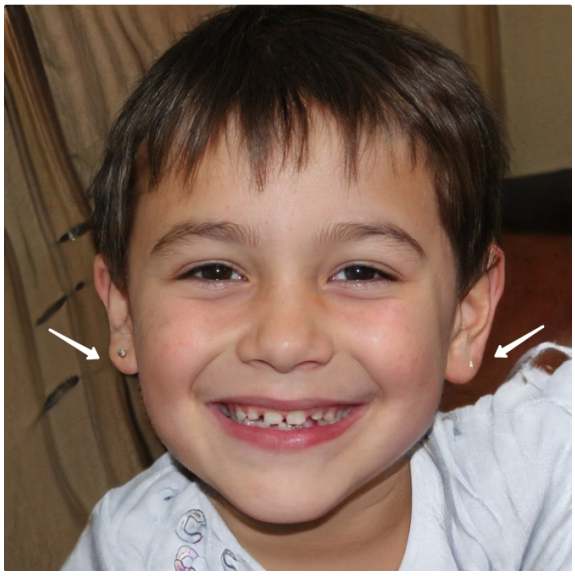


Figure 2: GANs often struggle with generating earrings [1].

4.2 Video-Face Manipulation

Because of the value of non verbal communication and identity, recent developments in AI powered face manipulation tools have caused great concern. However, older research on detecting CGI created faces helps us to build a framework for deep fake detection. Currently there are two main strategies being implemented in the field: handcrafted solutions and deep learning solutions. We will first review the information on handcrafted solutions.

4.2.1 Handcrafted Solutions

Since GANs produce visible artifacts, it is possible to use these errors as means of determining if a sequence has been manipulated. Normal biological functions produce miniscule variations, which are hard to replicate. For example, one method found proposed in [16] uses eye blinking patterns and frequency. It is also possible to use heartbeats, as suggested in [8] and [12]. Another strategy relies on detecting traces of warping, as all generated faces need to be altered to match the source video. This can be done using inconsistent head poses [31]. In the future, these visual imperfections may become obsolete and the proposed handcrafted solutions won't be useful.

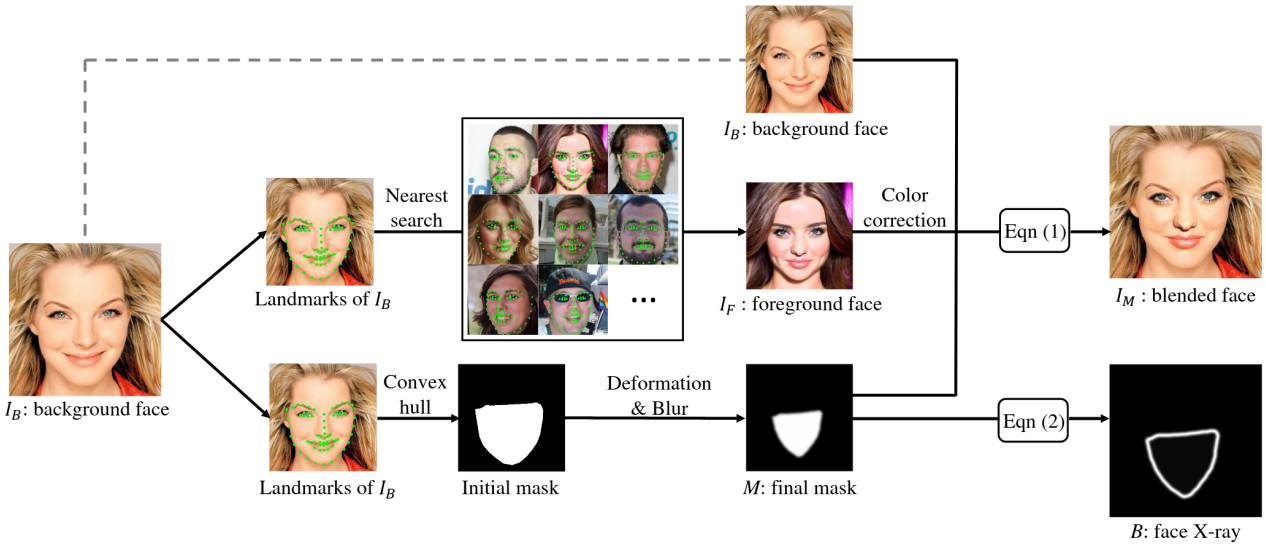


Figure 3: Overview of generating a training sample using Face X-ray [15].

4.2.2 *Deep Learning Solutions*

There are two options when implementing learning networks for detection of deepfakes. One is the use of simple architectures with a small number of levels. Afchar et al [2] created Meso-4 with only four layers of convolution and pooling. The other option is to use very deep networks. The latter has proven to be more effective for videos than shallow CNNs and handcrafted solutions. To optimize generalization, robustness and adaptability to new forms of manipulation novel techniques are emerging. One such is presented in the work of Li et al. It uses a completely new image representation model called face X-ray. This method relies on the fact that most current ways of creating a fake image contain a blending step, to help merge the fake parts of an image into the real background. Face X-ray reveals the blending boundary, which should not be present in pristine images. The pros of this method are immense, as it does not rely on finding specific artifacts. Furthermore, the training phase can be done without fake images, significantly improving detection, as shown in Figure 3.

4.3 Issues and Counterforensics

Another important aspect of the field right now is keeping up with attackers and remedying shortcomings of previously described methods. Conventional detection strategies can easily be overcome, as it is possible to remove and alter camera device fingerprints. Deep learning methods are proving to be more robust in this regard, giving investigators a new way to fight back. They are not without issues however. First and foremost is the lack of adequate datasets that provide enough useful information. This is especially true for videos. Similarly, in order for alterations to be recognized, they need to be represented in the training set. This is obviously not possible with new forms of GAN generated media. One-class methods provide a possible solution but then we have the problem of detecting normal image editing operations as malicious. Furthermore, it is possible to fool a CNN into wrongly classifying an image simply by injecting the target image with a noise pattern [7]. One defense CNNs have against these types of attacks is compression. Authors of [18] proved strong lossy compression massively reduces strengths of noise attacks. But more complex attacks have come to experts' attention in recent years. Neves et al [21] describe a method which can entirely remove GAN fingerprints from an image, while in [9] that is combined with injecting fake traces of real cameras to further fool classifiers. The result is a GAN generated image which is no longer recognized as such.

5. CONCLUSION

The importance of media forensics is rapidly increasing due to the quick development of deepfake technology and its potential for abuse. Conventional approaches to detecting image manipulation are no longer effective enough. New detection methods based on deep learning are necessary to keep up with the pace of progress. These new methods often build upon the conventional approaches while broadening their scope of utility. The various methods differ in the amount of prior knowledge required to use them, their effectiveness with regard to different kinds of images, etc. To improve the chances of successfully detecting forgery, it is essential to identify the right method for a particular image or video. There is a constant tug-of-war between new forgery detection methods and refinement of concealment practices

6. REFERENCES

[1] Random face generator (this person does not exist).

[2] D. Afchar, V. Nozick, J. Yamagishi, and I. Echizen. Mesonet: a compact facial video forgery detection network. *CoRR*, abs/1809.00888, 2018.

[3] S. Agarwal and H. Farid. Photo forensics from jpeg dimples. In *2017 IEEE Workshop on Information Forensics and Security (WIFS)*, pages 1–6, 2017.

[4] M. Albright and S. McCloskey. Source generator attribution via inversion. *CoRR*, abs/1905.02259, 2019.

[5] M. Barni, L. Bondi, N. Bonettini, P. Bestagini, A. Costanzo, M. Maggini, B. Tondi, and S. Tubaro. Aligned and non-aligned double jpeg detection using convolutional neural networks. *J. Vis. Comun. Image Represent.*, 49(C):153–163, nov 2017.

[6] B. Bayar and M. C. Stamm. A deep learning approach to universal image manipulation detection using a new convolutional layer. In *Proceedings of the 4th ACM Workshop on Information Hiding and Multimedia Security*, IHamp;MMSec '16, page 5–10, New York, NY, USA, 2016. Association for Computing Machinery.

[7] B. Biggio and F. Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331, 2018.

[8] U. A. Ciftci and I. Demir. Fakecatcher: Detection of synthetic portrait videos using biological signals. *CoRR*, abs/1901.02212, 2019.

[9] D. Cozzolino, J. Thies, A. Rössler, M. Nießner, and L. Verdoliva. Spoc: Spoofing camera fingerprints. *CoRR*, abs/1911.12069, 2019.

[10] D. Cozzolino and L. Verdoliva. Single-image splicing localization through autoencoder-based anomaly detection. In *2016 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–6, 2016.

[11] Z. Fan and R. de Queiroz. Identification of bitmap compression history: Jpeg detection and quantizer estimation. *IEEE Transactions on Image Processing*, 12(2):230–235, 2003.

[12] S. Fernandes, S. Raj, E. Ortiz, I. Vintila, M. Salter, G. Urosevic, and S. Jha. Predicting heart rate variations of deepfake videos using neural ode. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 1721–1729, 2019.

[13] H. Fu and X. Cao. Forgery authentication in extreme wide-angle lens using distortion cue and fake saliency map. *IEEE Transactions on Information Forensics and Security*, 7(4):1301–1314, 2012.

[14] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila. Analyzing and improving the image quality of stylegan. *CoRR*, abs/1912.04958, 2019.

[15] L. Li, J. Bao, T. Zhang, H. Yang, D. Chen, F. Wen, and B. Guo. Face x-ray for more general face forgery detection, 2019.

[16] Y. Li, M. Chang, and S. Lyu. In ictu oculi: Exposing AI generated fake face videos by detecting eye blinking. *CoRR*, abs/1806.02877, 2018.

[17] J. Lukáš, J. Fridrich, and M. Goljan. Detecting digital image forgeries using sensor pattern noise. In E. J. D. III and P. W. Wong, editors, *Security, Steganography, and Watermarking of Multimedia Contents VIII*, volume 6072, pages 362 – 372. International Society for Optics and Photonics, SPIE, 2006.

[18] F. Marra, D. Gragnaniello, and L. Verdoliva. On the vulnerability of deep learning to adversarial attacks for camera model identification. *Signal Processing: Image Communication*, 65:240–248, 2018.

[19] F. Marra, D. Gragnaniello, L. Verdoliva, and G. Poggi. Do gans leave artificial fingerprints? *CoRR*, abs/1812.11842, 2018.

[20] F. Marra, D. Gragnaniello, L. Verdoliva, and G. Poggi. A full-image full-resolution end-to-end-trainable CNN framework for image forgery detection. *CoRR*, abs/1909.06751, 2019.

[21] J. C. Neves, R. Tolosana, R. Vera-Rodríguez, V. Lopes, and H. Proença. Real or fake? spoofing state-of-the-art face synthesis detection systems. *CoRR*, abs/1911.05351, 2019.

[22] J. Park, D. Cho, W. Ahn, and H.-K. Lee. Double jpeg detection in mixed jpeg quality factors using deep convolutional neural network. In V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, editors, *Computer Vision – ECCV 2018*, pages 656–672, Cham, 2018. Springer International Publishing.

[23] A. Popescu and H. Farid. Exposing digital forgeries in color filter array interpolated images. *IEEE Transactions on Signal Processing*, 53(10):3948–3959, 2005.

[24] A. C. Popescu and H. Farid. Statistical tools for digital forensics. In J. Fridrich, editor, *Information Hiding*, pages 128–147, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[25] Y. Rao and J. 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, 2016.

[26] A. Swaminathan, M. Wu, and K. R. Liu. Digital image forensics via intrinsic fingerprints. *IEEE Transactions on Information Forensics and Security*, 3(1):101–117, 2008.

[27] L. Verdoliva. Media forensics and deepfakes: an overview. *CoRR*, abs/2001.06564, 2020.

[28] L. Verdoliva, D. Cozzolino, and G. Poggi. A feature-based approach for image tampering detection and localization. In *2014 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 149–154, 2014.

[29] Q. Wang and R. Zhang. Double jpeg compression forensics based on a convolutional neural network. *EURASIP Journal on Information Security*, 2016, 10 2016.

[30] S. Wang, O. Wang, A. Owens, R. Zhang, and A. A. Efros. Detecting photoshopped faces by scripting photoshop. *CoRR*, abs/1906.05856, 2019.

[31] X. Yang, Y. Li, and S. Lyu. Exposing deep fakes using inconsistent head poses. *CoRR*, abs/1811.00661, 2018.

[32] S. K. Yarlagadda, D. Güera, P. Bestagini, F. M. Zhu, S. Tubaro, and E. J. Delp. Satellite image forgery detection and localization using GAN and one-class classifier. *CoRR*, abs/1802.04881, 2018.

5 - Razno / Miscellaneous

Analiza sistema avto kamer s pomočjo metapodatkov

Digitalna forenzika 2021/2022

Blaž Černi
Fakulteta za računalništvo in informatiko
bc7608@student.uni-lj.si

Miha Fabčič
Fakulteta za računalništvo in informatiko
mf8974@student.uni-lj.si

Enej Bačić
Fakulteta za računalništvo in informatiko
eb0635@student.uni-lj.si

POVZETEK

Glavna tema dela so avto kamere. Avto kamera je lahko uporaben vir za dokazovanja resnice pri prometnih nesrečah na sodišču, prav tako pa lahko posnetek tudi zajame kakšno kriminalno dejanje na ulici. Zajema veliko uporabnih podatkov, ki se jih pri forenzični analizi lahko uporabi, na primer, čas, lokacija, hitrost in podobno. V delu se dotaknemo osnov delovanja avto kamer, njihove uporabe, obdelave posnetkov in forenzične analize posnetkov, še posebej s pomočjo metapodatkov, ki jih kamere med svojim delovanjem generirajo. Dotaknemo se tudi legalnosti uporabe avto kamer v Združenih državah Amerike in v Sloveniji. Poleg tega se tudi sami lotimo pridobivanja meta podatkov iz posnetka avto kamere, pridobljenega iz interneta. Analiziramo tudi posnetek, ki ga pridobimo iz mobilne aplikacije DroidDashcam [1], ki deluje kot avto kamera. Iz posnetka katerega pridobimo iz interneta uspešno pridobimo meta podatke s pomočjo orodja exiftools, medtem ko smo pri pridobivanju podatkov iz posnetka, katerega je generirala aplikacija DroidDascham neuspešni. Iz raziskovanega članka pridemo do zaključkov, da se meta podatki glede na različne modele avto kamer precej razlikujejo, hkrati pa so zelo uporabni za rekonstrukcijo dogodkov na primer ob prometni nesreči.

Ključne besede: Avto kamere, videoposnetki, analiza, pridobivanje podatkov

1. UVOD

Avto kamere so dandanes zelo dostopne in se vse več uporabljajo. Svetovna industrija avto kamer raste iz dneva v dan. Za obdobje med 2019 in 2024 je napovedana 15.4 odstotna rast. Začelo se je tako, da so se uporabljale različne serije kamer v avtih, na primer, serija GoPro kamer, ki je bila prvotno namenjena akcijskemu snemanju. Avto kamere so uporabljene v različne namene. Novejši avtomobili imajo ponavadi različne kamere tako spredaj in zadaj, ki se uporabljajo predvsem za asistenco pri parkiranju in kot pomoč pri voznikovem zaznavanju okolice. Te kamere ponavadi niso učinkovite kot dokazno gradivo pri prometnih nesrečah ali kriminalu. Kamere, ki nas zanimajo so tako imenovane angl. *Dashcams*. Osnovni namen tovrstnih kamer je dokazovanje v primeru prometnih nesreč. V nekaterih državah, na primer, v Združenih državah Amerike, je uporaba takšnih avto kamer celo obvezna za določena vozila. To so, na primer, policijski avtomobili, avtobusi in taksiji. V nekaterih industrijah se avto kamere uporabljajo tudi za varovanje delavskih pravic, hkrati pa tudi kot dokaz za njihova ravnanja, da svoje odgovornosti za dejanja ne morejo zanikati pred nadrejenimi. V primeru, da snemamo "za zabavo" in želimo zgolj deliti zanimiv posnetek na socialnih omrežjih, je potrebno paziti na varovanje osebnih podatkov. Avto kamera je po navadi nameščena na armaturno ploščo vozila. Lahko je obrnjena naprej ali nazaj. V primeru taksistov je največkrat obrnjena nazaj, da lahko snema dejanja potnikov, medtem ko ima policija avto kamere obrnjene naprej za snemanje dogajanja na cesti pred njimi in v okolici [2]. Vse pogosteje uporabljene so tudi t. i. "dvojne" kamere; gre za dve kameri, ki snemata dogajanje pred vozilom in za vozilom. V splošnem avto kamera shrani multimedijske podatke na odstranljiv medij, na primer, na SD kartico. Proizvajalci avto kamer po navadi za namene dela forenzikov nudijo tudi programsko opremo namenjeno pregledu podatkov na teh kamerah, saj se sama forenzična orodja težko kosajo z različnimi tipi in modeli avto kamer. Raziskava [15] navaja in klasificira različne tipe metapodatkov, ki jih generira 14 različnih avto kamer, izdelanih s strani 11 različnih proizvajalcev.

Začeli bomo s pregledom področja v poglavju 2, nadaljevali s sorodnimi deli v poglavju 3, sledil bo oris dela avtorjev izbranega članka v poglavjih 4, 5, 6, 7 in zaključili bomo v poglavju 8, v katerem bomo podali še naš primer izvažanja metapodatkov iz videoposnetka.

2. PREGLED PODROČJA

V naslednjih podpoglavjih opišemo pregled področja avto kamer, njihovo uporabnost v primeru prometnih nesreč, legalnost uporabe, osnovne operacije avto kamere in strukturo multimedijskih vsebin.

2.1 Legalnost uporabe avtokamer

Različne države imajo lahko različna pravila glede legalnosti uporabe kamer v osebnih vozilih in kje v osebnih vozilih so te avto kamere lahko nameščene. Določene zvezne države v Združenih državah Amerike, na primer, dovolijo uporabo avto kamer, ampak imajo stroge pogoje glede tipa kamer,

kje se lahko uporabijo in kam jih smemo namestiti. Za primer vzamimo zvezno državo Kalifornija, kjer je v uradnem pravilniku [3] točno določeno, da je dovoljena uporaba avto kamere v osebnih avtomobilih dokler sledi vsem specifikacijam zakona in snema tudi zvok. Prav zato je odgovornost vsakega uporabnika takšne avto kamere, da potnike, še preden stopijo v avtomobil, obvesti o tem, da je avto kamera prisotna in snema tudi zvok.

2.1.1 Kako je glede legalnosti avto kamer v Sloveniji?

Glede na naravo je takšna oblika kamere neke vrsta videonadzora na javnem kraju. To trenutno ureja 74. člen Zakon o varstvu osebnih podatkov. Ta člen pa velja le za osebe javnega in zasebnega sektorja, kot ju definira ZVOP v 22. in 23. točki 6. člena. ZVOP pa ne velja za posnetke za domačo rabo. Za posnetke za domačo rabo velja: "(1) Ta zakon se ne uporablja za obdelavo osebnih podatkov, ki jo izvajajo posamezniki izključno za osebno uporabo, družinsko življenje ali za druge domače potrebe.".

V zgoraj omenjeno torej sodi uporaba takšni kamer, vendar se posnetki lahko obdelujejo zgolj za osebne potrebe. V primeru, da je takšna zbirka velika bo težko dokazati, da gre izključno za "osebno uporabo".

Torej lahko povzamemo, da snemanje je dovoljeno za osebno uporabo, ni pa priporočljivo oz. po zakonu, da hranimo večje količine takšnih posnetkov. Primer: če snemamo, hranimo na primer za en mesec nazaj raje kot za 2 leta nazaj.

2.2 Uporaba avtokamer v primeru prometne nesreče

Avto kamere v osebnih avtomobilih lahko ulovijo neizpodbitne dokaze o tem, kako so se dogodki v določeni prometni nesreči odvijali in kdo je bil za prometno nesrečo kriv. Pomagajo nam na različne načine:

- Posnetek je objektiven in nepristranski dokaz.
- V primeru udeležbe v prometni nesreči, kjer drugi voznik ne priznava krivde, se lahko z uporabo posnetka avto kamere identificira resničnega krivca.
- Ker avto kamera snema tudi zvok, se lahko uporabi na sodišču kot dokaz, da je neka trditev resnična.

2.3 Osnovne operacije avto kamere

V tem poglavju podrobneje opišemo na kakšen način avto kamera shranjuje podatke, kako upravlja z datotekami ter katerih načinov snemanja se lahko poslužimo z njo.

2.3.1 Shramba podatkov

Avto kamere najbolj pogosto za shranjevanje podatkov uporabljajo naprave kot so SD kartica ali pa Micro SD kartica, ki temeljijo na angl. *Flash* spominu. Iz tega sledi, da kapaciteta spomina takšnih naprav ni prav velika. Do multimedijskih vsebin na takšnih spominskih karticah ni težko priti, ker gre za medije, ki se jih brez težav odstrani iz naprave. Določene avto kamere so tudi povezane z oblakom in nalagajo multimedijske vsebine preko spletnih servisov.

2.3.2 Upravljanje z datotekami

Ker spominske kartice v avto kamerah nudijo premalo prostora za shrambo multimedijskih vsebin, se lahko zgodi, da zmanjka prostora med samim snemanjem. V takšnih situacijah avto kamere ponavadi izbrišejo najstarejšo vsebino preden začnejo snemati novo.

2.3.3 Različni načini snemanja

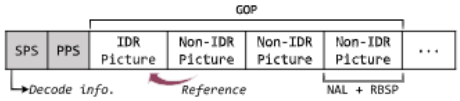
Avto kamere podpirajo različne načine snemanja. Vsak izmed načinov se uporablja kot pogoj za začetek snemanja v določenih trenutkih. Obstaja osnovni oziroma normalni način, ki je vklopljen, ko se vozimo, parkirni način, ko avtomobil ni prižgan in tudi način dogodka (angl. *Event mode*), ki se sproži ob zaznavanju sunka na pospeškometru. Poleg tega je možno snemanje vklapljati in izklapljati tudi ročno.

2.4 Struktura multimedijskih vsebin

Za shranjevanje multimedijskih vsebin na avto kamerah sta dva formata najbolj popularna - MP4 in AVI. Oba formata shranjujeta različne tipe podatkov z uporabo posebnih enot imenovanih angl. *chunk*. Te posebne enote imajo svojo hierarhično strukturo za upravljanje z različnimi tipi podatkov in vsaka izmed enot - *chunk*, sestoji iz tipa, velikosti in podatkovnega polja. Gre za v naprej definirane standardizirane enote, ki pa jih lahko razvijalci definirajo po svoje.

2.5 Video kodeki

Digitalne video naprave uporabljajo kodeke za stiskanje video vsebine na učinkovit način. Najbolj popularna standarda za avto kamere sta H.264 in H.265. Za učinkovito stiskanje podatkov sestoji tok video vsebine iz skupine slik, imenovanih *GOPs - group of pictures*. Poleg skupine GOPs sestoji standard H.264 tudi iz nekaterih drugih komponent kot sta *SPS* in *PPS*, ki vsebujejo dekodirano informacijo, in ostalih kot je prikazano na sliki 1.



Slika 1: Interna struktura standarda H.264

3. SORODNA DELA

Ob pregledu področja sorodnih del smo našli nekaj člankov glede samih avto kamer in nekaj glede digitalne forenzike in preučevanja le-te. Preučili smo posamezna dela, citiranost del in reference, ki jih vsebujejo.

Prvi članek [14], ki smo ga obravnavali kot sorodno delo, obravnava 7 različnih modelov avto kamer. Glede na podatke strani Sciencedirect [4] je članek citiran 5-krat. Vključuje pa 71 različnih referenc. Raziskovalci v članku predstavijo namen avto kamer, njihovo uporabo ter podatke, ki jih v okviru digitalne preiskave lahko pridobimo. Glede zakonskih in ostalih določil se nanašajo na Združeno kraljestvo in njihove zakone ter policijo. Poleg videoposnetka, ki ga beležijo avto kamere, avtorji predstavijo podrobno tudi avdio posnetek. Če podamo primer, avdio posnetek lahko služi kot dodano dokazno gradivo pri preiskavi in reševanju lete. Ob povezavi video in avdio posnetka pridobimo širšo

sliko dogajanja. Na primer, dogajanje pred prometno nesrečo, prepiranje v avtu, neprevidnost, prometna nesreča, ... V nadaljevanju članka predstavijo področje zbiranja dokazov iz avto kamer in procesiranja posnetkov. Naredijo eksperiment s 7 avto kamerami, kjer zrežirajo razne situacije uporabe (parkirišče, prometna nesreča, beleženje GPS podatkov,...). Za vse izmed 7 kamer so predstavili rezultate uporabnosti v različnih situacijah.

Drugo sorodno delo [12] oz. bolj podrobno delo je disertacija z naslovom "Dashcam Video Streaming and Storage". Avtor obširno predstavi področje avto kamer. Delo je obsežnejše kot prvo obravnavno delo saj gre za dizertacijo v prvem pa je le članek. Avtor podrobno obravnava področje samega hranjenja posnetkov. Predstavi podatkovne baze za hranjenje podatkov in izvede praktični primer uporabe.

Tretje sorodno delo, ki nam je služilo tekom naše naloge [16]. Članek je glede na IEEExplore spletno stran [5] citiran 4-krat. To delo se nam je zdelo izredno zanimivo saj obravnava področje dokazov iz posnetkov avto kamer. Avtorji dajo največji poudarek na uporabi posnetkov avto kamer kot dokazno gradivo in kdaj se lahko uporabljajo kdaj ne takšni posnetki. Kot zanimivost predstavijo tudi področje uporabe posnetkov kot dokaz pri zavarovalniških zahtevkih.

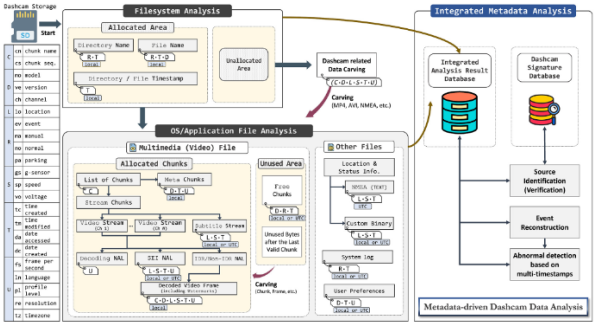
Četrto sorodno delo [13] je povezano z našim obravnavanim delom, saj ga naše sorodno delo citira. Avtorji v delu naslavljajo področje forenzične analize različnih video formatov. Implementirajo in testirajo svoj program za izluščevanje podatkov na 19 digitalnih fotoaparatih, 14 mobilnih telefonih in 6 video produkcijskih orodjih. Rezultate so predstavili v članku.

4. EKSPERIMENTALNO OKOLJE

Avtorji članka so za izvajanje njihovega eksperimenta oziroma raziskave izbrali avto kamere različnih starosti, različnih datotečnih sistemov in multimedijskih formatov, načinov snemanja in podobno. Skupno so izbrali 14 avto kamer, proizvedenih s strani 11 različnih proizvajalcev. Vsako napravo so namestili na avtomobil, da so simulirali snemanje kamere v različnih načinih.

5. VSEBINA METAPODATKOV AVTO KAMERE

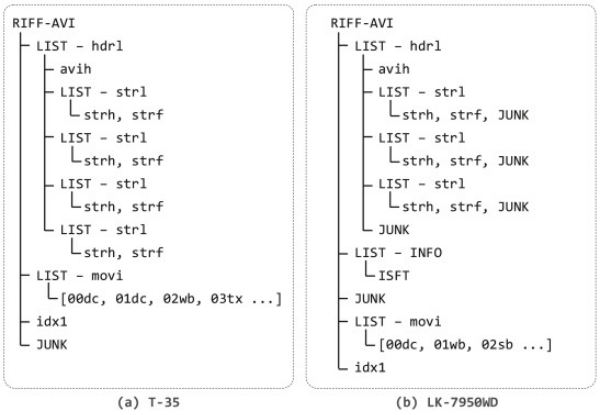
Raziskovanja metapodatkov v avto kameri so se lotili na način, kot je prikazan na sliki 2. S pomočjo analize so dobili 22 različnih metapodatkov, shranjenih na avto kameri na različnih lokacijah. To so ugotovili s pomočjo sistematične analize datotečnega sistema, operacijskega sistema in aplikacije. Večina uporabljenih modelov avto kamer je uporabljala FAT32 ali pa exFAT datotečni sistem. Direktorijska hierarhija omogoči raziskovalcem, da identificirajo specifične tipe modelov avto kamer in časovnih žigov. Vsaka naprava generira unikatno direktorijsko strukturo zato je pred forenzično raziskavo avto kamer potrebno pridobiti informacije, ki so specifične za napravo. Vsak model avto kamere ima svoja pravila za imenovanje datotek, ki jih generira. Določene uporabne metapodatke za forenzično analizo kot so datum in čas posnetka, kanala kamere in način snemanja, se lahko pridobi že iz samega imena multimedijske datoteke. Ker avto kamere velikokrat snemajo po principu 24/7, se multimedijske vsebine pogosto brišejo in na novo ustvarjajo, zato morajo forenziki uporabljati posebne procese (na primer obrezovanje, angl. carving) za povrnitev "izgubljenih" datotek.



Slika 2: Predlog avtorjev za analizo metapodatkov avto kamere

5.1 Vsebniki datotek

Kot smo že ugotovili v poglavju 2.4, format multimedijskega vsebnika shrani različne tipe podatkov s pomočjo posebne enote imenovane angl. *chunk*. Za potrebe naše raziskave ga bomo poimenovali *kos*. Specifične modele avto kamer je mogoče prepoznati preko informacij, povezanih s temi kosi. Različni seznam generiranih kosov iz dveh različnih modelov avto kamer je tudi dobro viden na sliki 3. Avtorji članka so s pomočjo njihovega eksperimenta ugotovili, da imajo vsi modeli, ki podpirajo MP4 format, tudi unikatna zaporedja generiranja zaporedja kosov. Še posebej uporabni kosi z vidika digitalnega forenzika so tako imenovani *meta kosi*, ki vsebujejo časovne žige, resolucijo, čas trajanja posnetka, informacije o napravi in podobno.



Slika 3: Seznam kosov kreiranih iz dveh različnih modelov avto kamer

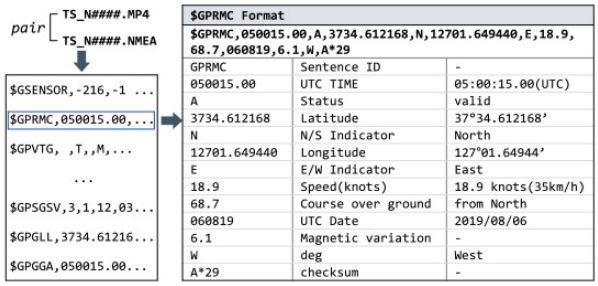
5.2 Tok video podatkov

Gre za tok podatkov, pri katerem so shranjeni surovi podatki, kjer se poleg vidne informacije shrani tudi precej uporabnih metapodatkov. Kodeka H.264 in H.62 shranita informacijo potrebno za dekodiranje resolucije in števila sličic na sekundo. Pri modelu avto kamere *CR-500HD*, so avtorji

ugotovili, da kamera generira informacije v posebnem formatu, kamor pa se shranijo tudi zanimivi metapodatki kot so čas posnetka in lokacija snemanja v obliki koordinat. Podobne informacije so uspeli pridobiti tudi iz toka podatkov za podnapise.

5.3 Ostale datoteke

Avtorji navajajo tudi posebno tekstovno datoteko (.NMEA), kjer avto kamere prav tako shranjujejo precej zanimive informacije za digitalno raziskavo kot je videti na sliki 4. Poleg tega pa so ugotovili tudi, da imajo določeni modeli kamer celo precej natačno opisovanje beleženja dogodkov, kjer so časovno označene informacije kot so zagon kamere, vstavitev SD kartice v kamero, začetek snemanja, kreiranje posnetka, spreminjanje načinon snemanja in podobno.



$GPRMC Format		
$GPRMC,050015.00,A,3734.612168,N,12701.649440,E,18.9,68.7,060819,6.1,W,A*29		
GPRMC	Sentence ID	-
050015.00	UTC TIME	05:00:15.00(UTC)
A	Status	valid
3734.612168	Latitude	37°34.612168’
N	N/S Indicator	North
12701.649440	Longitude	127°01.64944’
E	E/W Indicator	East
18.9	Speed(knots)	18.9 knots(35km/h)
68.7	Course over ground	from North
060819	UTC Date	2019/08/06
6.1	Magnetic variation	-
W	deg	West
A*29	checksum	-

Slika 4: Primer .NMEA datoteke kreirane s kamero DrivePro220.

6. INTEGRACIJA IN ANALIZA METAPODATKOV IZ AVTO KAMER

Avtorji so za učinkovito shranjevanje podatkov metapodatkov ddefinirali shemo za podatkovno bazo z različnimi tabelami. Tabele so bile razdeljene na tabelo z osnovnimi podatki avto kamere, tabelo z informacijami direktorijske strukture avto kamere, tabelo z informacijami datotek v direktorijih, tabelo z metapodatki in tabeli s časovnimi oznakami. S pomočjo predlagane strukture podatkovne baze so kreirali časovnico dogodkov za avto kamero, ki omogoča boljšo analizo dogodkov s pomočjo pridobljenih meta podatkov in tudi rekonstrukcijo toka dogodkov ob morebitni nesreči, kot je prikazano na sliki 5. Eden izmed ciljev avtorjev ob zaključku analize je bil tudi identifikacija oziroma razlikovanje avto kamer na podlagi pridobljenih metapodatkov. Njihova analiza je hitro potrdila, da je v ta namen potrebno uporabiti kombinacijo različnih metapodatkov. Razlikovanje samo na podlagi enega tipa metapodatkov, na primer, hierarhije datotek ali pa pravil imenovanja datotek ni mogoče, ker določeni modeli avto kamer uporabljajo enaka pravila poimenovanja.

7. IMPLEMENTACIJA AVTORJEV

Na podlagi njihovih ugotovitev so avtorji implementirali avtomatsko orodje za procesiranje metapodatkov, pridobljenih iz avto kamer. Njihovo orodje imenovano *plaso* je bilo razdeljeno na dva dela: *predprocesiranje* in *analizo*. Prvi del sestoji iz pridobivanja podatkov iz datotek pridobljenih iz avto kamer in ekstrahiranja metapodatkov. Drugi del pa se uporabi za izvajanje filtriranja, interpretiranja in integriranja zapisov nad izhodom iz prvega dela. Kot rezultat

No	Time	Timezone	source	type	data
1	2020/01/05 14:20:30	UTC	filesystem	time created	-
2	2020/01/05 14:20:30	UTC	meta_chunks	time created	-
3	2020/01/05 14:20:30	UTC	filename	record.mode	normal
4	2020/01/05 14:20:31	UTC	subtitle	location	37.583230, 127.027490
5	2020/01/05 14:20:31	UTC	subtitle	status	0.5,-0.4,-0.6
6	2020/01/05 14:20:31	UTC	subtitle	speed	14
n			...		
n+1	2020/01/05 14:25:10	UTC	filename	record.mode	event
n+2	2020/01/05 14:25:10	UTC	subtitle	location	37.583099, 127.027152

Slika 5: Primer rekonstrukcije dogodkov avtorjev za avto kamero.

teh operacij, se kreira podatkovna baza, katero smo opisali v poglavju 6. Prototip implementacije so kot odprtokodni projekt objavili tudi na Github [6].

8. PRIMER IZVAŽANJA METAPODATKOV IZ VIDEOPOSNETKA

V tem delu smo iz dveh posnetkov avto kamere poskušali izvoziti metapodatke. V prvem primeru smo pridobili posnetek [7] avto kamere z interneta, izvozili metapodatke, iz njih izluščili geolokacijo poti in jo nato izrisali na zemljevid. V drugem primeru smo pridobili mobilno aplikacijo, ki deluje na podoben način kot avtokamera in iz nje poskušali pridobiti metapodatke.

8.1 Prvi primer

Na Sliki 6 je viden posnetek [7], ki je bil posnet z avto kamero Rove R2-4K [8]. Najprej smo z orodjem **exiftools** [17] uspešno dobili metapodatke o geolokaciji, kot je vidno na Sliki 7. Iz pridobljenih podatkov lahko razberemo čas posnetka, geografsko širino ter dolžino, hitrost in stanje senzorja za pospešek. V avto kameri je bilo nastavljeno, da se v metapodatke zapiše geolokacija na vsako sekundo.

Slika 6: Posnetek avto kamere Rove R2-4K.

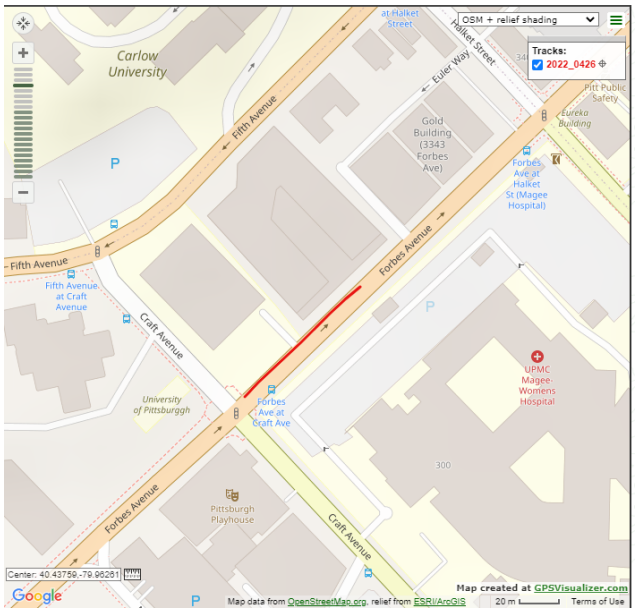
Te metapodatke geolokacije smo nato prav tako z orodjem exiftools izvozili v format .gpx. Dobljeno datoteko smo nato odprli s spletno stranjo GPSVisualizer [18], ki je dane geolokacije prikazala na zemljevidu, kot je vidno na Sliki 8.

8.2 Drugi primer

Na naš telefon smo si namestili aplikacijo DroidDashcam [1], ki spremeni telefon v avto kamero. Kakor ostale avto kamere tudi ta aplikacija snema posnetke, dolge nekaj minut,

```
GPS Date/Time                    : 2022:04:26 11:56:26Z
GPS Latitude                     : 40 deg 26' 13.87" N
GPS Longitude                    : 79 deg 57' 46.93" W
GPS Speed                        : 35
Accelerometer                    : -0.02, 0.96, 0.23
GPS Date/Time                    : 2022:04:26 11:56:27Z
GPS Latitude                     : 40 deg 26' 14.10" N
GPS Longitude                    : 79 deg 57' 46.64" W
GPS Speed                        : 35
Accelerometer                    : -0.05, 1.01, 0.21
```

Slika 7: Prikaz metapodatkov geolokacije.



Slika 8: Prikaz izvoženih podatkov na zemljevidu.

na posnetek pa lahko doda uro, geolokacijo in hitrost avtomobila, kot je prikazano na Sliki 9. Datoteke so shranjene v formatu MP4 in zakodirane s kodekom AV1 in ob pomanjkanju prostora na shranjevalnem mediju se prav tako prepiše najstarejši posnetek.

Slika 9: Zapisani podatki na posnetku.

Metapodatke smo najprej dobili z orodjem exiftools [17]. Orodje je pridobilo osnovne metapodatke kot je recimo resolucija posnetka, dolžina posnetka, datum in čas snemanja itd. Ni pa uspel dobiti metapodatkov o geolokaciji. Metapodatke o geolokaciji smo poskušali pridobiti tudi z orodjema ffmpeg [9] ter mp4box [10], vendar tudi ti dve orodji nista bili uspešni.

Nato smo videoposnetek odprli s programom ImHex [11], ki prikaže binarno vsebino datoteke. S tem programom smo poiskali nize v vsebini datoteke kot so na primer “GEO“, “LOCATION“, “GPS“ in podobne, vendar se ti niso nahajali v datoteki. Poskušali smo tudi poiskati nize kakršnih koli že znanih podatkov, na primer, lokacijo snemanja (iskali smo niz “Ljubljana“), vendar tudi teh nismo dobili. Metapodatkov o geolokaciji nismo našli, čeprav so prisotni v videoposnetku. Na podlagi tega sklepamo, da jih je aplikacija DroidDashcam [1] zapisala v sam posnetek in ne kot metapodatke.

9. ZAKLJUčKI

V nalogi smo prikazali pregled področja avto kamer. Avto kamere niso zgolj naprave, ki nam omogočajo varnejšo vožnjo, temveč služijo tudi kot dokazni material pri morebitnih preiskavah. Kot uporabnik je treba biti previden, da je uporaba avto kamere sploh legalna. Poleg same vsebine videoposnetka so pomembni tudi metapodatki, ki jih najdemo na avto kameri.

Pri preverjanju metapodatkov enega izmed naših videoposnetkov smo uspešno pridobili podatke o lokaciji in času snemanja, medtem ko na drugem videoposnetku teh podatkov nismo uspeli dobiti. Našo raziskavo bi lahko nadgradili z nakupom avto kamere, saj bi tako lahko pregledali še dodatne metapodatke, ki se nahajajo v datotečnem sistemu.

10. ZAHVALA

Posebna zahvala za pomoč pri naslovu Kako je glede legalnosti avto kamer v Sloveniji? 2.1.1 gre izr. prof. dr. Primožu Gorkiču, univ. dipl. prav.. Pomagal nam je pri razrešitvi dileme kako je z legalnostjo avto kamer pri nas, ter nas podučil glede pravnega vidika uporabe avto kamer v Sloveniji.

11. LITERATURA

[1] Dosegljivo: https://play.google.com/store/apps/details?id=com.helge.droiddas [Dostopano: 28.4.2022].

[2] Dosegljivo: https://helbocklaw.com/what-role-do-dashcams-play-in-car-accidents/. [Dostopano: 20.4.2022].

[3] Dosegljivo: https://leginfo.legislature.ca.gov/faces/codes$_{d}$isplaySection.xhtr $VEHsectionNum = 26708.[Dostopano : 20.4.2022]$.

[4] Dosegljivo: https://www.sciencedirect.com/. [Dostopano: 27.4.2022].

[5] Dosegljivo: https://ieeexplore.ieee.org/. [Dostopano: 27.4.2022].

[6] Dosegljivo: https://github.com/kukheon1109/Dashcam$_{P}$laso.[$Dostopano$: 2.5.2022].

[7] Dosegljivo: https://exiftool.org/forum/index.php?topic=5095.120. [Dostopano: 28.4.2022].

[8] Dosegljivo: https://www.rovedashcam.com/products/rove-r2-4k-car-dashcam. [Dostopano: 28.4.2022].

[9] Dosegljivo: https://ffmpeg.org/. [Dostopano: 28.4.2022].

[10] Dosegljivo: https://github.com/gpac/gpac/wiki/MP4Box. [Dostopano: 28.4.2022].

[11] Dosegljivo: https://github.com/WerWolv/ImHex. [Dostopano: 28.4.2022].

[12] J. Costa. *Dashcam Video Streaming and Storage*. PhD

dissertation, FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO, 2019.

[13] T. Gloe, A. Fischer, and M. Kirchner. Forensic analysis of video file formats. *Digital Investigation*, 11:S68–S76, 2014. Proceedings of the First Annual DFRWS Europe.

[14] H. S. Lallie. Dashcam forensics: A preliminary analysis of 7 dashcam devices. *Forensic Science International: Digital Investigation*, 33:200910, 2020.

[15] K. Lee, J.-H. Choi, J. Park, and S. Lee. Your car is recording: Metadata-driven dashcam analysis system. *Forensic Science International: Digital Investigation*, 38:301131, 2021.

[16] A. Mehrish, A. Subramanyam, and M. Kankanhalli. Multimedia signatures for vehicle forensics. In *2017 IEEE International Conference on Multimedia and Expo (ICME)*, pages 685–690, 2017.

[17] Phil Harvey. *ExifTool*. Kingston, Ontario, Kanada, 2022.

[18] A. Schneider. Gps visualizer.

An Enhanced Blockchain-Based Digital Forensics Architecture Using Fuzzy Hash

Luka Boljević
lb7093@student.uni-lj.si

Teodor Janez Podobnik
tp7220@student.uni-lj.si

Hana Zlobec
hz2731@student.uni-lj.si

ABSTRACT

A recently proposed way to achieve better IoT security using blockchain technology includes using an immutable ledger, a decentralization architecture and a well-established low-level cryptographic algorithm. An issue that presents itself is that not all IoT devices are compatible with existing implementations of this idea, as some of them use conventional hash, which is useless for comparing similar files (as even the smallest change causes the hash to be completely different). The authors of the reviewed paper [5] propose an approach using fuzzy hash (in addition to conventional hash for authentication) to construct the Blockchain's Merkle tree, which allows the discovery of altered files by comparing its hash to the one in the Blockchain network. We propose and implement an improved approach, which could be beneficial to all types of digital forensics.

Keywords

Blockchain, Internet of Things, Fuzzy hash, IoT forensics, digital examination

1. INTRODUCTION

Everyday usage of IoT devices is becoming more and more common, which means that the diversity and sheer amount of data that forensic tools have to process in case of an investigation is also rapidly increasing. IoT stands for "**The Internet of Things**" and is a network of physical objects with sensors, processing ability, ability to exchange data with other devices and does not require human interaction to do so. A common examples of IoT devices are "smart home" appliances e.g. thermostats, smart locks, cameras... but also drones and cars connected to the network, which creates a wide attack surface that expands with more such devices emerging.

The diverse nature of data from IoT devices provide difficulties for ensuring the collected evidence has not been tampered with. This is additionally hindered by the lack of an integrated standard for IoT-based forensic investigation [4]. Existing methods rely on the data of interest being always readily accessible, which can not always be said for IoT related data.

The idea of using **blockchain technology** in digital forensics is becoming more and more interesting, as it provides a better alternative to the traditional centralized architecture, which is based on a central authority verifying the collected digital evidence. This provides an opportunity for the evidence being tampered with, either by insiders or attackers.

Blockchain is a list of blocks (records) that are linked together. Every block contains a cryptographic hash of the previous one, transaction data and a timestamp, which proves that the data existed when the block was published. The transaction data is usually represented as a **Merkle tree**, where each leaf is a data node. Blockchain is resistant to alterations, since no block can be modified without this impacting all blocks that follow it [1]. Using a blockchain for integrity preservation of data acquired in a forensic process is becoming more popular, as it makes the internal information and the ledger visible to all participants, therefore allowing any participant to verify data's integrity. There are, however, some constraints when it comes to using blockchain for IoT. The first is the constrained processing power and storage capacity of IoT devices which is necessary to secure and maintain a blockchain. Secondly, each new block's capacity is restricted (transactions are limited to a few dozen per second), which makes it hard to keep up with the rapid deployment of new IoT devices. Then there is the confirmation delay (single block creation), which would require a lightweight cryptographic hash algorithm [9], because of the aforementioned slow availability of new blocks.

The downside of traditional hashing techniques, which are usually used in blockchain, is the fact that every unique file/data stored on a blockchain, no matter how similar it is to some other data, will have a *completely different* transaction hash. This is due to its inherent property called *cryptographic diffusion* and it enables certain pieces of evidence (that might have similar binary strings as already documented evidence) to remain undiscovered. This is where **fuzzy hash** comes in. It enables comparing hashes of files that are *homologous*, which means they are similar, but are not perfect copies, since the similarity is retained in the hash. Using fuzzy hash, we can compare incomplete, malformed, partially obstructed files to the original in order to find the

connection with the suspect.

The method from the main article[5], which we aim to improve, has the difficulties of IoT forensics in mind, but any part of digital forensics could benefit from this architecture.

1.1 Article Overview

In chapter 2 we provide an overview of the proposed approach from [5], a brief description of its main components, and the authors' results in 2.5. In chapter 3, we describe the downsides of that approach and describe and propose our own architecture, which uses Smart Contracts on the Ethereum network. In chapter 4, we test and compare a few fuzzy hashing algorithms that can be employed within the two described architectures. The article is concluded with chapter 5, where we repeat our main findings.

2. PROPOSED APPROACH

The approach builds on previous work done on distributed ledger architectures, with the significant improvement of using **fuzzy hash** instead of *traditional hash* for preserving the integrity of blocks and easier analysis, as seen in figure 1. By combining these approaches it makes the forensic examination more manageable by accomplishing the following:

- Continual integrity for the whole evidence chain,
- Immutable storage of the record fingerprints,
- Transparency of the evidence chain,
- Identification of related records

In the following subsections, we will describe the main components of the framework and how the immutable chain of records fingerprints is constructed. We will refer to a hash of the record as a **fingerprint** or **digest**, since we are not excluding an option for the record to be digitally signed. Similarly, we will use terms such as **record** or (*transactional*) **evidence**, but they all refer to the same thing.

2.1 Proposed Forensic Chain

The proposed framework uses a distributed ledger for storing fingerprints of **transactional evidences** (TEs), while enabling users to sort related records based on **fuzzy hash**.

2.2 Transaction

Forensic chain starts by **fuzzy hashing** the data, producing a transaction hash that uniquely identifies it's corresponding data. Since each node on a blockchain should retain a copy of the complete ledger, it's reasonable to keep the amount of data stored on the chain as low as possible. To further optimize the amount of data stored on each block, a Merkle tree is constructed.

2.3 Merkle Tree

The **Merkle tree** is a hash tree, constructed by hashing all transaction hashes until it converges into a single hash called Merkle root, as seen on the Figure 2. Merkle root ensures that the each investigation's findings can be securely verified against it, requiring only reduced number of hashes that it relates to, rather than all hashes from the tree.

2.4 Blocks

To form an immutable object, Merkle root is combined with the following items to form a single block on the blockchain: (i) pre-block hash, (ii) which version it is, (iii) the nonce, (iv) the timestamp, (v) the block state and (vi) the Merkle root. Joining the described components we form an unbreakable chain of records, transparently stored on a decentralized network.

2.5 Results of Proposed Approach

Authors of [5] implemented the proposed model in **Python** and tested its response time, delay and throughput capabilities. They started by building the Merkle tree with fuzzy hash after applying SHA256 to the TE records, then validating the blockchain, using fuzzy hash for the Merkle root and then implementing the PoW and creating a text file with the block information.

First, they compared node response time in the cases of using fuzzy and traditional hash to create the Merkle tree. They measured the time the node takes to receive the transaction, mine a new block and create the text file with block information. The results show that in this model, using fuzzy hash reduces the response time by an average of 2% when compared to the version using traditional hash to encode transactional evidence records. This confirms that the model can deal with a large number of blocks and therefore validates the idea of using this model in real time.

They then tested the CPU usage in dependency of the number of generated blocks. As they expected, needed CPU resources increase with the need for new blocks, although the need for CPU resources seems to stagnate with the larger number of needed blocks. To stress-test, they simulated 50, 100 and 500 concurrent users with the users performing operations like getting the blockchain status, posting the transaction to the node and getting the mined block status. They observed that the completion time increased with the number of users and that getting the blockchain status and mining took the most time, which makes sense since mining needs more resources to complete PoW and hashing the Merkle root.

Lastly they tested the model on a Raspberry Pi, to confirm its ability to run on IoT devices. They considered the Raspberry Pi as a node in the blockchain network and focused on monitoring time consumption and power consumption, as these are the main issues running blockchain on IoT. They measured the power using a wall outlet power meter. The results were compared to consumption when using a laptop, as you can see in table 1.

	Laptop	Raspberry Pi
$\bar{t}$[s]	3	13
$\bar{P}$[mW]	4	12

Table 1: Average measurements of time and power consumption of generating a new block and adding it to the blockchain network. [5]

The time consumption difference makes sense since we are comparing a laptop with more resources compared to a Rasp-

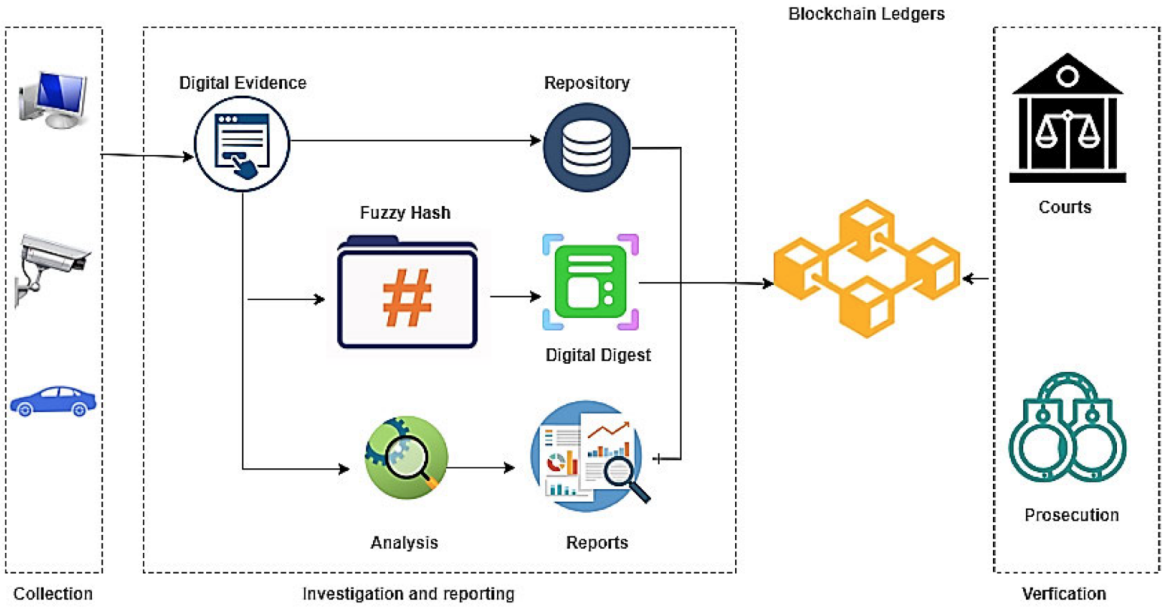


Figure 1: IoT evidence management using blockchain and fuzzy hash [5].

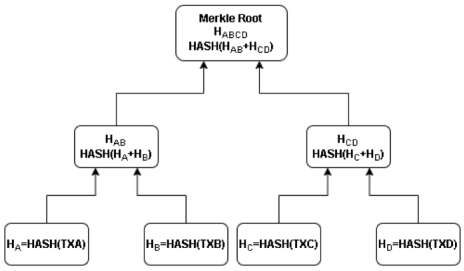


Figure 2: Merkle Tree

berry Pi. If we then compare this difference to the difference in power consumption, it is smaller.

The model does show some space for improvement on its execution time and complexity, which requires further research and testing. In any case, the proposed model seems like a good option even for implementations on devices with power and memory scarcity (like IoT devices).

3. ALTERNATIVE APPROACH

The downside of authors approach lies in the fact that in order to accomplish it, core algorithms of the blockchain need to be modified in order to suffice the need for similar records identification. We believe a better approach is to utilize Smart Contracts on Ethereum network, that enables users to build **decentralized applications** (DApps) on top of the blockchain, while preserving the underlying technology.

The following sections summarize the main components of the alternative approach and compare the two methods. Note that the two models mainly differ in terms of usability, while both address the same issue.

3.1 Data Digest

Each execution in the **Smart Contract** (SC) costs a fee, therefore it's desirable to perform as many as possible operations outside of the network. In other words, before interacting with the SC, we **(fuzzy) hash** the data locally - not to mention hashing is a costly operation and with this, we avoid higher costs. Each new hash of the potentially modified evidence is chained by prefixing it with the fuzzy hash of the evidences previous state which allows us to chronologically order the records.

Additionally, when pushing data to the blockchain it's advisable to lower the amount of it as much as possible. Not only would it take undesirable amount of time to complete the transaction (about 14 minutes to save just 1 MB of data), but it would also cost much more than to store the data just on an occasional database. In the following section we describe an additional technology used to store data and circumvent this issue.

3.2 Data Storage

In comparison to centralized database, **InterPlanetary File System** (IPFS) is a decentralized database we utilize to preserve transparency and immutable storage of the data. Primary purpose of this technology is not meant to store sensible information since any participant of the IPFS network can host and see your files. Therefore an additional public key infrastructure is needed, to encrypt the data, but this is out of scope of this article.

Stored data is content addressable enabling user to retrieve

the content based on an unique identifier. On top that, IPFS network allows us to reference arbitrary content using a single identifier which is achieved by **InterPlanetary Name System** (IPNS). The concept is summarized in Algorithm 1. In the subsequent chapters we will refer it as an IPNS identifier and use it to group evidence and its modification in the Smart Contract. In practice, the user is still required to keep a separate note of relevant content addresses.

Our proposed solution is not bound to use IPFS as a storage platform, rather the main component of the system is to use Smart Contracts to enable any party to self-verify the integrity of data.

Algorithm 1 Storing data on IPFS

INPUT: Digital evidence DE
OUTPUT: IPNS identifier I : Fuzzy hash F
$K \leftarrow Storedata\_on\_IPFS(DE)$
$I \leftarrow BindCID\_to\_IPNSidentifier(K)$
$F = calculate\_fuzzy\_hashed\_value(DE)$
return I, F;

3.3 Smart Contract

Smart contracts are digital contracts, which are kept on the Ethereum network and execute in a secure and immutable way while not requiring a third-party to communicate with the rest of the network.

When applied to IoT devices, **Proof of Work** (PoW) algorithm needed to establish a consensus between nodes about the ledger requires a lot of resources (memory and processing power), consequently a simplified consensus algorithm needs to be considered. Note that this issue is already partly addressed by the new Proof of Stack (PoS) consensus algorithm Ethereum community is actively developing.

With IoT, using a simplified PoW algorithm with a lower difficulty decreases the time needed to create a new block and with that, makes reaching the consensus between the nodes of the IoT network easier and much faster.

4. HASH ALGORITHMS COMPARISON

The proposed architectures both utilize **fuzzy hashing** instead of traditional hashing algorithms, to make sure that the integrity of the blockchain is maintained, and to enable preservation of similarity between files. This can not only detect whether a file has been tampered with but also provides a measure of change. In this chapter, we will compare 3 popular fuzzy hashing algorithms to see which one may be best suited for the purposes the architectures seek to achieve.

4.1 ssdeep

The first algorithm we will consider is `ssdeep` [3], probably the most popular fuzzy hashing algorithm. The process of `ssdeep` fuzzy hashing begins by splitting the file into several blocks, which are then each separately hashed. At last, all block hash values are then concatenated to create a joined fuzzy hash value. The length of the resulting fuzzy hash is influenced by the file size, block size and the output size of the chosen hashing algorithm. The proposed method in [5] precisely uses `ssdeep`, which outputs context-sensitive piecewise hashes [2] (**CTPH**), which is also what fuzzy hashes are called. This means we are able to identify identical bytes in the same order, even if bytes separating them vary in length and content. Files like that are also called **homologous**.

After the fuzzy hash is constructed, the digital investigator can make conclusion on the size of modifications. He is also able to determine which block of the files were changed. `ssdeep` outputs a measure of similarity between two files, meaning a 100 denotes files are equal, while 0 means files are totally different.

4.2 LZJD

The next fuzzy hashing algorithm we will have a look at is `LZJD`, or **Lempel-Ziv Jaccard Distance** [7, 8]. `LZJD` is a distance metric designed for arbitrary byte sequences, and was originally used for malware classification. The background and motivation for `LZJD` was given in [8], but here we're going to briefly cover what `LZJD` does to compute file similarity.

This method works by building a set of previously seen strings from the given byte string b. The set initially starts out empty, and a pointer starts at the beginning of the file looking for substrings of length 1. If the pointer is looking at a substring that has been seen before, we leave it in place and increment the desired substring length by 1. If the pointer is at a substring that has not been seen before, it is added to the set. Then the pointer is moved to the next position after the substring, and the desired substring length reset to 1. This is repeated until no new items can be added to the set. The function returns the constructed set. The strings in the set will get progressively longer as the length of the input increases. A pseudo code, for easier understanding, is given in Algorithm 2.

Algorithm 2 Lempel-Ziv set [8]

Input: Byte string b
$S \leftarrow \{\}$
$start \leftarrow 0$
$end \leftarrow 1$
while $end \leq |b|$ **do**
    $b_s \leftarrow b[start : end]$
    if $b_s \notin S$ **then**
        $S \leftarrow S \cup \{b_s\}$
        $start \leftarrow end$
    end if
    $end \leftarrow end + 1$
end while
return s

Once we have these sets for two binary strings of interest, we can compute the **Jaccard similarity** between these two sets:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

The value $J(A, B)$ is in the range $[0, 1]$, so the output of `LZJD` is a percentage which indicates how similar two byte strings are. Computing this similarity, especially for longer byte strings, turns out to be rather slow, so approximations are used to speed up the process. However, we will not go into details here, as this is not the topic of this paper.

(a) Original image with no watermark (b) Changed image with added watermark (bottom right)

Figure 3: Images used for comparing fuzzy hashes

4.3 TLSH

The last fuzzy hashing algorithm we will consider here is `TLSH` or **Trend Micro Locality Sensitive Hash** [6]. Given a byte stream with a minimum length of 50 bytes, `TLSH` will generate a hash value which can be used for similarity comparisons. We will shortly cover the steps `TLSH` does to produce a hash value for a byte string.

Observe any byte string. The `TLSH` hash of that byte string is computed during 4 steps:

1. Process the byte string using a sliding window of size 5 to populate an array of bucket counts.
2. Calculate the quartile points of the bucket array, q_1, q_2 and q_3
3. Construct the digest header values (header of the digest has 3 bytes)
4. Construct the digest body by processing the bucket array

The final `TLSH` digest constructed from the byte string is the concatenation of the hexadecimal representation of the digest header values from step 3, and the hexadecimal representation of the binary string from step 4. The scoring function that is used for `TLSH` is rather complicated, but it is just important to point out that in theory, it is in the range $[0, \infty]$. i.e. it is theoretically unbounded from above, unlike `ssdeep` and `LZJD`. This means the higher the `TLSH` score, the more differences exist between the two binary strings we are comparing (while a score of 0 of course means the two strings are identical). Likewise, in the original paper for `TLSH`, the authors reason the choices for some of the sub-parts of each step, but we will not go into that here. We will immediately dive into comparing these 3 fuzzy hashes.

4.4 Comparison

Let us now perform the comparisons. First off, consider two pieces of text, with only one change being that the second text uses a capital B for the first occurrence of the word "building":

- "Because highly fit schemata of low defining length and low order play such an important role in the action of genetic algorithms, we have already given them a special name: ***building*** blocks. Just as a child creates magnificent fortresses through the arrangement of simple blocks of wood, so does a genetic algorithm seek near optimal performance through the juxtaposition of short, low-order, high-performance schemata, or building blocks"
- "Because highly fit schemata of low defining length and low order play such an important role in the action of genetic algorithms, we have already given them a special name: ***Building*** blocks. Just as a child creates magnificent fortresses through the arrangement of simple blocks of wood, so does a genetic algorithm seek near optimal performance through the juxtaposition of short, low-order, high-performance schemata, or building blocks"

The small Python script we wrote returned the result seen in figure 4.

```
TEXT COMPARISON:
TLSH hash comparison(the lower the number, the better):
difference = 84

SSDeep hash comparison(the higher the number the better):
similarity = 97

LZJD hash comparison(the higher the number the better):
similarity = 0
```

Figure 4: The comparison/similarities returned for the provided text

We see in figure 4 that `ssdeep` seems to be the best one out of the bunch. It returns that the two short paragraphs are 97% similar. On the other hand, `LZJD` was not able to detect any differences between them, while `TLSH` detected it. If we make another change, and instead of the word being "Building", we set it to "BUilding", `ssdeep` now returns a similarity of 100%, while `TLSH` returns the result of 87 (only an increase of 3 in the difference score). `LZJD` still returns 0. If we set the word to be "BUIlding", `ssdeep` returns

97%, while TLSH surprisingly returns only 11. Even though the second result returned by `ssdeep` is not the most accurate, we can definitely turn our attention away from `TLSH`, as it showed major inconsistencies with the returned score. Hence, `ssdeep` seems to be (way) more consistent with its results (at least for text).

Now consider two pictures, one without a watermark (figure 3a), and the other one completely the same, but with our watermark (figure 3b) - this picture was altered ever so slightly. A change like this, in practice, may indicate that the criminal tried to portray the image as his own, i.e. he denies any copyright infringement or plagiarism. The results returned by our short script are shown in figure 5.

```
IMAGE COMPARISON:
TLSH hash comparison(the lower the number, the better):
difference = 199

SSDeep hash comparison(the higher the number the better):
similarity = 0

LZJD hash comparison(the higher the number the better):
similarity = 0
```

Figure 5: The comparison/similarities returned for the shown images

For reference, if we take any picture, change one byte of it, and compare those two pictures, `TLSH` will return a result of just 1! (while `ssdeep` and `LZJD` both return 0). Now, even though the pictures are slightly different, we notice (in figure 5) that `ssdeep` and `LZJD` fail to detect the differences, while `TLSH` detects it, with a difference of 199. It is a noticeably larger score than 1, however, unlike `ssdeep` and `LZJD`, it is at least sensitive enough to detect the differences. The similar result is obtained with other, similar images (one original, and one with a very light watermark). Out of these three algorithms, we can say that `TLSH` is the most consistent for images.

5. CONCLUSIONS

The approach outlined in [5] is a rather adequate approach preservation of data integrity. Above all, it allows digital investigators to carry out their investigations and examinations in a more manageable and modern way, while still providing more or less completely secure evidence preservation. A blockchain based digital forensic framework is a perfect for this, but the described approach does not come without downsides. The problem lies in the fact that in order to successfully accomplish it, the core of the blockchain needs to be changed, only such that similar records can be identified.

Our approach utilizes Smart Contracts running on the Ethereum network. This enables users to build decentralized applications on top of the blockchain, while preserving the underlying technology. This removes the need to change the core of the blockchain, while solving the exact same issue as the original approach. Likewise, our architecture enables users to use also other fuzzy hashing technologies interchangeably, making it backward compatible and easily upgradeable in the future.

Setting aside the resources required to secure and efficiently run a larger scale blockchain network for IoT devices, the lightweight and popular fuzzy hashing algorithms that would be algorithms of choice, namely `ssdeep` for text based evidence, and `TLSH` for images and videos as discussed in section 4, are easy to use and the code is open source too. Unfortunately, we are yet to see a fuzzy hashing algorithm that can work well for both images *and* text at the same time, but our proposed architecture is constructed in such a way where any fuzzy hashing algorithm can be used.

Using blockchain for integrity preservation of evidence during a forensic investigation is growing in popularity, especially in recent years. The difficulty of preserving the integrity of our evidence (or anything for that matter) is built right into and literally defines blockchain as a concept, and we are making use of exactly that. The aforementioned lightweight fuzzy hashing algorithms are provably good for digital forensics purposes. Regardless of the drawbacks of the approach described in [5], a blockchain-based digital forensics architecture brings many advantages over traditional investigation and evidence storing techniques, and as this technology develops, it is only going to become easier for digital investigators to perform their already cumbersome investigations.

6. REFERENCES

[1] Cao, Bin and Liu, Weikang and Peng, Mugen. *Blockchain-Driven Internet of Things*, pages 93–115. 2022.

[2] J. Kornblum. Identifying almost identical files using context triggered piecewise hashing. 3, 2006.

[3] Kornblum, Jesse. Identifying Almost Identical Files using Context Triggered Piecewise Hashing. Digital Investigation 3(suppl.), 91-97. *Digital Investigation*, 3:91–97, 09 2006.

[4] MacDermott, Áine and Baker, Thar and Shi, Qi. Iot Forensics: Challenges for the Ioa Era. pages 1–5, 02 2018.

[5] Mahrous, Wael A. and Farouk, Mahmoud and Darwish, Saad M. An Enhanced Blockchain-Based IoT Digital Forensics Architecture Using Fuzzy Hash. *IEEE Access*, 9:151327–151336, 2021.

[6] Oliver, Jonathan and Cheng, Chun and Chen, Yanggui. TLSH – A Locality Sensitive Hash. In *2013 Fourth Cybercrime and Trustworthy Computing Workshop*, pages 7–13, 2013.

[7] Raff, Edward and Nicholas, Charles. An Alternative to NCD for Large Sequences, Lempel-Ziv Jaccard Distance. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pages 1007–1015, New York, NY, USA, 2017. ACM.

[8] Raff, Edward and Nicholas, Charles K. Lempel-Ziv Jaccard Distance, an effective alternative to ssdeep and sdhash. *Digital Investigation*, feb 2018.

[9] Tomoyasu Suzaki and Kazuhiko Minematsu and Sumio Morioka and Eita Kobayashi. TWINE : A Lightweight Block Cipher for Multiple Platforms. In *Selected Areas in Cryptography*, 2012.

Izzivi izobraževanja in standardizacije na področju mobilne forenzike

Sandra Vizlar
Fakulteta za računalništvo in informatiko
Večna pot 113
Ljubljana, Slovenija
sv9147@student.uni-lj.si

Maša Juras
Fakulteta za računalništvo in informatiko
Večna pot 113
Ljubljana, Slovenija
mj0101@student.uni-lj.si

POVZETEK
Usposabljanja na področju mobilne forenzike so v današnjih časih vedno bolj pomembna. Seminarska naloga predstavlja nekatere izmed izzivov s katerimi se zaradi pomanjkanja leteh srečujejo udeleženci forenzične preiskave. To področje se vedno bolj razvija, tudi kriminalci vedno bolj izkoriščajo mobilne naprave za izvajanje raznih zločinov. Z namenom zagotavljanja boljšega usposabljanja je bilo narejenih tudi že veliko raziskav. V članku [1], ki sva ga uporabili za zgled, je predstavljenih tudi nekaj rezultatov glede kategorizacije usposabljanj ter mnenj o pridobivanju posameznih spretnosti oz. pomanjkljivostih. Naredili sva še kratko analizo predmetnikov na slovenskih fakultetah, ki vsaj delno pokrivajo področje povezano z mobilno forenziko.

Ključne besede
mobilna forenzika, standardizacija

1. UVOD
Kot seminarsko nalogo sva si izbrali članek z naslovom *Law enforcement educational challenges for mobile forensics* [2]. Cilj seminarske naloge je podrobnejša analiza področja in primerjava z našim okoljem. Članek govori predvsem o širjenju znanja s področja mobilne forenzike ter predstavi rezultate opravljenih analiz ter raziskav. To znanje je v današnjih časih nujno potrebno tudi za organe pregona. Pomembno je tudi pravno znanje v forenziki, saj je ob izvajanju vseh forenzičnih postopkov potrebno upoštevati tudi vse zakone.

Najprej predstaviva projekt, ki skrbi predvsem za izboljšanje digitalne varnosti. Nato preletiva področja, ki jih je članek [2] opisal ter poskušava najti vzporednice in primerjave z našim okoljem. Posvetiva se še predmetnikom na različnih slovenskih fakultetah, ki pokrivajo podobna področja ter na koncu predelava izzive, ki jih izobraževanje na tem področju ustvarja.

2. PREGLED PODROČJA
Obstaja ogromno literature v povezavi z izobraževanjem in metodami izobraževanja oz. usposabljanja, vendar je poudarek članka [2] predvsem na potrebah kazenskega pregona po mobilnem forenzičnem usposabljanju. Avtorji raziskujejo trenutno situacijo na tem področju ter opravijo analizo. Proces je vključeval štiri glavne faze, in sicer:

1. pregled literature: zbrani so bili materiali, ki se osredotočajo na digitalno forenzično izobraževanje in usposabljanje, izobraževanje o mobilni forenziki, razvoj in izzive mobilne forenzike ter kazenski pregon
2. zbiranje odprtokodnih informacij: identifikacija, zbiranje in analiziranje obstoječih tečajev mobilne forenzike
3. vprašalnik za predavatelje: kvalitativna raziskava
4. intervjuji: kvalitativni nestrukturirani spletni ali telefonski pogovori

V primerjavi s tem je v Sloveniji [3] še precejšnje pomanjkanje na tem področju. Slovenija nima državne inštitucije za digitalno forenziko, pač pa vse potrebne postopke glede tega izvaja Policija oziroma eden izmed oddelkov za računalniško preiskovanje ali pa Center za računalniško preiskovanje.

2.1 FORMOBILE
FORMOBILE [4] je projekt EU, financiran v okviru programa *the Horizon 2020*. Gre za projekt, pri katerem si prizadevajo izboljšati digitalno varnost in zaščito v EU. Konzorcij FORMOBILE sestavljajo skrbno izbrani partnerji, ki jih koordinira univerza Mittweida. Združuje organe pregona, podjetja, civilne organizacije in akademske inštitucije z različnih področij. Skupno je združenih 19 projektnih partnerjev iz 13 različnih držav EU ter dveh pridruženih držav. Vsak izmed njih je strokovnjak na svojem področju.

Kriminalci uporabljajo mobilne telefone za komunikacijo, koordinacijo, organizacijo in izvedbo ilegalnih dejavnosti, zato je cilj projekta zagotoviti orodja, metode ter razna izobraževanja, ki bodo organom pregona pomagali izslediti kriminalce.

Le-ti uporabljajo najnovejše tehnologije, da jih organi pregona ne bi odkrili pri zagrešitvi kriminalnih dejanj. Najpomembnejše je, da imajo organi pregona varne, zanesljive ter zaupanja vredne načine dostopa, dekodiranja in uporabe podatkov kot dokaz.

Sodelujoči v projektu si želijo razviti popolno verigo forenzičnih preiskav, ki cilja na mobilne naprave. Veriga forenzične raziskave je razdeljena na tri korake:

- pridobivanje podatkov,
- dekodiranje ter
- analiza podatkov.

Rezultat bi nato moral biti celosten pogled nad vsemi področji mobilne forenzike, ki bi omogočal nadaljnje raziskave.

2.2 Izzivi izobraževanja na področju mobilne forenzike

Avtorji članka [1], ki sva ga uporabili za zgled, so izpostavili probleme v izobraževanju, s katerimi se srečamo, ko govorimo o mobilni forenziki.

Eden izmed teh je pomanjkanje znanja o pravilnem zajemu dokazov ter neučinkoviti rabi forenzičnih orodij [2]. *Oparnica* [5] je mnenja, da bi zato morali uvesti 'hands-on' pristop usposabljanja, ki bi moral vključevati vse udeležence forenzične preiskave (forenzične strokovnjake, obrambo in tožilstvo, sodnike ter policiste).

Problem je tudi pomanjkanje samega usposabljanja na temo mobilne forenzike. *Bajramović* [6] izpostavi izzive, s katerimi se srečujemo pri usposabljanju za le-to in sicer:

1. uporaba posodobljenih in realističnih materialov za usposabljanje
2. premalo usposabljanj.

Kröger et al. [7] ter *Paullet et al.* [8] so v svojih člankih predlagali samo vsebino usposabljanj, in sicer:

1. da se usposabljanja fokusirajo na operacijske sisteme mobilnih naprav kot so iOS in Android
2. usposabljanje za uporabo orodij v namen preiskave
3. informiranje o tipih mobilnih naprav
4. postopki obdelave dokazov.

Alva in Endicott-Popovsky [9] pa sta v svojem članku izpostavila, da so tožilstvo, obramba in pa tudi sodniki premalo izobraženi o tematiki digitalne forenzike nasploh. Predlagali so, da se tudi oni udeležijo usposabljanj in tako pridobijo znanje o presojanju sprejemljivosti dokaza.

Humphries et al. [1] so kot problem izpostavili tudi pomanjkanje jasno definiranih standardov oz. postopkov mobilne preiskave.

2.3 Mobilna forenzika kot del predmetnika v tujini

Humphries et al. [1] so izvedli raziskavo o tem, katere korake mobilne forenzične preiskave pokrivajo razna usposabljanja in predmeti na fakultetah. Analizo so izvedli v 30 državah, kjer so pregledali kar 93 tečajev.

V tabeli 1 lahko vidimo, da so področja s slabšo pokritostjo forenzika mobilnih omrežij, dekripcija in enkripcija ter vizualizacija podatkov pridobljenih iz dokazov. Pomanjkanje izobraževanja o sami vizualizaciji podatkov predstavlja slabše razumevanje dokazov s strani organov pregona.

Category	Number of Codes	Total occurrences of Codes
Analysis	173	644
Acquisition	146	948
Automation	64	195
Network	60	183
Security	56	152
Forensic Readiness	46	239
Investigation	39	188
Devices	35	124
Regulative	24	86
Decryption	24	87
Development	24	37
Integrity	24	62
Challenges	20	38
Identifiers	17	104
Communications	16	82
Operating Systems	15	207
Preservation	14	58
Documenting	12	66
Extraction	9	63
Crime Scene	9	48
Evaluation	5	19
Visualisation	3	8
Encryption	3	3
Metadata	2	7
Framework	2	21

Table 1: Analiza področij, ki jih pokrivajo tečaji o mobilni forenziki [1]

3. IZZIVI MOBILNE FORENZIKE V SLOVENIJI

V Sloveniji se ravno tako kot drugje srečujemo s problemom standardizacije. Že sam pojem digitalnega forenzika ni čisto jasno definiran. V splošnem je digitalni forenzik oseba, ki se ukvarja z zbiranjem, shranjevanjem in analizo digitalnih dokazov, vendar pa ni jasno definiranih pogojev, ki jih mora izpolnjevati posameznik za opravljanje tega dela [10].

Selinšek [11] v svojem članku poudari, da tehnološki napredek zahteva stalen razvoj digitalne forenzike (katere veja je tudi mobilna forenzika), kar vodi do hkratnega razvoja dveh področij: tehničnega z razvojem forenzičnih orodij in pravnega s spreminjajočimi pravnimi standardi. Poudari, da dokler se ne najde univerzalne metodologije za validacijo digitalnih dokazov in dokler države ne vzpostavijo certificiranja preiskovalcev, bodo morali odločevalci v kazenskih postopkih (večinoma sodniki) sami odločati o sprejemljivosti dokazov, kar pa pomeni da bodo tudi oni primorani nadgrajevati svoje znanje.

4. MOBILNA FORENZIKA KOT DEL PREDMETNIKA V SLOVENIJI

Tudi pri nas se najdejo programi, ki izobražujejo o mobilni forenziki. V tabeli 2 so predstavljeni rezultati analize predmetnika teh predmetov z različnih fakultet v Sloveniji [12][13][14]. Kot lahko opazimo pokrivajo osnove digitalne forenzike, kjer se fokusirajo na osnovne pojme s katerimi se tekom forenzične preiskave srečamo ter s tem kako zagotoviti verodostojnost digitalnih dokazov. Zajeto je tudi tehnično ozadje, kjer se predstavi predvsem datotečne sisteme in arhitekturo naprav s katerimi se bodo srečevali. Predmetniki vključujejo tudi opis orodj s katerimi se izvaja forenzična preiskava. Študente poučijo tudi o samem postopku forenzične preiskave in o sestavi poročila.

Kategorija področja digitalne forenzike	Obseg kategorije
Uvod v digitalno forenziko	- osnovni pojmi digitalne forenzike
	- digitalni dokazi
	- verodostojnost dokazov
Tehnično ozadje	- datotečni sistemi
	- tipi datotek
	- pomnilniški mediji
	- uvod v računalniško arhitekturo in OS
	- šifriranje podatkov
	- omrežne tehnologije
Postopek forenzične preiskave	- dokumentiranje
	- zavarovanje dokazov
	- vrste analize
Poročanje	- izločanje dokazov
	- sestava poročila
	- pojasnjevanje najdenega
Orodja	- orodja za forenzične preiskave

Table 2: Analiza predmetnikov digitalne forenzike na slovenskih fakultetah [12][13][14]

5. ZAKLJUČEK

V delu smo naredili pregled nad izzivi, s katerimi se srečujejo udeleženci mobilne forenzične preiskave. Glavni problemi so pomanjkanje materiala za usposabljanja in pomanjkanje usposabljanj nasploh. Zanimive so predvsem ugotovitve kako je s predmetniki, ki vsebujejo področje mobilne forenzike po fakultetah. Izpostavili smo katera področja so med bolj pokritimi in katera bi morala biti izboljšana. Strinjamo se s sklepi ostalih avtorjev na tem področju in menimo, da je največji problem na tem področju prav pomanjkanje širine področja večine predmetov na temo mobilne forenzike.

6. LITERATURA

[1] Georgina Humphries, Rune Nordvik, Harry Manifavas, Phil Cobley, and Matthew Sorell. Law enforcement educational challenges for mobile forensics. *Forensic Science International: Digital Investigation*, 38:301129, 2021.

[2] Vikram S. Harichandran, Frank Breitinger, Ibrahim Baggili, and Andrew Marrington. A cyber forensics needs analysis survey: Revisiting the domain's needs a decade later. *Computers Security*, 57:1–13, 2016.

[3] Simon Miklavčič. Organizacija postopkov forenzike mobilnih elektronskih naprav v slovenski policiji. 2012.

[4] Formobile project, 2020. `https://formobile-project.eu/`.

[5] Goran Oparnica. Digital evidence and digital forensic education. *Digital Evidence and Electronic Signature Law Review*, 13(0), November 2016.

[6] Edita Bajramovic. Challenges in mobile forensics technology, methodology, training, and expense. 2014.

[7] Knut Bellin and Reiner Creutzburg. Conception of a course for professional training and education in the field of computer and mobile forensics. *Proc. of SPIE Vol*, 8406:84060W–1, 05 2012.

[8] IMPLEMENTING a SUCCESSFUL TRAIN-THE-TRAINER PROGRAM IN MOBILE FORENSICS AND SECURITY. *Issues In Information Systems*, 2017.

[9] Aaron Alva and Barbara Endicott-Popovsky. Digital evidence education in schools of law. *Journal of Digital Forensics, Security and Law*, 2012.

[10] Jasna Jurc. Digitalna forenzika in digitalni dokazi. 2011.

[11] Liljana Selinšek. Vpliv avtomatizacije digitalnih forenzičnih preiskav na dokazovanje v kazenskem postopku1. *Revija za kriminalistiko in kriminologijo/Ljubljana*, 72(3):219–232, 2021.

[12] https://aips.um.si/predmetibp5/ucnaenotainfo.asp?ueid=29003&leto=2021&jezik=. `https://aips.um.si/predmetibp5/UcnaEnotaInfo.asp?\UEID=29003&Leto=2021&Jezik=`. (Accessed on 05/08/2022).

[13] Digitalna forenzika. `https://fri.uni-lj.si/sl/predmet/63530`. (Accessed on 05/08/2022).

[14] https://aips.um.si/predmetibp5/ucnaenotainfo.asp?ueid=25796&leto=2021&jezik=. `https://aips.um.si/predmetibp5/UcnaEnotaInfo.asp?\UEID=25796&Leto=2021&Jezik=`. (Accessed on 05/08/2022).

6 - Omrežna forenzika in forenzika storitev / Network Forensics and Service Forensics

Digital investigation using WhatsApp

Vanessa Wang
Fakulteta za računalništvo in informatiko, Večna pot 113
Ljubljana, Slovenija
vw6857@student.uni-lj.si

Sophie Normand
Fakulteta za računalništvo in informatiko, Večna pot 113
Ljubljana, Slovenija
sn5232@student.uni-lj.si

Tom Fiette
Fakulteta za računalništvo in informatiko, Večna pot 113
Ljubljana, Slovenija
tf7720@student.uni-lj.si

ABSTRACT

WhatsApp is a messaging app used worldwide by more than a billion people and has become part of everyday life for many as a free alternative to SMS. Therefore, it could potentially give investigators useful information. However, obtaining those information does not come without hurdles because WhatsApp started to have end-to-end encryption in 2016, making real-time insight much harder to obtain. Before 2016, WhatsApp was in plain text XMPP and was not nearly as problematic for digital investigators and analysts. WhatsApp is a double-edged sword for digital forensics: it could be a great source of information (text messages, audio ones, calls, photos files, etc.) but many elements can hinder the access to them due to deletion and end-to-end encryption.

This report gives an overview of the challenges specific of WhatsApp now that it is end-to-end encrypted and explains methods of how one can go about getting real-time as well as non real-time information through WhatsApp. As each approach to get information has both benefits and drawbacks, we will discuss their use and provide conditions that would make some techniques more relevant.

Keywords

WhatsApp forensics, Mobile forensics, Investigation, Encryption

1. INTRODUCTION

While SMS are sent and stored on servers in plain text, WhatsApp has been providing end-to-end encryption to its users since 2016. Thus classic wiretapping cannot be applied when one wants to have live information such who the person is communicating with and what the content of their communication is at the moment. Wiretapping can only give metadata. Alternative techniques exist and are still the subject of ongoing research, as some issues still need to be solved and WhatsApp keeps updating, making some techniques obsolete. The difficulty of obtaining real-time information led to the focusing of WhatsApp investigation on post-mortem investigation.

Several approaches exist to bypass the obstacles faced, sometimes exploiting the fact that WhatsApp can make choices of its design that will prioritize the users' experience rather than their privacy.

First, basic information about WhatsApp will be given in order to understand the methods that will be described for WhatsApp investigation. One section will be dedicated to obtaining real-time information through WhatsApp and another one will be dedicated to non-real time investigation, which is the more common kind.

2. WHATSAPP

This section deals with the characteristic information of WhatsApp useful for understanding the investigation conducted by the investigators.

2.1 Database

When investigating, one is interested in knowing where the messages are stored so that they can be analyzed. Sent and received messages are temporarily stored in the WhatsApp application's servers. So, according to WhatsApp's terms of use, sent messages are stored temporarily on WhatsApp's servers, until the recipient receives the message, then they will be deleted [1]. So, the WhatsApp server contains a lot of useful information or evidence but only for a very short period of time. In order to access this data, investigators must adhere to WhatsApp's privacy policy, which can be difficult. This is why it is important for investigators to exploit the files saved in the mobile device, which are the message log and database. Indeed, the WhatsApp database records all conversations and messages (sent and received) [3].

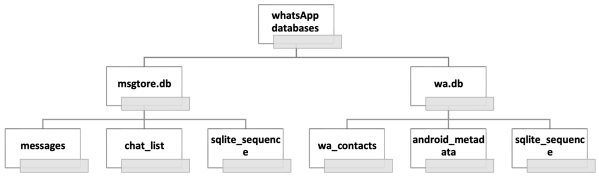


Figure 1: WhatsApp databases structure.[3]

The Figure 1 shows that the structure of the databases is done in a hierarchical manner on three levels, the first contains the main data, the second contains the sub-databases and the third contains the other sub-files.

The chat database is in msgstore.db. This database stores the exchanged messages according to their nature, text and multimedia. msgstore.db consists of three tables with different roles, the three tables are as follows[1]:

- The messages table stores a record for each message sent or received. The record stored includes two sets of information, the message content and its metadata. Examples of metadata are the message type and the media hash. In addition to that, this record also lists other attributes of the message such as the communication partner, message status, timestamp, etc. This is the table that contains the most information.
- The chat_list table stores records about conversations. Each record concerns a contacted user.
- The sqlite_sequence table, generated automatically, only allows the database to function properly. It has no useful purpose in the investigation of WhatsApp. The analysis of this table is therefore discarded by the investigators.

The contact database is in wa.db. This database also contains three tables, with different characteristics. Only the wa_contacts table is of interest to investigators during investigations, as the other tables are not useful for the investigation. The wa_contacts table stores a record for each added contact. This table lists whether the added contact is on WhatsApp or not. In particular, this contact record contains the name, phone number, and WhatsApp ID of the contact.

Part of the investigators' job is to analyze the submitted databases. Keyword indexing is often used to analyze WhatsApp data. For this, a list of keywords is added to a tool, which searches for these keywords in a dataset[2]. One way to collect the data is to capture the traffic between two devices. We extract from PCAP, an application programming interface for capturing network traffic, the network traffic sent and received by the WhatsApp server. We can set up two hotspots to capture the traffic. This traffic can be captured using the packet capture tool Wireshark. This traffic creates two sets of data, data that sends messages and data that receives messages[2].

2.2 Protocols

WhatsApp allows you to communicate in different ways, via text messages, photos and calls. All these means of communication are end-to-end encrypted. Today, the protocol used by WhatsApp for VOIP calls is the STUN protocol[2]. STUN is short for Session Traversal Utilities for NAT. STUN provides the mechanism to communicate with users behind a network address translation (NAT) firewall, which keeps their IP addresses private within the local network (LAN). The network interception allows to identify the use of VOIP at a given time by a person. However, the STUN protocol only applies to WhatsApp phone calls and not to text messages. Therefore, STUN traffic can only be used to analyze a small amount of communication.

2.3 Methodology

WhatsApp does not store decrypted messages or the private keys to decrypt them. Therefore, there is an iterative method regarding the continuous acquisition, analysis and documentation of data:

- Prepare: Determine the make and model of the items that is subject to the investigation, hardware and software that is needed, and people we need for the investigation
- Acquire: Sometimes we don't physically need the smartphone. We can acquire the information remotely. However, the device can also be physically necessary. This step should be adapted according to the needs.
- Analyze: This includes automated or manual analysis. Another question at this stage is whether additional research is needed to obtain more relevant data that contributes to the purpose of the survey.
- Report: The end of the process requires some technical thinking. The report must be complete and clear on the research questions.

After numerous tests to determine the reliability of the investigation tools, the NIST mobile device tool test assertion and test plan was found to be the most relevant test plan for the WhatsApp investigation tool.

NNIST test assertions	Description
Core test assertions	
MDT-CA-01	If a mobile device forensics tool provides the user with an "*Acquire All*" data objects acquisition option, then the tool shall complete the logical/file system acquisition of all data objects without error
MDT-CA-02	If a mobile device forensics tool provides the user with a "*Select All*" individual data objects, then the tool shall complete the logical/file system acquisition of all individually selected data objects without error
MDT-CA-03	If a mobile device forensics tool provides the user with the ability to "*Select Individual*" data objects for acquisition, then the tool shall complete the logical/file system acquisition for each exclusive data object without error
MDT-CA-04	If connectivity between the mobile device and forensics tool is disrupted for a logical/file system acquisition, then the tool shall notify the user that connectivity has been disrupted
MDT-CA-05	If a mobile device forensics tool completes logical/file system acquisition of the target device without error, then the tool shall have the ability to present acquired data objects in a useable format via either a preview-pane or generated report
MDT-CA-06	If a mobile device forensics tool completes logical/file system acquisition of the target device without error then the tool shall have the ability to present subscriber and equipment related information (e.g., IMSI, IMEI, MEID/ESN, MSISDN) in a useable format
MDT-CA-07	If a mobile device forensics tool completes logical/file system acquisition of the target device without error then all supported data elements: PIM data (address book, calendar, notes), call logs, SMS, MMS, chat logs, stand-alone files (audio, pictures, video), application, social media and Internet related data (bookmarks, browsing history), email and GPS data shall be presented in a useable format
MDT-CA-08	If the mobile device forensics tool completes logical/file system acquisition of the target device without error, acquired data containing non-Latin characters shall be presented in their native format
MDT-CA-09	If the mobile device forensics tool completes logical/file system acquisition of the target device without error, hash values are reported for acquired data objects or overall case file
MDT-CA-10	If the logical/file system generated case file or individual data objects are modified via third-party means, then the tool shall provide protection mechanisms disallowing or reporting data modification

Figure 2: NIST specification.[1]

Figure 2 describes the test assertions selected by NIST. It should be noted that the NIST assertions were designed to cover all mobile forensic acquisition tools. Unlike MDT-CA-03, which is concerned with the selection of individual data objects, which illustrates the mechanism followed by WhatsApp acquisition tools, the assertions MDT-CA-01 and MDT-CA-02 are not applicable to the WhatsApp case.

3. REAL-TIME INVESTIGATION

Sometimes historical data is not enough, therefore one should try to get live insight: is that person calling/messaging

someone? If so, whom? What are they saying? In this section, three types of approaches for real-time investigation will be introduced. The first type will be represented by two of Dennis Wijnberg's and Nhien-An Le-Khac's techniques in early 2021 [5] the principle which boils down to impersonating another the suspect that is monitored, the second one relies on public information [5], and the third one focuses on communication patterns [5, 2].

3.1 Impersonating

This subsection will describe two experiments carried by Dennis Wijnberg and Nhien-An Le-Khac[5].

3.1.1 Taking over the WhatsApp account

One way to deal with the issue of end-to-end encryption is simply to take over the monitored suspect's WhatsApp account. When entering the suspect's phone number to log in on a device that does not belong to the suspect (from the point of view of the app, it might be the suspect's new device) , WhatsApp sends a confirmation code by SMS to the number entered. That code is necessary to actually take over the suspect's account, therefore this technique can only be carried out if the confirmation SMS can be intercepted, for example by wiretapping SMS traffic at the telecom provider. As SMS are not encrypted, it is not much of a challenge. However, several risks are to be considered when taking over the suspect's account. Indeed, the suspect may notice that his account was logged off on his phone. To limit that risk, the method has to be applied when the suspect is not using the app because they are on a plane or sleeping. That kind of information can be available thanks to public social media accounts of the suspect's for example. Nonetheless, some WhatsApp features will thwart this method of taking over the WhatsApp account. If someone that sent a message to the suspect has the "security notifications" enabled, they will receive a WhatsApp message notifying that the suspect's security code has changed. In addition, if the suspect has enabled the two-step verification system, then both an SMS verification and an email verification are to be completed. Another similar method of taking over the suspect's account is SIM swap.

3.1.2 Creating a WhatsApp web session

For this method, using the suspect's phone is at some point required. On a laptop for example, it is possible to create a WhatsApp web session, by scanning the QR code displayed on the screen of laptop with the suspect's phone. After that scanning, almost all the WhatsApp data can be retrieved except for phone calls that are neither visible nor recorded, and deleted messages. Contacts, files, pictures, text messages can all be retrieved. Not only has this method the major drawback of needing the suspect's phone, but it also has several risks. Indeed, the suspect may get a notification about the web session or notice an unusual battery drain. Again, this method can also be rendered useless by a WhatsApp security feature (if enabled) that requires a bio-metric verification to create a web session (verification by fingerprint, face or iris scan).

3.2 WhatsApp OSINT (WhatsApp Open Source INTelligence)

Although WhatsApp is end-to-end encrypted, it remains a social medium that can display public information. This simple method has the major advantage of only requiring the suspect's phone number. By saving this number on any phone that does not belong to the suspect, one can know when the suspect was seen last time, their profile picture and the suspect's "about" message. The profile picture can help determine the identity of the suspect with reverse image tools, like Google Reverse Image search, Yandex or TinEye. Microsoft Azure Face API can compute scores of similarity with pictures of persons of interest. it is possible to have higher quality picture by using WhatsApp Web. However, all this information can be hidden depending on the suspect's settings. If one sends a message to the suspect, the latter's name will be displayed, although it could be a pseudonym.

3.3 Sniffing network

Sniffing network can be used to know whether some communication is made over WhatsApp, while remaining discreet.

3.3.1 WhatsApp phone calls

The following technique aims to know whether the monitored suspect is calling through WhatsApp. WhatsApp phone calls are no exception for the implemented encryption: while the content of the call may not be actually retrieved, it can still be useful to know if the suspect is calling someone. The method relies on STUN protocol which is specific to phone calls, thus that approach cannot work for text messages. Unlike the methods mentioned before, the technique of network sniffing will most likely not raise the suspect's suspicion. Dennis Wijnberg and Nhien-An Le-Khac note

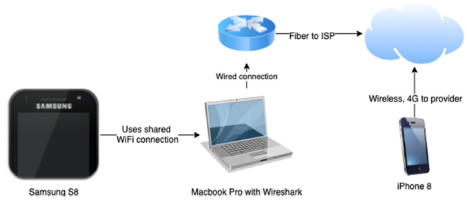


Figure 3: Experiment carried by Dennis Wijnberg and Nhien-An Le-Khac [5]

that it is possible to know the suspect's IP address and find that of the person he is communicating with if they are both in each other's contact list [5, 1, 4]. By using filters when looking at STUN traffic, one can deduce if a call between the two is happening. In the experiment Dennis Wijnberg and Nhien-An Le-Khac carried, the suspect used his phone (Samsung S8) to call person B who is using a cellular connection on a iPhone 8 (Figure 3). A laptop sharing WiFi

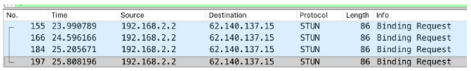


Figure 4: STUN traffic captured by Wireshark [5]

connection with the suspect could use Wireshark to sniff the traffic (Figure 4). If person B is using their home WiFi-connection without VPN, their location could be obtained.

In the case of a cellular connection, the IP address is handled with carrier grade NAT, which means getting the identity of person B is not obvious. Person B may be identified with the network provider's log files. That method is best when the two parties are wiretapped.

3.3.2 WhatsApp messages

In 2020, Rick Cents and Nhien-An Le-Khac published a detailed experiment for sniffing [2] network in which they identified patterns in WhatsApp exchanges of messages with a success rate of more than 85 percent for all the scenarios tested. They experimented with four different devices and

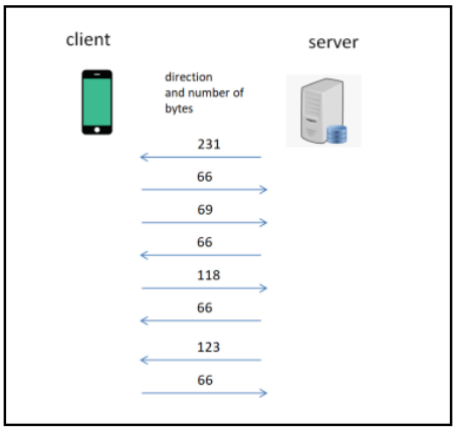


Figure 5: Pattern for receiving and reading directly with packet size in bytes [2]

different situations: sending a message and reading it directly (and not directly), receiving not reading instantly, and receiving reading instantly. One of the patterns is shown on Figure 7. They explained the cases that did not match the identified pattern exactly with TCP retransmission that occurs sometimes.

3.4 Discussion

Law enforcement agencies have a limited scope to work with when it comes to real-time investigation. Methods that exist have high risk of raising the suspect's suspicion and/or do not yield enough information. WhatsApp end-to-end encryption makes content hard to access. Moreover, WhatsApp provides features that reinforce users' privacy, thus a user keen on his privacy only has to go through the WhatsApp settings without advanced knowledge about forensics in order to thwart most plans using the aforementioned methods. The difficulty of live investigation for WhatsApp is the reason why the focus of WhatsApp investigation is usually on post-mortem analysis.

4. NON-REAL TIME INVESTIGATIONS

If sometimes real-time investigations are necessary for the forensics investigators, sometimes they just need to access the data to prove that something happened, for instance a message sent or a proof that two peoples already talked together, etc.. . .

4.1 Where to search?

The main goal of the investigators is thus, to find where the messages are stored, so they can access them and unfold the process. An interesting place to search could be the WhatsApp servers. Indeed, on Figure 6 we see that every transmission is going through the servers. But all sent and received messages are stored for a very limited period of time, and the investigators must respect all WhatsApp's privacy policy and rules, which a very heavy procedure.

Figure 6: How a message on WhatsApp is transmitted [3]

That's why investigating on the sender and receiver's devices is important, and often the valuable place to look for proves. WhatsApp users store many artifacts of high value as proof for the investigation, as the files that are saved are the message log and the database for all conversations and correspondences. The Figure 7 shows the approach through which the data was acquired, as it checks whether the device is rooted or not, and if it is rooted, a logical copy is taken, but if it is not rooted, a physical copy is acquired.

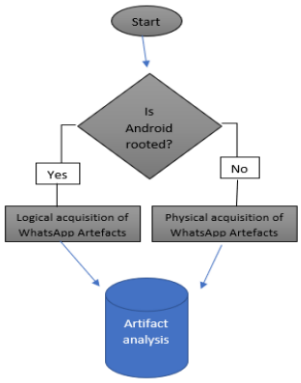


Figure 7: Methodology Design of Acquiring and Analysis of WhatsApp Artefacts [3]

4.2 A typical way to proceed in mobile forensics

The main part of the investigations has been provided on Android Smartphones, often non-rooted, which means that the user does not possess superuser access of the Android OS, so they can not alter or modify the core file system of the OS. In every simulation or try, we follow the method of NIST in digital forensics. All steps were realized so that to obtain valid and admissible evidence which can be presented in a witness report that can be submitted to the court.

4.2.1 Collection

In the collection stage, the physical evidence is identified and collected, which is the Smartphone used in the crime. In this first step, they must state evidence such as name and photo

of the Smartphone, Model Number, IMEI number which uniquely identify a mobile phone in the world, and the OS version.

Table 1: Smartphone evidence specification

Name	Xiaomi Redmi Note 9
Model number	M1901FT4
IMEI Number	123456789123456
OS Version	MIUI Version : 12.5.4 Android Version : 11 RP1A.2000720.0.11

Information on Android smartphones is usually stored in different formats and in different ways, providing some kind of security and confidentiality. Therefore we need to be careful not to change unnecessary information on the device. In this context, investigators make a logical copy by connecting the phone to the computer via a USB cable directly. Then they also install some tools to avoid rooting the mobile and then they are ready to perform examination. Among the tools, Mohamed Shadeed used androidADB, ADB fastboot, java JDK, python to extract the data [3].

At this stage, data and evidence are secured from any change and destruction.

4.2.2 *Examination*

This stage is divided into two stages: the acquisition first, then the data extraction process. MOBILedit and Belkasoft EvidenceCenter X Trial tools were used in the acquisition and data extraction process from the victim's device in Mahmoud Jazzar article [4].

Shadi Zakarneh decided to use Final Forensic Tool with the acquisition method : Logical ADB backup method. This process will only copy files that the user can access and see [6]. During the acquisition progress, we can notice that with Final Forensic tools, it downgraded the version of WhatsApp in order to easier extract and decrypt the database. Then it is upgraded to the actual version.

Back to Mohamed Shadeed method, once the data have been acquired, he needed to know where the WhatsApp data were hidden. After making the backup they found files, including the following :

- /sdcard/WhatsApp/Databases
- /sdcard/WhatsApp/media
- /sdcard/WhatsApp/Backups

In the first folder he found the SQLite database called mgstore.db which contains some data from WhatsApp exchanges, as below : After being able to find the key of the database

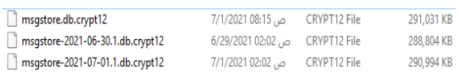


Figure 8: mgstore.db, part of WhatsApp from database [3]

with some tools (DB extractor for instance), we can decrypt the database using WhatsApp viewer and access all the content as shown in the figure[h]. When this process is

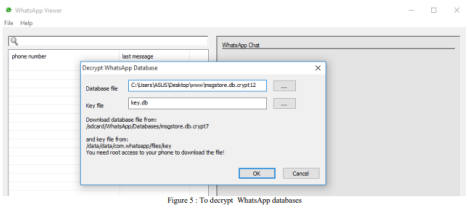


Figure 9: The decryption of a WhatsApp database [3]

complete, the acquisition process information is saved and documented in the report. This information includes investigator information, acquisition time and information, and image information including file name, file size, and MD5 hash of the file.

4.2.3 *Analyze*

Before starting the analysis phase, the investigator verifies the integrity of the evidence image by recalculating the image hash value (MD5) and comparing it to the hash value calculated during the acquisition process. Investigators per-

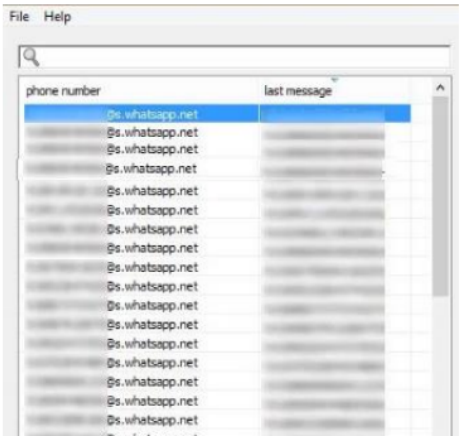


Figure 10: Open WhatsApp database [3]

formed an analysis process using images captured in the previous step to obtain more evidence related to the crime. Evidence is collected and verified by examining and reading conversations stored in the WhatsApp database as shown on Figure 10.

The results obtained are taken from the WhatsApp database and backups on smartphone where text messages were detected and read, as well as properties such as duration, contacts, photos, audio and video clips. Here is an example on Figure 11.

4.2.4 *Report*

After the analysis phase, the results and data of all phases of the investigation are reported. The report contains information and factual introductions about the investigators.

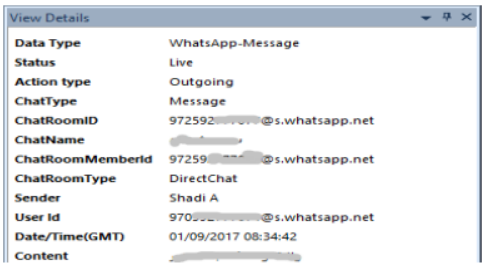


Figure 11: An example of WhatsApp message details [6]

The report contains a document of all the evidence obtained, including a chronological document of the collected events. In this case, we can state all the evidences found in the WhatsApp database, properties such as duration, contacts, photos, audio and video clips, presented in Table 2.

Table 2: Evidence in WhatsApp Database

Mobile Phone Number	+33 6 58 78 ** **
User Name	Sophie
WhatsApp Version	2.20.206.24
Contacts	150
Messages	14414
Deleted messages	350
Calls	220
WhatsApp photos	5896
WhatsApp audio files	48
WhatsApp video files	740
WhatsApp documents	140

All operations must be done within the scope of legal and approved procedures for the report to be accepted by the court.

Through actual digital forensics tools, the investigations on WhatsApp data can be found relatively quickly and provide some evidence for the investigators to present to the court.

5. CONCLUSION

The end-to-end encryption in WhatsApp truly limits the possibilities for getting real-time investigation. However, there are several tools that can help investigators get non-real time information. Other hurdles in WhatsApp forensics include the fact that messages can be deleted and the difficulty of knowing whether artefacts are local or synced.

6. REFERENCES

[1] K. Alissa1, N. A. Almubairik, L. Alsaleem, D. Alotaibi, S. A. Malak Aldakheel, S. B. Nazar Saqib1, and M. Alshahrani. A comparative study of whatsapp forensics tools. *SN Applied Sciences*, October 2019.

[2] R. Cents and N.-A. Le-Khac. Towards a new approach to identify whatsapp messages. 19th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (IEEE TrustCom-20), Cyberspace Security Workshop, December 2020.

[3] M. Shadeed, L. A. Arram, and M. Owda. Forensic analysis of "whatsapp" artifacts in android without root. *Advances in Science, Technology and Engineering Systems Journal*, 7(2):127–132, November 2021.

[4] F.-C. Tsai and E.-C. C. D.-Y. Kao. Whatsapp network forensics: Discovering the communication payloads behind cybercriminals. February 2018.

[5] D. Wijnberg and N.-A. Le-Khac. Identifying interception possibilities for whatsapp communication. DFRWS 2021 APAC, April 2021.

[6] S. Zakarneh. Forensic investigation of whatsapp on android smartphone's. August 2021.

7 - Forenzika pomnilnika / Memory Forensics

Forenzika pomnilnika napadov z USB napravami

Martin Prajnc
Fakulteta za računalništvo in informatiko
Večna pot 113
Ljubljana, Slovenija
mp1238@student.uni-lj.si

Klemen Klemar
Fakulteta za računalništvo in informatiko
Večna pot 113
Ljubljana, Slovenija
kk3365@student.uni-lj.si

POVZETEK

USB naprave ne predstavljajo le fizične shrambe datotek, ki jo lahko imamo vedno pri roki in jo brez težav priklapljamo v različne operacijske sisteme, ampak predstavljajo tudi vektor napada, če ima napadalec fizični dostop do računalnika oz. sistema, ki ga želi napasti. V članku se bomo osredotočili predvsem na forenziko pomnilnika, ki je posledica napada naprave Hak5 Rubber Ducky in Bash Bunny [6]. Napad je bil izveden na računalnik Windows 10, osredotočili pa se bomo na artefakte slik iz sistemskega pomnilnika, ki so bile izvečene s pomočjo orodij *usbhunt* in *dhcphunt*. Za potrebe forenzike je bilo potrebno preveriti tudi DHCP izpise, ki so omogočili vpogled v dejavnost omrežja.

Ključne besede

Digitalna forenzika, pomnilnik, USB naprave, Rubber Ducky, Bash Bunny

1. UVOD

Napadi s pomočjo USB naprav predstavljajo nove in unikatne izzive za forenzike računalniške varnosti. Takšne napade napadalci dokaj lahko izvedejo v kolikor imajo fizični dostop do naprave, katero želijo napasti. Napadi za seboj ne pustijo dosti sledi, saj so te USB naprave načrtovane tako, da skoraj v celoti delujejo na pomnilniku, torej za seboj pustijo na disku le malo oz. tako rekoč nič sledi. Prav tako so takšne USB naprave dokaj poceni, in zelo dostopne na tržišču. Poleg tega pa napadalci, ki jih uporabljajo, za njihovo uporabo ne potrebujejo veliko znanja. Je pa res, da čeprav na disku za seboj ne pustijo veliko sledi, pa je nekaj vseeno pustijo v pomnilniku, kamor naložijo vse ukaze, ki jih nato izvede gostiteljski sistem. Izkaže se, da se ti ukazi lahko izvlečejo iz pomnilnika še dolgo po tem, ko je napad že bil izveden, in da je to je možno tudi ko je gostiteljski sistem ugasnjen. Cilj je raziskati t.i. indikatorje ogroženosti, ki ostanejo prisotni na gostiteljskem sistemu po izvedenem napadu. Primerjali bomo sledi po napadu, oz. kako dolgo sledi ostanejo v pomnilniku in kako se sčasoma spreminjajo glede na dejavnost uporabnika, z dvema najbolj priljubljenima orodjema za izvajanje USB napadov - USB Rubber Ducky in Bash Bunny. Na koncu bomo še predstavili orodja in metode za pridobivanje digitalnih dokazov o delovanju takšnih naprav s pomočjo izpisov iz pomnilnika.

2. DEFINICIJA PLATFORME ZA NAPAD IN OZADJE

Napadalna platforma, ki temelji na USB, se povezuje na žrtvin računalnik preko več vmesnikov, kot je razvidno iz slike 1. Sprva je kompromitirana fizična zaščita sistema, da je lahko USB naprava vstavljena in jo operacijski sistem prepozna, to je seveda možno le, če ima napadalec fizičen dostop do sistema oz. računalnika. Ko se naprava uspešno poveže na sistem, potem lahko poveča svojo funkcionalnost, tako da se lažno predstavi kot poznan tip naprave, npr. naprave za shrambo, HID (angl. Human Interface Device) tipkovnico, omrežni vmesnik ipd. Naprava potem izkoristi svojo lažno identiteto, da opravi škodoželjne dejavnosti na sistemu. Primeri teh aktivnosti so lahko: poizvedovanje, ekstrakcija podatkov in dostava ter izvršitev zlonamerne programske opreme. Po napadu je naprava odstranjena iz sistema.

USB platforma za napad predstavlja zelo unikatno grožnjo, saj je za napadalca zelo preprost proces, ki ne pritegne veliko pozornosti. Proces napada preko USB je avtomatiziran in lahko izvede ogromno dejavnosti v zelo kratkem času, česar napadalec ročno ne bi zmogel. Sled napadalca je prav tako precej hitro zabrisana, saj jo je možno najti le v kratkem času po napadu (nekaj ur, do največ nekaj dni), če sploh vemo kje iskati.

2.1 Forenzika spomina

Začetki uveljavitve forenzike pomnilnika segajo v leto 2005, od takrat je postala tudi nepogrešljiv del znanja za profesionalce informacijske varnosti [1]. V zadnjih letih so orodja za analizo spomina, kot recimo Volatility Framework, zelo napredovala glede zmožnosti pregledovanja [4]. Prav tako je bilo največ napredka na sistemskem nivoju, bolj malo pa je bilo na aplikacijskem nivoju [5]. Temeljita analiza uporabniškega pomnilnika omogoča preiskovalcem, da pridobijo več informacij, kot z uporabo tradicionalnega omrežja za forenziko datotečnega sistema. V pomnilniku se lahko dolgo časa zadržijo občutljive informacije, ki jih uporabljajo aplikacije, kar je pogosto slaba programerska praksa.

Največji izziv pri izdelavi orodij, ki obnovijo forenzične arte-

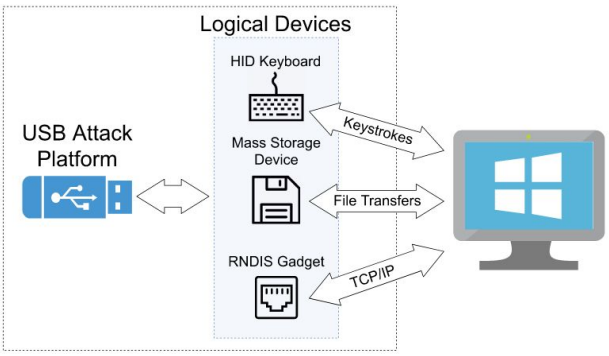


Figure 1 – Model groženj za platformo USB [6]

fakte iz spomina je, da se večina aplikacij precej razlikuje v implementaciji in okoljih izvajanja. Zaradi tega so takšna orodja visoko specializirana in je zanje potrebno premagati velike tehnične izzive pri obratnem inženiringu (angl. reverse engineering).

Ustvarjanje dodatne plasti abstrakcije med procesnim pomnilnikom in analitikom (ali izdelovalcem orodij) je možna rešitev. Nekateri javanski objekti so lahko v celoti rekonstruirani, tudi če so bili pobrani v smeteh (angl. garbage collected). Orodja, ki izkoriščajo prednosti dotične strukture v pomnilniškem izvajalnem okolju, kot npr. Java in JavaScript, kažeta izjemno obetavno izboljšanje forenzike uporabniškega pomnilnika, ki temelji na aplikacijah.

3. METODOLOGIJE

V poglavju 3.1 je najprej na kratko opisano kakšna programska in strojna oprema je bila uporabljena za izvajanje poskusa[6]. V poglavju 3.2 je pridobivanje podpisov, iz katerih se nato s pomočjo programskega orodja Volatility pridobijo podatki za nadaljnje analiziranje. V poglavju 3.3 je opisan razvoj dveh razširitev za programsko orodje Volatility, ki sta bili uporabljeni za procesiranje in analizo podatkov. V poglavju 3.4 pa je opisan način obdelave in vizualiziranja rezultatov.

3.1 Ustvarjanje scenarija

Napad s pomočjo dveh USB naprav je bil izveden na računalniku z operacijskim sistemom Windows 10, ki je bil simuliran na virtualnem stroju VMWare. Nosilca (ang. payload), ki sta bila na USB napravah pa sta javno dostopna na Hak5-ovem GitHub repozitoriju [3] [2], celoten diagram programske in strojne opreme pa je opisan na tabeli 1. Testiranih je bilo več nosilcev, in sicer se je ugotavljalo ali pri katerih sled kaj hitreje oz. počasneje izgine s pomnilnika. Izkazalo se je, da sled na pomnilniku približno enako dolgo, ne glede na to kateri nosilec je bil uporabljen, zato se je v nadaljevanju uporabljalo le en nosilec na posamezni USB napravi. In sicer "Reverse shell" na Bash Bunny USB napravi, in "Document exfiltration" na Rubber Ducky USB napravi. Oba nosilca sta napisana v skriptnem jeziku, ki omogoča simuliranje pritiske tipk na tipkovnici. Tako oba nosilca na gostitelju najprej odpreta PowerShell, kjer vanj hitro vpisujeta ukaze za izvedbo samega napada. En scenarij

Workstation Details			
System Details		Software Details	
Device	Details	Software	Version
Processor	Intel Core i7-875H	VMWare Workstation Pro	15.5.1
System Type	64-bit OS, x64 processor	Volatility	2.6
Virtual Memory (VRAM)	2.00 GB	Windows 10	1903_18362
Bash Bunny	Firmware v1.6		
Rubber Ducky	Firmware v1.0		

Table 1 – Shema podrobnosti uporabljene strojne in programske opreme [6]

je bil sestavljen iz slik pomnilnika, ki so bile pridobljene z začasno zaustavitvijo virtualnega stroja in kopiranjem .vmem datotek iz direktorija tega virtualnega stroja. Te datoteke predstavljajo posnetek trenutnega stanja celotnega simuliranega računalnika, s tem tudi fizičnega pomnilnika, ki ga je mogoče kasneje obnoviti. Omeniti moramo tudi, da je bil pri ustvarjanju in kopiranju teh datotek uporabljen VMWare for Linux, kjer se posnetki trenutnega stanja ustvarijo avtomatično, ko začasno ustavimo virtualni stroj; kar je drugače kot pri VMWare for Windows, kjer se posnetek trenutnega stanja ustvari ročno.

3.2 Pridobivanje podpisov

Ko vemo, kako te USB naprave komunicirajo z operacijskim sistemom, lahko začetno analizo pomnilnika izvedemo z nekaj preprostimi iskalnimi nizi. Shranjena slika sistema služi kot izhodišče za zaznavanje indikatorjev ogroženosti podpisov in pridobivanje potencialnih podatkovnih struktur.

Unix-ovo orodje *strings* je bilo uporabljeno z možnostjo *-td* za pridobivanje nizov s slike in zapisovanje pomika v format, ki je kompatibilen s programskim orodjem Volatility. Izhodna datoteka nizov je bila filtrirana z orodjem *grep*, ki preiskuje nize, povezane z aktivnostjo naprave. Iskalni nizi so vsebovali: IP naslove naprav, permutacije besed "Rubber Ducky" in "Bash Bunny", proizvajalce USB naprav in productID kode naprav. Kljub filtriranju, je ostalo še vedno veliko število zadetkov v izhodni datoteki, katerim je nato programsko orodje Volatility z razširitvijo *strings* razbralo prostore procesov in naslove pomnilnika, na katerem so se nahajali. Prostor na pomnilniku v bližini teh rezultatov je nato bil ročno pregledan s pomočjo razširitve *vorshell*, kjer je bilo določeno ali le-ti vsebujejo očitno podatkovno strukturo, ki smo jo želeli najti. Velika večina teh zadetkov se je nahajala na prostem prostoru v pomnilniku in ni pripadala nobenemu procesu, ali pa se je nahajala med na videz naključnimi biti. Ti zadetki bi bili koristni pri zaznavanju izkoriščenosti naprave, ampak ne zagotavljajo nobenih dodatnih informacij, ki bi bile povezane s časom ali vrsto aktivnosti, ki so se izvajale na tej napravi. Nekaj zadetkov, zlasti dveh znotraj svchost procesov, sta izgledali, da bi lahko vsebovali predvidljivo podatkovno strukturo, ki bi jo lahko pridobili in obnovili. Temu se bomo posvetili v naslednjem podpoglavju, kjer opisujemo kako te podatkovne strukture prepoznati, obhoditi, pridobili in obdelali forenzično relevantne podatke o aktivnosti in delovanju dveh zlonamernih USB naprav.

3.3 Razvijanje razširitev

V podpisih najdeni nizi, opisani v prejšnjem razdelku so lahko dobri indikatorji ogroženosti za odkrivanje sledi napadov na sistem z Rubber Ducky ali Bash Bunny USB napravami. Del tega prispevka se ukvarja tudi s predstavitvijo teh nizov, ki se lahko uporabijo za hitro določanje ali je bila takšna

USB naprava nedavno povezana na računalnik. Takšni zaznani nizi v podpisih so lahko dobra odskočna deska za poglobljeno forenzično preiskavo o tem, kakšne ukaze so zlonamerne USB naprave izvajale na gostiteljskem računalniku.

3.3.1 Usbhunt

Med pregledovanjem rezultatov zadetkov nizov, ki so bili opisani v prejšnjem razdelku, je bilo ugotovljeno, da se v nekaterih velikih JSON strukturah večkrat ponovijo isti nizi proizvajalca USB naprav in productID kode naprav. Po poglobljenem raziskovanju teh JSON strukturah je bilo ugotovljeno, da spadajo pod Microsoftovo telemetrijo in diagnostične podatke v svchost procesu. Celoten seznam potrebnih Microsoftovih diagnostičnih dogodkov je na voljo na njihovem spletnem mestu. Po pregledu seznama telemetričnih dogodkov, za katere Microsoft zahteva, da so omogočeni, je bilo ugotovljeno, da sta z dejavnostjo USB naprav najbolj povezana dogodka *Windows.Kernel.DeviceConfig.DeviceConfig* in *Windows.Inventory.Core.InventoryDevicePnpAdd*, ki sicer nista specifično povezana z zlonamernima Rubber Ducky in Bash Bunny USB napravama, ampak bi ju lahko ustvarila katerakoli USB naprava povezana na gostitelja.

Po branju Microsoftove dokumentacije, smo ugotovili, da dogodek *DeviceConfig* zagotavlja informacije o gonilniku za namestitev gonilnika v jedru sistema. Ker je ta dogodek povezan z namestitvijo gonilnika, bi moral biti ta dogodek prisoten v pomnilniku šele, ko je naprava prvič priključena in so gonilniki že nameščeni. To je bilo v primeru našega testiranja res, torej teh dogodkov ne moremo neposredno povezati z napadi zlonamernih USB naprav.

Drug dogodek, *InventoryDevicePnpAdd*, ki bi lahko bil relevanten pri odkrivanju sledi zlonamernih USB naprav, je v Microsoftovi dokumentaciji opisan, da vsebuje osnovne metapodatke o napravi PNP in z njo povezanem gonilniku za pomoč pri posodabljanju sistema Windows. Te informacije se uporabljajo za ugotavljanje, ali bosta PNP naprava in gonilnik ostala kompatibilna pri nadgradnji sistema Windows. Ta dogodek se torej ustvari vedno, ko operacijski sistem ustvari novo virtualno napravo, in ker ni povezan z začetno namestitvijo gonilnikov, lahko sklepamo, da ostane v pomnilniku tudi po ponovnem zagonu sistema in izvajanju nadaljnjem izvajanju napadov.

JSON strukturi obeh opisanih dogodkov vsebujeta časovne žige v zvezi s povezljivostjo naprav in tudi druge metapodatke, ki so pomembni za forenziko. Več o tem je napisano v naslednjem razdelku 4.

Usbhunt je razširitev za program Volatility, s katerim lahko pridobivamo podatkovne strukture za Windows 10 iz izpisov pomnilnika, ter jih prikazujemo na ukazni vrstici. Orodje deluje tako, da poišče začetni naslov potencialne JSON strukture dogodka. Ko je takšna struktura zaznana, se podatki iz te JSON strukture pridobijo in razvrstijo v seznam z uporabo algoritma, kot je opisan na sliki 2. Moramo pa pripomniti, da ni dovolj, da bi za ta postopek preprosto uporabili algoritem iz standardne JSON knjižnice za obdelavo nizov, saj jo le-ta lahko poškodovana ali okrnjena, torej bi jo standardni algoritem lahko še bolj poškodoval oz. nekatere dele kar prepisal. Ta algoritem za obdelavo JSON struktur, deluje v času O(n), in je bil implementiran kot deterministični končni avtomat, sestavljen iz niza stanj, ki se začnejo z začetnim stanjem. Algoritem se bere podatke in izvaja operacije, ki so pomembne za kontekst trenutnega stanja. Stanje se nato spremeni glede na pogoje iz prejšnjega stanja. Ko je doseženo končno stanje, je bila JSON struktura v celoti popravljena in jo lahko uvozimo v Python kot slovar. Ker je ta algoritem implementiran kot končni avtomat, bo vsak veljaven vhod ustvaril veljaven izhod. V našem primeru je veljavni vhod JSON struktura, ki je bila prepisana oz. okrnjena bodisi na začetku, bodisi na koncu. Ne moremo pa popraviti struktur, ki ima nedotaknjen začetek oz. konec, ampak je bila prepisana ali okrnjena nekje na sredini. Slika 3 prikazuje diagram končnega avtomata algoritma za rekonstrukcijo JSON struktur.

```
1: for char ∈ data do                          ▷ Delimiter stack creation
2:     if isDelimeter(char) then
3:         stack.push(char)
4:     end if
5: end for
6: state ← START, index ← 0
7: while state ≠ END do                        ▷ Begin reconstruction
8:     if not invalidStart(stack) then
9:         state ← PRE-TRUNCATED
10:        rebuildHead(data, stack)
11:    else if prematureEnd(stack) then
12:        state ← POST-TRUNCATED
13:        rebuildTail(data, stack)
14:    else if end of data and isValid(stack) then
15:        state ← END, index ← index + 1
16:    else state ← OK                         ▷ Proceed through data
17:    end if
18:    process(state, data, index)
19: end while
```

Figure 2 – Algoritem za rekonstrukcijo JSON struktur [6]

3.3.2 Dhcphunt

Med iskanjem IP naslovov, povezanih na "reverse shell", je bilo opaženo, da so nekateri zadetki redni in predvidljivi nizi s strukturo, ki je podobna dnevnikom omrežja (ang. network logs). Nadaljnja analiza je pokazala, da so to dnevniki, vsebovani v procesu svchost, ki je odgovoren za zagotavljanje podatkov Windowsovem pripomočku netsh.

Druga razširitev za program Volatility je *Dhcphunt*, ki izvleče te dnevnike in jih prikaže v ukazni vrstici. Večina teh dnevnikov je povezana z aktivnostjo DHCP na gostitelju. Vsi vnosi v dnevnik se začnejo s časovnim žigom ter statičnim nizom, ki se vedno pojavi pred spremenljivimi podatki. Podobno kot pri postopku pridobivanja podatkov pri algoritmu *Usbhunt*, je tudi tukaj prvi korak pri pridobivanju podatkov netsh dnevnikov branje podatkov in iskanje nizov za določitev začetnega naslova niza, ki ga je želimo pridobiti. Ko algoritem najde nek iskan niz v dnevniku, ki ga je mogoče izvleči, to naredi z branjem določenega števila bitov pred in za ciljnim nizom, ter osnovnimi operacijami preoblikovanja niza, tako da le-ta ni poškodovan in tako da ne vsebuje znakov, ki jih ne bi bilo mogoče natisniti. S tem se da učinkovito rekonstruirati celotno časovnico dogodkov, predstavljenih v pripo-

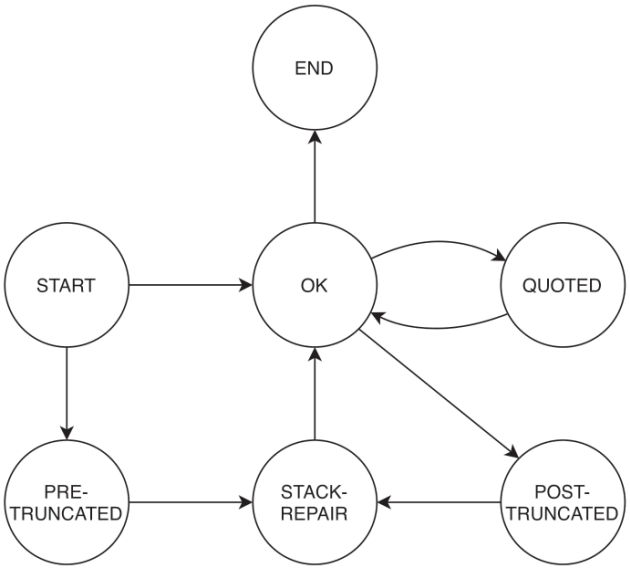


Figure 3 – Končni avtomat algoritma za rekonstrukcijo JSON struktur [6]

močku netsh.

3.4 Zbiranje in vizualizacija podatkov

Da bi ugotovili kako dolgo so bile te strukture prisotne in kako dolgo lahko pričakujemo, da bodo ostale v pomnilniku (če sploh), je bil uporabljen obstoječi framework za pridobivanje in vizualizacijo pomnilnika, predstavljen v Tyler et al., 2020 [7].

S pomočjo nekaj Bash skript in razširitev programa Volatility, so bile slike sistemskega pomnilnika analizirane v rednih časovnih presledkih po simulaciji priklopa zlonamernih USB naprav na računalnik v virtualnem okolju. Za vsako pomniško sliko je bila ustvarjena CSV datoteka, ki je vsebovala izvlečene podatkovne strukture, indikatorje ogroženosti in odmike njihovih naslovov. Ta postopek je bil večkrat izveden, najprej za Bash Bunny za uporabo reverse shell-a, nato pa še za Rubber Ducky za uporabo pridobivanja dokumentov. Zbiranje podatkov je potekalo v obdobju 24 ur. Intervali med pridobivanjem podatkov so bili med eno in dvema minutama, odvisno od tega, kako dolgo je program Volatility izvajal analizo in generiral CSV od prejšnje ponovitve. Ti generirani CSV-ji so nato bili preneseni na vhod v program Volatility, da bi iz njih pridobili vpogled v življenjsko dobo in razpoložljivost podatkov. Enako je bil uporabljen Volatility tudi za vizualizacijo, ki je izrisal dva grafa, in sicer: prisotnosti in celovitosti podatkov na časovni premici (sliki 4 in 5). Prvi graf prikazuje koliko podatkov katerega tipa je bilo v pomnilniku v vsakem časovnem intervalu med zbiranjem podatkov, naslednji pa prikazuje koliko časa podatki ostanejo v pomnilniku preden se prepišejo in izračuna do kakšne stopnje ti sčasoma postanejo poškodovani. Analiza in posledice teh podatkov so podrobneje opisani v poglavjih 4 in 5.

4. UGOTOVITVE

Primer izhoda razširitve *Dhcphunt* za orodje Volatility je razviden na sliki 6 . Primera JSON struktur (ki sicer nista

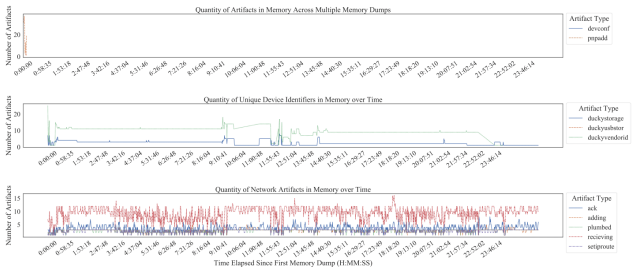


Figure 4 – Rubber Ducky - časovnica podatkov v pomnilniku [6]

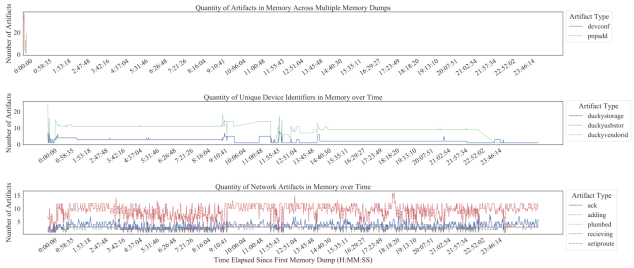


Figure 5 – Bash Bunny - časovnica podatkov v pomnilniku [6]

popolni JSON strukturi, saj so bili zaradi boljše preglednosti odvečni in forenzično nepomembni podatki izpuščeni) pridobljeni z razširitvijo *usbhunt* za orodje Volatility, sta razvidna na slikah 7 in 8.

```
Volatility Foundation Volatility Framework 2.6.1
Address Log Message
------------------ ------------------------------------------------------------------------------------------------------------------------
0x9a9327b 9-21-2020 11:35:55:447Recieving a DHCP message on {d0ed74cd-51ce-4597-91bf-599fb6a2d0ce}. Error code is 0
0x9a935fb 9-21-2020 11:35:55:447Recieving a DHCP message on {d0ed74cd-51ce-4597-91bf-599fb6a2d0ce}. Error code is 0
0x9a93f9b 9-21-2020 11:44:15:426Recieving a DHCP message on {d0ed74cd-51ce-4597-91bf-599fb6a2d0ce}. Error code is 0
0x9a9397b 9-21-2020 11:35:55:463Adding the address of 192.168.232.128 on {d0ed74cd-51ce-4597-91bf-599fb6a2d0ce}. Error code is 0.
0x9a9389b 9-21-2020 11:35:55:463Successfully Plumbed the address: 192.168.232.128
0x9a936dc 9-21-2020 11:35:55:447ACK of 192.168.232.128 from 192.168.232.254 is accepted on {d0ed74cd-51ce-4597-91bf-599fb6a2d0ce}.
0x48d9403b 9-21-2020 11:44:46:946Recieving a DHCP message on {93a38501-7a81-44c9-ad0a-4430f5e690f9}. Error code is 0
0x48d9438b 9-21-2020 11:44:46:962Recieving a DHCP message on {93a38501-7a81-44c9-ad0a-4430f5e690f9}. Error code is 0
0x48d9457b 9-21-2020 11:44:46:962DhcpSetIpRoute: ADD: Dest=0.0.0.0, DestMask=0.0.0.0, NextHop=172.16.64.1, Metric=0, Address= 172.16.64.10
0x48d9481b 9-21-2020 11:44:46:993Adding the address of 172.16.64.10 on {93a38501-7a81-44c9-ad0a-4430f5e690f9}. Error code is 0.
0x48d9473b 9-21-2020 11:44:46:993Successfully Plumbed the address: 172.16.64.10
0x48d9449c 9-21-2020 11:44:46:962ACK of 172.16.64.10 from 172.16.64.1 is accepted on {93a38501-7a81-44c9-ad0a-4430f5e690f9}.
0x4d072d9b 9-21-2020 11:44:45:946Recieving a DHCP message on {93a38501-7a81-44c9-ad0a-4430f5e690f9}. Error code is 121
0x4d07207c 9-21-2020 11:44:15:426ACK of 192.168.232.128 from 192.168.232.254 is accepted on {d0ed74cd-51ce-4597-91bf-599fb6a2d0ce}.
0x71576b7b 9-21-2020 11:35:50:649Recieving a DHCP message on {d0ed74cd-51ce-4597-91bf-599fb6a2d0ce}. Error code is 1223
```

Figure 6 – Primer izpisa razširitve *Dhcphunt* za orodje Volatility [6]

4.1 Indikatorji ogroženosti

Med analizo je bilo najdenih je bilo več zaporedij bitov, katere je mogoče uporabiti kot indikatorje, da je bila na gostiteljski sistem priključena zlonamerna USB naprava Rubber Ducky ali Bash Bunny. Tabela 2 opisuje te kazalnike.

Device	Type	Example
Bash Bunny	Reverse Shell Payload	Q STRING "powershell -W Hidden \"Remove-ItemProperty ...
Rubber Ducky	USB ID	VID_05AC&PID_0220
Bash Bunny	USB ID	VID_F000&PID_FF03
Rubber Ducky	Device Instance ID	USBSTOR\\DISK&Ven_ATMEL&Prod_Ducky_Storage
Rubber Ducky	Hardware Identifier	USBSTOR\\DiskATMEL___Ducky_Storage___1.00

Table 2 – Indikatorji ogroženosti

Vsaka USB naprava ima identifikacijsko številko proizvajalca in izdelka (productID), ki jo operacijski sistemi uporabljajo za identifikacijo, saj tako vedo, kateri gonilniki so potrebni komunikacijo z napravo. V primeru zlonamerne USB naprave Bash Bunny, se kot identifikacijska številka proizvajalca uporablja

```
{
 "name": "Microsoft.Windows.Inventory.Core.InventoryDevicePnpAdd",
 "time": "2020-07-31T16:48:58.7911891Z",
 "ext": {
  "protocol": {
   "devMake": "VMware, Inc.",
   "devModel": "VMware7,1"
  },
  "user": {
   "localId": "w:4D621E97-299E-7C02-BD51-54C81554ED6D"
  }
 },
 "data": {
  "ContainerId": "{70c9a979-a0cd-56e2-afec-26b5c53a8ce3}",
  "ParentId": "usbid_05ac&pid_0220\\123123123123",
  "Description": "USB Mass Storage Device",
  "BusReportedDescription": "HID Keyboard and MSC",
  "ClassGuid": "{36fc9e60-c465-11cf-8056-444553540000}",
  "Manufacturer": "Compatible USB storage device",
  "Model": "USB Mass Storage Device",
  "Inf": "usbstor.inf",
  "DriverName": "usbstor.sys",
  "DriverVerVersion": "10.0.18362.1",
  "DriverVerDate": "06-21-2006",
  "Provider": "Microsoft",
  "DriverPackageStrongName": "usbstor.inf_amd64_c04619397269cc4c",
  "DriverId": "0000d0f4d0ac3c2b29dbc671b22034e209cb410dd9b2",
  "Service": "usbstor",
  "baseType": "Ms.Device.DeviceInventoryChange"
 }
}
```


Figure 7 – Primer JSON strukture indikatorja ogroženosti pri dogodku *InventoryDevicePnpAdd* [6]


```
{
    "name": "Microsoft.Windows.Kernel.DeviceConfig.DeviceConfig",
    "time": "2020-07-31T16:45:56.5111482Z",
    "ext": {
        "device": {
            "localId": "s:B85550C1-8574-4594-92F0-16BD531BF5FD",
            "deviceClass": "Windows.Desktop"
        },
        "protocol": {
            "devMake": "VMware, Inc.",
            "devModel": "VMware7,1"
        },
        "user": {
            "localId": "w:4D621E97-299E-7C02-BD51-54C81554ED6D"
        }
    },
    "data": {
        "DeviceInstanceId": "USB\\VID_F000&PID_FF03&MI_00",
        "FirstHardwareId": "USB\\VID_F000&PID_FF03&REV_0333&MI_00",
        "LastCompatibleId": "USB\\Class_ef",
        "ClassGuid": "{4d36e972-e325-11ce-bfc1-08002be10318}",
        "DriverInfName": "rndiscmp.inf",
        "DriverProvider": "Microsoft",
        "DriverDate": "06/21/2006",
        "DriverVersion": "10.0.18362.1",
        "InstallDate": "2020-09-21T15:44:40.6689571Z"
    }
}
```


Figure 8 – Primer JSON strukture indikatorja ogroženosti pri dogodku *DeviceConfig* [6]

F000 in identifikacijska številka izdelka FF03. S to identifikacijsko številko ni registriran noben proizvajalec, kar pomeni, da so vse naprave, ki jo uporabljamo, obravnavamo kot sumljive in si jih je potrebno podrobneje ogledati. Niz v tabeli 2 prikazuje, kako sta ti dve identifikacijski številki prikazani v pomnilniku, njuna prisotnost v pomnilniku pa posledično pomeni dejavnost naprave Bash Bunny v opazovanem sistemu.

Pri zlonamerni USB napravi Rubber Ducky, pa je malo drugače, saj ta poskuša prikriti ti dve številki, tako da za identifikacijsko številko proizvajalca uporablja 05AC, za identifikacijsko številko izdelka pa 0220. Ta kombinacija identifikacijskih številk je registrirana kot Apple HID tipkovnica. Čeprav ti dve na videz nista sumljivi, pa identifikacijska številka strojne opreme, ki je posredovana operacijskemu sitemu Windows vsebuje opazen podniz "Ducky_Storage", torej lahko iz tega sklepamo na dejavnost naprave Rubber Ducky v opazovanem sistemu.

Podobno so bile po napadih z obema napravama v pomnilniku v čistopisu najdene operacije izvedene v PowerShell-u, kar lahko obravnavamo kot še en indikator ogroženosti.

4.2 DHCP dnevniki v pomnilniku

DHCP dnevniški zapisi predstavljeni v razdelku 3.3.2, pridobljeni z razširitvijo *Dhcphunt* za programsko orodje Volatility, vsebujejo časovne žige in informacije o lokalnih DHCP odjemalcih. Te informacije so koristne predvsem, ker lahko z *Dhcpunt* odkrijemo nekatere zlonamerne USB naprave, ki se pretvarjajo za RNDIS Ethernet pripomočke, ter tako dobijo IP naslov. Seznam najdenih takih dnevniških zapisov je razviden v tabeli 3. Seveda pa to ni popoln seznam vseh dnevniških zapisov v svchost-u, temveč le tista, ki so vsebovala najbolj relevantne IP naslove, ki bi nas v našem primeru zanimali. Celotnega seznama niti ne bi bilo mogoče sestaviti, saj Microsoft nima javno objavljene dokumentacije za takšne dnevniške zapise. Tako v primeru Rubber Ducky kot Bash Bunny, je bilo v pomnilniku med trajanjem poskusa prisotnih več primerkov takšnih dnevniških zapisov. Izkaže se pa, da se ta del pomnilnika med izvajanjem sistemskih procesov ne sprosti, kar s forenzičnega vidika predstavlja zanimiv scenarij. Ugotovljeno je bilo, da se ob generiranju novih dnevniških zapisov, stari prepišejo. Čeprav bi lahko pričakovali, da bi mogoče bili dnevniški zapisi o DHCP odjemalcih v pomnilniku kadarkoli, temu ni tako, saj se le-ti hranijo v njem različno dolgo - od nekaj minut do več ur.

Dogodka *DeviceConfig* in *InventoryDevicePnpAdd* se generirata z namestitvijo gonilnika naprave in povezavo z zunanjo napravo. Ta dogodka se shranita v procesu svchost v JSON strukturi zakodirani v UTF-8 formatu. Te vsebujeta velike količine metapodatkov v zvezi z povezano napravo, vključno s časovnimi žigi in različicami gonilnika. Podatki, vsebovani v teh strukturah, ki bi bili najbolj pomembni z vidika forenzične analize, so podrobneje opisani v tabeli 4.

4.3 Dogodki Windows diagnostic

Nekateri pomembni podatki, ki so bili najdeni s pomočjo orodja za diagnostiko pomnilnika, Windows diagnostic, vključujejo tudi čas, kdaj je bil dogodek ustvarjen, različico namestitve gonilnika, globalni edinstveni identifikator (GUID) uporabniškega računa, s katerim je bil dogodek ustvarjen, productID USB naprave, in GUID od vsebnika, kateremu pripada virtualna USB naprava. Ravno ta GUID vsebnika, kateremu pripada virtualna USB naprava, se lahko uporabi za identifikacijo fizične USB naprave, kar je pomembno, saj lahko nekatere zlonamerne USB naprave, kot je rec-

DHCP Log Messages
Recieving (sic) a DHCP message on ...
ACK of ... from ...
DhcpSetIpRoute: ADD: Dest = ...
Adding the address of ...
Successfully Plumbed the address: ...

Table 3 – Dnevniški zapisi lokalnih DHCP odjemalcev pridobljeni z *Dhcphunt*

Field	Description
DeviceConfig	
- time	Timestamp of event creation
- ext.user.localId	GUID of user
- data.DriverInfName	Device Driver
- data.InstallDate	Driver installation date
- data.DeviceInstanceId	Device vendor and productID
InventoryDevicePnpAdd	
- time	Timestamp of event creation
- device.user.localId	GUID of user
- data.HWID.Value	Device vendor and productID
- data.containerId	Parent device container ID
- data.Description	Vendor supplied description
- data.Manufacturer	Vendor supplied manufacturer
- data.Model	Vendor supplied model information
- data.Inf	Device driver
- data.Provider	Device provider

Table 4 – Windows Diagnostic izpis dogodkov, relevantnih za forenzično analizo

imo Bash Bunny, ponaredijo oz. simulirajo več logičnih USB naprav. V našem primeru, ko smo Bash Bunny uporabili za poganjanje reverse shell-a, je naprava hkrati ponaredila dve logični napravi, in sicer: RNDIS Ethernet vtičnik in HID tipkovnico, saj ji to omogoča, da simulira pritiske tipk in ustvari povezavo s TCP/IP napravo. ID vsebnika imata v tem primeru obe simulirani logični napravi enak, kar namiguje na sumljivo aktivnost naprave s tem ID-jem vsebnika. Na slikah 4 in 5 je razvidno kako dolgo ti podatki ostanejo v pomnilniku, da jih je še mogoče rekonstruirati.

4.4 Dokazi o sledeh

Sled zlonamerne USB naprave Rubber Ducky o izvajanju ukazov v PowerShell-u, je bila najdena v pomnilniku kot del DOS ukaza. V pomnilniku se je pojavil kot niz "`{BACKSHLASH}`", ki je tipičen ukaz Ducky Script besedila - skriptni jezik za programiranje tovrstnih Rubber Ducky naprav. Sledi zlonamerne USB naprave Bash Bunny, ki je bila uporabljena za izvajanje reverse shell-a, so bile prav tako najdene v pomnilniku, a tokrat kot niz pojavil "Q_STRING", ki je tipičen ukaz Bash Bunny besedila, ki je skriptni jezik za programiranje tovrstnih Bash Bunny naprav. Naša analiza se ni osredotočala na specifične nabore ukazov, ki bi jih posamezna naprava ustvarila, saj je le-teh preveč in se med seboj zelo razlikujejo. V Hak5 dokumentaciji je sicer objavljeno veliko primerov takšnih skript, iz česar smo lahko sestavili seznam znanih najbolj koristnih ukazov, ki smo ga uporabili za filter pri iskanju nizov iz vseh zadetkov pri zbranih podatkih.

5. DISKUSIJA

Diagnostični dogodki, ki smo jih pridobili z uporabo orodja *usbhunt* so agnostični strukturam OS in pomnilniškem kontekstu. Z drugimi besedami to pomeni, da ni pomembno, če virtualni opisni naslov (angl. virtual addresss descriptor), ki vsebuje nek niz, še vedno pripada procesu. Če predel pomnilnika, ki vsebuje niz, še ni prepisal nek drug proces, potem so diagnostične strukture še vedno na voljo. To je precej uporabno, če napadalec uporabi proti-forenzični pristop. Torej tudi če napadalec pobriše pobriše *event viewer* in Windows registre, so diagnostične strukture verjetno še vedno v pomnilniku.

Za forenzika so zelo uporabni diagnostični dogodki *DeviceConfig* in *InventoryDevicePnpAdd*, saj vsebujejo pomembne informacije kot so na primer: ID prodajalca in ID produkta, GUID uporabnika, ki je bil odgovoren za dogodek, čas dogodka. S pomočjo teh informacij je forenziku mogoče odgovoriti na ključna vprašanja: Katera naprava je bila priklopljena? Kdo jo je priklopil? Kdaj je bila priklopljena?

Se pa nekatera polja v različnih dogodkih med seboj razlikujejo. *InventoryDevicePnpAdd* ima slednja unikatna polja: ID vsebnika, opis, proizvajalec, proizvajalec gonilnika. ID vsebnika je uporaben, ko skušamo ugotoviti, katero fizično napravo predstavlja priklopljena virtualna naprava. Ostala polja niso tako zanesljiva, saj jih je možno ponarediti. Rubber Ducky je preprosto prepoznati, saj se sistemu predstavi, kot "AMTEL Ducky Storage USB Device".

Edino polje, ki je unikatno pri dogodku *DeviceConfig* je časnovni žig ob namestitvi medija. To je lahko uporabno pri ugotavljanju, kdaj je bila naprava priklopljena v ciljni sistem.

Analiza podatkov, zbranih v eksperimentu, razkriva, da so diagnostični dogodki v pomnilniku dolgo časa. Dogodki *InventoryDevicePnpAdd* so bili v primeru Rubber Ducky v pomnilniku 1 uro (slika 4), v primeru Bash Bunny pa 11 ur. To pomeni, da bi nekatere zelo pomembne informacije o priključeni napravi lahko našli še kar nekaj časa po napadu. Razlike med napravami in količino časa, ko so bili segmenti shranjeni v pomnilniku se pojavijo zaradi razlike v implementaciji. Rubber Ducky ustvari le 1 HID, medtem ko BASH Bunny ustvari 2 logični napravi.

Sodeč po eksperimentu, so bili unikatni identifikatorji naprav najdeni v pomnilniku tudi 24 ur po napadu, kar pomeni, da če bi forenziki prišli do napadenega računalnika in ga analizirali, še preden bi ta bil ugasnjen, je velika verjetnost, da bi lahko izluščili te identifikatorje.

V eksperimentu so avtorji prav tako izmerili stopnjo, do katere so bili prej omenjeni artefakti prepisani v pomnilniku skozi čas. V primeru indikatorjev kompromitiranja (ID instance naprave, Identifikator strojne opreme, ID USB-ja) in Windows-ovih diagnostičnih dogodkov, so ti bili popolnoma nespremenjeni, dokler so ostali v pomnilniku. DHCP zapisi pa so po drugi strani bili zelo nekonsistentni, saj so novo zapisi konstantno generirani.

6. ZAKLJUčEK

Z uporabo Windowsovih telemetričnih diagnostičnih dogodkov je možno pridobiti velike količine metapodatkov, povezanih z USB napravami tudi do 11 ur po izključitvi naprave iz sistema. Izrezovanje dogodkov iz slik fizičnega pomnilnika nam omogoča njihovo obnovo, tudi v primeru, če napadalec uporabi proti-forenzične metode.

Zlonamerne naprave, kot sta Rubber Ducky in Bash Bunny lahko razpoznamo z analizo latentnih artefaktov, prisotnih v pomnilniku po njihovi uporabi. V izvedenih testih so artefakti ostali v spominu več kot 24 ur, potencialno so lahko prisotni, dokler je sistem prižgan.

Prav tako so lahko skripte napada USB naprav izluščena v celoti iz pomnilnika, po tem ko je naprava bila izključena iz sistema. Z uporabo tudi drugih indikatorjev ogroženosti in diagnostičnih dogodkov, se lahko ti artefakti uporabijo, da naredijo celoten obris naprave USB.

7. VIRI

[1] DFRWS 2005 Forensics Challenge — old.dfrws.org. `http://old.dfrws.org/2005/challenge/`, 2005.

[2] Payload backdoor · hak5darren/usb-rubber-ducky wiki. `https://github.com/hak5darren/USB-Rubber-Ducky/wiki/Payload---Data-Exfiltration---Backdoor`, 2018.

[3] Payload reverse shell · hak5darren/usb-rubber-ducky wiki. `https://github.com/hak5darren/USB-Rubber-Ducky/wiki/Payload---reverse-shell`, 2018.

[4] A. Case, R. D. Maggio, M. Manna, and G. G. Richard. Memory analysis of macos page queues. *Forensic Science International: Digital Investigation*, 33:301004, 2020.

[5] A. Case and G. G. Richard. Memory forensics: The path forward. *Digital Investigation*, 20:23–33, 2017. Special Issue on Volatile Memory Analysis.

[6] T. Thomas, M. Piscitelli, B. A. Nahar, and I. Baggili. Duck hunt: Memory forensics of usb attack platforms. *Forensic Science International: Digital Investigation*, 37:301190, 2021.

[7] T. Thomas, M. Piscitelli, I. Shavrov, and I. Baggili. Memory foreshadow: Memory forensics of hardware cryptocurrency wallets – a tool and visualization framework. *Forensic Science International: Digital Investigation*, 33:301002, 2020.

Recovering the master key from RAM to break Android's file-based encryption

Žiga Putrle

ABSTRACT

One of the major challenges in today's digital forensics is how to acquire data from an encrypted storage device without knowing the right decryption key. One possible way to acquire the key, without breaking the cypher, is by extracting a key from the systems memory which might be possible if we have physical access to a powered on device and we are able to make a copy of the memory's content. In this essay, we review and discuss how it might be possible to acquire a master key from a modern Android system that uses File-based encryption.

Keywords

Digital forensics, Computer security, Mobile phone security, Cold boot attack

1. INTRODUCTION

It seems that most modern off the shelf personal computers (PCs) and mobile phones use disk encryption to ensure that the user's data cannot be obtained without the user's permission. This is sensible precaution for a case when a device is lost or stolen, but it can become a problem when digital investigation team tries to analyze the encrypted content. The most straightforward way, and often the only feasible one, to decrypt the data is to acquire the decryption key from the user of the device; but this might not always be possible due to inability or unwillingness of the user to provide the key. In this case, the digital investigators are forced to use alternative approaches to decrypt the data. One such approach is called a *Cold boot attack*.

A Cold boot attack is a collection of methods the goal of which is to obtain the content of the RAM of a powered on device by rebooting the device and dumping it's content. The basic assumption behind the Cold boot attack is that the device might store sensitive data in the memory that is not encrypted; for example, the decryption key that the digital investigation team can use to decrypt the entire disk of the device.

Even though such approaches for obtaining sensitive data from a device are known for a decade (as far as we know, one of the first articles [2] published on the topic was published in 2008), it seems that there is still no way of fully protecting devices against such methods. In the article *One key to rule them all: Recovering the master key from RAM to break Android's file-based encryption* [1], written by Tobias Groß, Marcel Busch, and Tilo Müller, authors explore how memory dump of a device can be used to acquire master key from modern Android systems that uses *File-based encryption*.

2. ACQUIRING RAM'S CONTENT

An interesting property of RAM, that is often not widely known, is that the content of the memory is preserved for a short period of time after the power is cut-off. This is known as *remanence effect* which enables us to obtain the content of the memory when performing Cold boot attack. In general, there are two ways that we might exploit this effect:

- either we reset the device that contains the RAM and boot it with a small program that allows us to copy the content of the memory, or
- we transplant the RAM module to another device that copies the content.

In order to successfully obtain the content of the memory, we might need to overcome several obstacles which we discuss in the remainder of this section.

(**Time limitation**) When the power is cut-off from the RAM, we have limited amount of time to obtain the content of the memory. The time window that we have extends for a couple of seconds until the content of the RAM degrades to such an extent that it is no longer useful. This time can be extended from a couple of seconds to several minutes (where in the extreme cases, also to several hours) if the RAM is cooled down to a lower temperature using as basic tools as air cans (i.e. "canned air") or a freezer. This was studied extensively by Helderman in [2] back in 2008. Figure 1 shows the effect of cooling on the RAM measured as part of an experiment in [2] by Helderman.

(**Location of the key**) Recently, some specialized hardware solutions suggested to keep relevant cryptographic keys in special registers in the CPU and not in the RAM [3]. Making the keys harder or infeasible to obtain. As far as we

	Seconds w/o power	Error % at operating temp.	Error % at −50 °C
A	60	41	(no errors)
	300	50	0.000095
B	360	50	(no errors)
	600	50	0.000036
C	120	41	0.00105
	360	42	0.00144
D	40	50	0.025
	80	50	0.18

Figure 1: Effect of cooling on error rates where devices A, B, C, and D are Dell Dimension 4100 (1999), Toshiba Portégé (2001), Dell Inspiron 5100 (2003), and IBM T43p (2006) respectively. An error is considered to be a change of a bit in a memory. (source [2] - Table 2).

know, such systems are not widely used or not even available. Furthermore, such registers are only meant to store cryptographic keys where other secrets are still kept in the RAM. Therefore, Cold boot attacks are still relevant and for the purpose of this essay we will assume that cryptographic keys are located somewhere in the memory.

(**Encrypted RAM**) One possible way of avoiding exploitation of RAM's content is to encrypt its content. This could be done by encrypting the entire RAM or only parts of it using software or hardware. But a problem with such an approach is that, it slows down memory access for an order of magnitude, and for this reason, such solutions are not widely used.

(**Secure boot**) Somewhat effective solution towards preventing such attack is to restrict which programs can be booted by a device (i.e. secure boot). Nowadays, this is a popular technique used by most of the major vendors of mobile phones, making performing Cold boot attacks on such systems difficult. In such cases, the only way to perform an attack is to avoid the boot restriction (which is often possible, especially if we are able to revert to a previous version of BIOS that might have a flaw), or to transplant the RAM module to another device. Which turns out to also be difficult when it comes to embedded devices, such as mobile phones, because RAM is usually permanently soldered to the board.

Because of such restrictions, it is often hard to obtain the content of the memory, but having the right equipment it is still possible. For the purpose of this discussion, we assume that we were somehow able to obtain a raw content of a device's RAM which we are now able to analyze.

3. FILE-BASED ENCRYPTION

A prevalent way of securing the data stored in the long term storage of a device (PC or mobile phone) is to encrypt the data. There are several ways how one can do that. One way is to use *Full-disk encryption* (FDE) where (almost) the entire disk is encrypted and the user needs to provide a decryption key before kernel can be loaded into memory and user data can be accessed. Where an alternative approach, called *File-based encryption* (FBE), is a bit more flexible and allows to encrypt only individual files while leaving some files unencrypted. This approach has several advantages:

- services being available without unlocking the phone (e.g. alarm clock, receiving calls, ...),
- incremental backup of individual files even though they are encrypted, rather then backing up the entire volume,
- different files are encrypted using different cryptographic keys, and
- cryptographic keys are only kept in memory while files decrypted by them remain open.

Because of this advantages, FBE is often used on modern mobile phones. Support for FBE was added to Android in version 7.0 and completely replaced FDE in version 10.0 and higher. Some devices also use a combination of FDE and FBE to encrypt the data; the data is initially encrypted with FBE using users authentication key and then additionally using FDE with a device bound key.

FBE is implemented as an extension of EXT4 file system that encrypts filenames, metadata and the files content. Each file is encrypted with a different key DEK_{f} which is derived from a masters key *MK*. Usually, each Android device uses at least two different *MK* keys. One that is bound to the device (MK_{d}) and is used to decrypt the files after booting before user unlocks the phone; where the other master keys are derived from users credentials (MK_{c}) after the phone is unlocked and are used to decrypt other files including the users personal data. Typically, *MK*'s are bound to the device using *TrustZone*, which keeps the keys safe before root and kernel level attacks; where the derivation of the keys is performed in TrustZone as well. MK_{c} are usually used to encrypt more sensitive files, e.g. users personal data, for example, chat history.

We can use key derivation function (*KDF*) with file specific nonce and related MK_{c} to derive DEK_{f} for each file; because of that each file and directory in EXT4 has two additional pieces of information: file specific nonce and a key descriptor for the master key that needs to be used for the derivation. Note that nonces and key descriptors cannot be encrypted because we need them to derive the file's DEK_{f}. Figure 2 shows how a file specific DEK_{f} is derived in order to access the content of the file. First (1), a request to access the content of the file is made. By using a key descriptor of the file, an appropriate MK_{c} is selected (2). DEK_{f} is derived (3) using the selected MK_{c} and the file's nonce. The files content is then decrypted with the decryption function (DF) (4) and send to the process that requested it.

4. PERFORMING AN ATTACK ON FBE

In order to be able to decrypt a file on a FBE encrypted disk, we need to obtain the MK_{c} that can be used to derive file specific decryption keys. In order to do that, we can use a weakness in the key derivation function (KDF) of FBE. The KDF was already fixed in the newer versions of the Android kernel. However, the KDF is not updated when phone is updated over wireless connection because the entire phone

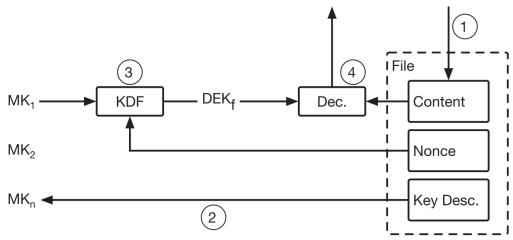


Figure 2: How a file is accessed on FBE storage

needs to be re-encrypted. Because of that, older versions of the phones remain vulnerable.

4.1 Attack model

We assume an attack model where we have a physical access to an Android mobile phone with FBE. The phone is turned on but is locked. This are all reasonable assumptions for an arbitrary case when the police seizes a suspect's phone as part of the investigation and the suspect did not have a chance to turn off the phone. We additionally assume that we were able to make a full copy of the RAM's content and that we have access to the encrypted storage on the phone. As mentioned before, getting access to both RAM and disk can be difficult but it can be done using the right equipment.

5. OBTAINING THE MASTER KEYS

The code of the Android kernel that was investigated is available at Googles code repository[1] under the commit tag ASB-2018-12-05_4.14-p-release. The FBE implementation in this specific commit uses AES256-ECB as a key derivation function (KDF) which is used with a master key and a file specific nonce to derive DEK_f that can be used to decrypt the related file; we denote that as $DEK_f = AES^{ECB}_{nonce_f}(MK_c)$. Note that, AES uses the same function for encryption and decryption. This allows us to also compute a master key MK if we have DEK_f and the file specific nonce, i.e. $MK_c = AES^{ECB}_{nonce_f}(DEK_f)$. This property and the fact that, for each file, its nonce and *MK* key descriptor are not encrypted allows us to derive the MK_c from the image of the RAM with high probability.

In the later version of the Android kernel, a new KDF was implemented that avoids this problem. Which KDF is used depends on a value passed to the function that does the key derivation. Therefore, it is not necessary that the new KDF is always used. At the time of the research, only one from the evaluated phones uses the new KDE (Section 7).

In order to derive the MK_c, we require file specific key DEK_f and nonce_f. We can obtain DEK_f from the RAMs image by using *aeskeyfind*[2] tool, developed by Halderman (2008) [2], which allows us to locate AES keys in the RAM's image. This approach uses several algorithms to find possible candidate keys and filters out unlikely candidates based on the entropy and symbol repetition while also using a combination of some other techniques. An alternative approach towards finding the keys would be to parse the structure in the memory used by the kernel; but it turns out that this method is not as resilient as the one used in *aeskeyfind* because of the RAM's content degradation after the power is cut off, i.e. it is not necessary that we get a precise copy of the devices RAM when performing the Cold boot attack because some bits might degrade (i.e. change value) before we are able to copy the RAM. Further, we can extract file specific nonces by parsing the volume of the copy of the disk storage that contains EXT4 file system. Remember, file nonces are not encrypted because, the kernel needs them to derive the right decryption key. By obtaining all the file decryption keys FK = $\{DEK_2, DEK_1, ..., DEK_n\}$ and all the nonces N = $\{\text{nonce}_1, textnonce_2, ..., \text{nonce}_n\}$ we are able to compute all the potential master keys M = $\{MK_1, MK_2, ..., MK_n\}$ by calculating them as

$$MK_i = AES^{ECB}_{nonce_i}(DEK_i)$$

for all i from 1 to n. The actual master keys are a subset of M. We can determine which ones there are by noting that decrypting content by AES using a wrong key will return random data. Therefore, if we obtain the same master key for two different pairs from FK $\times$ N we have found the actual master key. From this it follows, that we need two different *DEK*, that are derived from the same master key, to be present in the memory to be able to confirm its master key.

There are some additional specific steps in relation to FBE and AES that we don't discuss here because they are not relevant to the main idea behind this approach but are relevant for performing the method. The additional steps can be found the original article [1].

6. DIGITAL FORENSIC TOOLS

In order to use the method introduced in the paper [1], the authors have:

- implemented a new tool called *fbekeyrecovery* that, given a raw image of EXT4 partition and memory dump, is able to extract the FBE master keys as described above,
- they extended *The Sleuth Kit* (TSK) to output FBE related file attributes, and decrypt file names and content automatically, and
- they extended *Plaso* framework to also extract events from FBE encrypted partitions provided a master key.

When analyzing disks encrypted with FDE, the process was usually divided into two stages; first, the content of the disk was decrypted and, second, TSK and Plaso were used to analyze the decrypted content. But in the case of FBE encryption, this separation is harder to achieve because FBE is tightly coupled with EXT4 file system. Because of that, authors decided to extend the functionality of TSK to also be able to decrypt the data during the analysis process.

Figure 4 shows how the modified tools interact between each other.

7. EVALUATION

[1]https://android.googlesource.com/kernel/common
[2]https://github.com/makomk/aeskeyfind

Release	Device	OS Version	Content Enc.	Name Enc.	old KDF	Metadata Enc.
2015	Samsung Galaxy S6	7.0	Full-Disk Encryption		–	–
2015	Google Nexus 6P	8.1.0	AES XTS	AES CBC CTS	✓	✗
2016	Huawei P9 lite	7.0	Full-Disk Encryption		–	–
2017	Google Pixel 2	10	private	AES HEH	✗	✗
2017	BQ Aquaris X	8.1.0	Full-Disk Encryption		–	–
2018	Google Pixel 3	9	private	AES CBC CTS	✓	✓
2018	Xiaomi Mi 8	8.1.0	private	AES CBC CTS	✓	✗
2018	Huawei P20 lite	8.0.0	AES XTS	AES CBC CTS	✓	✗
2019	Google Pixel 4	10	private	AES CBC CTS	✓	✓
2019	Samsung Galaxy S10	10	✳	✳	✳	✗
2020	Huawei P40 Pro	10.1.0	✳	✳	✳	(✓)

Figure 3: Devices included in the evaluation - For each device we can see when it was released (Release), the name of the device (Device), the version of the Android on the device (OS Verison), how long term storage is encrypted (Content Enc.), which encryption method was used (Name Enc.), if a new KDF is used (New KDF), and if metadata is encrypted (Metadata Enc.). * indicates that the authors were not able to access the metadata about the encryption of the device and therefore were not able to perform further evaluation.

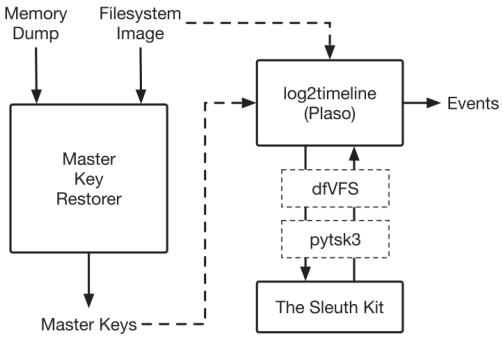


Figure 4: How the modified tools interact between each other

The suggested method was evaluated in two different ways. For two devices, Nexus 5X and Pixel XL, a full master key recovery was done; where for other devices, the encryption scheme was evaluated and, based on that, determined if the method is applicable. Devices from broad range of vendors were included, such as Samsung, Huawai and Xiaomi. For the devices where the master key was recovered, they were able to analyze the content of the storage with the modified TSK and Plaso tools. Figure 3 lists the phones included in the evaluation. From the table we can see that 5 of the evaluated devices use old KDE encryption that allow us to use the suggested method.

8. CONCLUSION

The discussed method, suggested by Tobias Groß, Marcel Busch, and Tilo Müller in [1], demonstrates how one can break the FBE encryption on a modern mobile phone if one is able to obtain a raw image of the device's RAM and storage. As argued, in some cases this is hard to do, but possible using the right tools. The suggested method exploits the fact that:

- key derivation method (AES ECB), used on some Android devices, uses the same function to encrypt and decrypt data,
- FBE does not encrypt the file's metadata used by the encryption (e.g. nonce_f), and
- that hundreds of file specific keys (DEK_f) can be present in RAM that can be used to derive and confirm some of the master keys (MK).

The authors confirm that the method is applicable by applying it to some modern mobile devices available on the market. They also upgrade some of the existing forensic tools (The Sleuth Kit and Plaso) so that the method can be applied in practice.

This article demonstrates that Cold boot attacks can still be performed in practice even on some of the most protected devices (mobile phones) on the civil market. The Cold boot attacks might be much easier to perform on modern day computers where RAM modules can often be unplugged thus allowing much easier access to the raw RAM image.

For those that are further interested in this topic, we highly recommend watching the presentation *An ice-cold Boot to break BitLocker* by Olle Segerdahl and Pasi Saarinen [4] from SEC-T conference. They give a nice overview of of Cold boot attacks in 2018 and also perform a live demonstration of how they can be done.

9. REFERENCES

[1] T. Groß, M. Busch, and T. Müller. One key to rule them all: Recovering the master key from ram to break android's file-based encryption. *Forensic Science International: Digital Investigation*, 36:301113, 2021. DFRWS 2021 EU - Selected Papers and Extended Abstracts of the Eighth Annual DFRWS Europe Conference.

[2] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest we remember: Cold-boot attacks on encryption keys. *Commun. ACM*, 52(5):91–98, may 2009.

[3] T. Müller, F. C. Freiling, and A. Dewald. Tresor runs encryption securely outside ram. In *USENIX Security Symposium*, 2011.

[4] O. Segerdahl and P. Saarinen. An ice-cold boot to break bitlocker (2018). SEC-T conference, URL: https://www.youtube.com/watch?v=RqvPZnLkP70, Accessed: may 2022.

8 - Forenzična orodja in postopki / Forensic tools and process

Generating digital forensic test images

Žan Gostič
Faculty of Computer and Information Science
University of Ljubljana
Večna pot 113
1000 Ljubljana, Slovenia
zg4121@student.uni-lj.si

Mihael Rajh
Faculty of Computer and Information Science
University of Ljubljana
Večna pot 113
1000 Ljubljana, Slovenia
mr0017@student.uni-lj.si

ABSTRACT

With the increasingly diverse landscape of digital evidence, both the experts and tools of digital forensics must be properly certified to deal with it effectively. In order to achieve this goal, quality datasets of digital data must be made available. Manual generation of test images can be very time consuming, and usage of second-hand drives presents privacy concerns. As such, emulating user and system actions presents itself as a valid solution. Several tools have been developed to address this need.

In this seminar paper, we present a few existing approaches for generating synthetic digital forensic datasets, with focus on TraceGen. TraceGen is a novel framework for automating user-generated stories through virtualization technologies. Through the use of Python scripts, it can reproduce various user behaviour. Additionally, we cover select mobile and network forensic dataset generation approaches. We find that each approach has its advantages and drawbacks, and that a unified overview of the field would be of great benefit.

Keywords

data generation, user emulation, forensic images

1. INTRODUCTION

The fast and widespread development of technology creates an abundance of increasingly diverse and complex digital evidence. Incorrect handling or interpretation of this evidence can lead to undermined investigations, wasted resources and abandoned cases in court. Digital experts must not only be able to properly retrieve, preserve, extract and analyse data, but also provide a suitable interpretation and conclusions. The level of certainty with which investigators present their findings is directly influenced by their experience in the field. Organizations also have a vested interest in protecting themselves against liabilities, increasing the expectations for professionals to have relevant knowledge and training [1].

A lack of generally accepted competencies and standards can make it difficult to assess qualifications in digital forensics, leading to the creation of several certification and training programs. In order to provide sufficient general knowledge as well as any specialized certifications, a wide range of high-quality digital forensic test images is required. While common forensic operations are implemented in various tools, these can contain bugs that lead to incorrect or misleading findings. At the rate of change that technology goes through, reliable error rates must be established quickly, leading to additional demand for quality test images [3].

While several collections of digital images have been formed over the years, many are not publicly available or have insufficient diversity [6]. A common approach for the generation of new test images is manual creation with the help of a qualified expert. This process can be very drawn out and leave little room for creating sufficient background noise on the device [7]. It reduces the capacity to test students and new techniques in their ability to cut through irrelevant data. Although this can be resolved through purchase of second-hand devices, privacy concerns come into play.

As a result, many approaches for emulating user and system activities have been proposed. In this paper, we focus on reviewing TraceGen [3], an automated framework for emulating user activity. In the following chapter, we first provide an overview of different approaches to emulating digital activity for disk forensics. Next, we present TraceGen, including a general description of its workflow. Then we separately describe generation of mobile and network traces. Finally, we present our conclusions.

2. OVERVIEW OF DISK IMAGE GENERATION APPROACHES

In their paper Woods et al. [11] describe four properties that are important when creating rich disk images. The first one, which they call 'answer keys', are the ground truth or solutions for students, and tell which digital evidence might be stored in which artefact. The next characteristic, which they call 'realistic wear and depth', states that the generated disk images should have use patterns that are realistic or human-like. In other words, the usage patterns like creating files, browsing the web, etc., should look as if a human has made those actions and not a machine. The third characteristic, called 'realistic background data', states that the 'incriminating' data should not look out of place in a disk image. In other words, 'incriminating' data in the disk image

should not be the only data of its kind, for example the only text file in the disk image. The fourth and final desirable characteristic is what is called 'sharing and redistribution', and describes that the generated hard disk images should be made freely available for download for the purpose of education. In the following subsections we will describe the existing approaches of generating testing disk images.

2.1 Manual disk image generation

The first approach for disk image generation is manual generation, which requires an expert to manually carry out the actions on a disk to create the disk image. A positive side of this approach is that it does create realistic actions, since a human carried out these actions when creating the disk image. On the other hand, this approach has two big drawbacks. The first drawback is that creating a rich history of actions is tedious and time consuming, since the actions have to be carried out in real time, and creating actions in this case would span a long time in days or even months. The other drawback of this method is that the human has to carry out all the actions, which in the case of 10000 actions can be very time consuming.

2.2 Forensig2

One of the systems that was developed is called Forensig2 (Forensic Image Generator Generator), as described by Moch and Freiling [7]. Forensig2 can automatically configure a virtual machine, copy files and install software with the use of the command line. One other thing that Forensig2 does is that it logs the actions that it performs, which can be later used for ground truth. The drawback of this framework is that it does not create artefacts that would be created if a human would open a folder and then manually copy a file. This human action creates numerous registry artefacts that are not generated when you programmatically copy a file.

2.3 Generating synthetic corpora

In addition of copying and deleting files, the approach proposed by Yannikos et al. [12] also includes downloading files from the internet and the import and export of disk images. However, the drawback of this system is that it considers only the file system interactions.

2.4 EviPlant

The approach of EviPlant [8] solves the problem of distribution of disk images with so called 'evidence packages'. It also discusses injecting data into disk images and how the injection of background noise into disk images can be difficult. However, EviPlant does not reduce the need for manual work in generating the initial evidence, only its reusability.

3. TRACEGEN FRAMEWORK

The TraceGen framework presented in the article solves the problems that were presented in other approaches, or at least tries to alleviate them. The framework tries to programmatically simulate user's actions and with that create realistic artefacts, instead of just generating them. The basic idea of the system is to run a script that receives an input csv file containing events, and then run some commands. At the end it generates a disk image and log of actions (the ground truth). The overall design of this system can be seen in Figure 1.

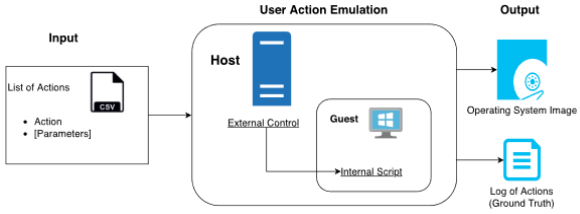


Figure 1: The figure shows the design of the whole system.

3.1 Types of actions

The framework considers two types of actions. The first are machine control actions, or actions that are performed outside of the virtual machine (VM), like powering on the VM, shutdown of the VM, and adjusting the BIOS time. The next type of actions are user actions which are further divided into 'External user actions' and 'Internal user actions'. External user actions are actions that a user performs outside of VM, like copying files, performing google searches etc. And internal user actions are almost the same as external user actions, with the only difference being that they are performed inside the VM.

3.2 Used technologies

The fist technology used by TraceGen is VirtualBox for creating disk images. It was used because of its simple set up and free availability. The next technology used by the framework was the Python programming language, which was chosen because of its simple syntax and long list of supported libraries.

3.3 Automation methods

The article discusses four automation methods used by TraceGen. The first one are APIs, which are used for automating or simulating user GUI interactions, like saving a file in Notepad. One such API is pywinauto. The next method are mouse and keyboard controls, which allow programmatically injecting keyboard inputs, mouse movements and clicks, and are used to simulate the user's mouse and keyboard interactions. The next automation method that TraceGen uses is the GUI interaction, which can programmatically execute mouse and keyboard actions. An example would be programmatically moving the mouse and clicking the save icon. Examples of such tools are 'Siculi' and 'PyAutoGUI'. The last automation technique is browser automation and is used to programmatically simulate a web browsing session.

3.4 Implementation

The implementation of the framework requires implementing the VM external actions. That is the booting of the VM and setting the BIOS time. Implementation of those actions is achieved with the vboxmanage command. The other part of the implementation consists of implementing the VM internal actions like saving a notepad file inside the VM. These actions are implemented using the same vboxmanage command with additional parameters like username, password, name of the application or script to be executed, etc. To simulate user interaction with the browser, they installed and used a program called Sendkey, which allows keyboard navigation of Google search results.

3.5 Stories

To know which commands the Python script must execute, the TraceGen framework uses something they call stories. Stories are a list of actions that are contained in a csv or tsv file. Each of these actions contains the date and time at which to perform the action, the action that needs to be performed, and any argument that the action need in order to be performed. There are two ways that a story can be executed. The first is called 'simulation mode'. This mode performs the given actions in real time. The plus side of this mode is that the timestamps are consistent and real. The drawback of this approach is that because the actions are performed in real time, constructing a month's worth of history would take a month. Another drawback is that the timestamps can only be from the future, since this approach waits until the time to execute the action arises. The other mode is called 'compressed-time simulation', which adjusts the clock of VM before executing each action. The drawback of this approach is that the timestamps of artefacts may not be consistent, but this approach does allow a large amount of artefacts to be constructed in a short period of time.

3.6 Results

In order to evaluate the framework, the article presents two kinds of experiments that they conducted. They used the compressed-time simulation mode of the tool and provided two CSV files containing stories to be executed. Through the use of randomization, they ensured that the time of repeated operations would change.

The first story consisted of four steps, and included setting system time, booting the VM, creating a notepad file, and finally shutting down the VM. The second story was longer with seven total steps, starting similarly with setting system time, booting the VM, and creating a notepad file. Then a png file is copied between folders, the browser is used, and the WinSCP is ran and logged into, before finally shutting down the VM.

Two continuous running tests were also done, where the first one set the time in the past and the other set the time in the future. To examine the traces left on the machine they used Plaso, a Python tool for creating a super timeline. They used pandas to analyse timelines created from test disk images. They recorded the number of events before and after the story has been executed. These results are shown in Figure 2.

Event counts from different sources.

Event Source	Before Story	After Story	Increment
EVT	183,628	246,653	63,025
REG	81,181	125,598	44,417
WEBHIST	27,869	30,689	2,820
Log	2,753	3,160	407
PE	1,019	1,032	13
LNK	621	852	231
OLECF	348	348	0
Total	297,419	408,332	110,913

Figure 2: The figure shows the event counts before and after story execution. The events are as follows: Windows Event (EVT), Registry (REG), Web History (WEBHIST), Log (LOG), Portable Executable (PE), Link (LNK).

To check if generated artefacts are consistent with the human-generated artefacts, they used the following process. First, they took the image of the VM. Then they activated the Procmon capture. Then the automated/human action was executed. Then the Procmon capture was stopped and saved, and the system image of VM was taken again.

After this process was completed, they tested the results by inspecting the Procmon logs and recording the differences in files and registry. It was found that a story can be successfully generated using relatively little user input and in a short amount of time. While some caution must be taken, it is possible to compare the output to real user action. The number of event counts on relevant days was also correctly increased.

4. MOBILE & NETWORK EMULATION

While a number of forensic image generation solutions have been proposed over the years, most are aimed at computer disk forensics. As mobile devices represent a sizable portion of digital technology usage, they form a sizable risk of cyber-attacks against individuals and organizations. It is therefore important to provide comprehensive solutions for generating mobile forensic artefacts [2]. We present two recent methodologies for creating synthetic smartphone test data.

On the other hand, the field of network forensics is interwoven with computer security due to the continuous increase of cyber-attacks. While many mentioned tools support generation of limited network usage artefacts, complete network packet captures are often desirable. Due to the increasing performance of modern networks, machine learning algorithms are employed to detect anomalies. In order to train such algorithms, large, diverse and quality datasets are required, driving up the need for suitable generators [9]. Three relevant frameworks are presented.

4.1 A mobile data generation methodology

In their recent paper [5], Gonçalves, Attenberger, and Baier present a methodology for generating a realistic, content-centric smartphone dataset. They assert three goals that synthetic datasets should meet, the first of which is that it should contain relevant and irrelevant data as specified by a user. Additionally, the dataset should contain a sufficient amount of data to be realistic, and should contain forensically irrelevant data that is automatically generated without being specified. They are currently conducting a survey to determine realistic data distributions.

4.2 FADE

FADE [2], the Forensic image generator for Android DEvices, is based on the hypothesis that realistic datasets can be created through Android emulators and Android API tools. It differs from other generators in that it uses a GUI to insert data into Android devices, allowing the user to launch the appropriate emulator and manually populate it with phone contact, phone call, and text message information, as well as provided files. Their future work will focus on automatically populating devices with Android application data and supporting newer Android API versions.

4.3 ID2T

The first of selected network forensics tools is ID2T [10], and produces labelled datasets with user defined attacks, either through script-based generation or PCAP modification. This allows the implementation of both simple and complex attack models. In one case, a large number of packets with similar parameters are inserted, while in the other, a relevant pcap file is provided with user specified replacement parameters.

4.4 Encapcap

Encapcap [9] instead focuses on generating virtual network packet captures from existing capture files. They reason that virtual networks play an increasingly important role in modern environments, and differ from physical networks to an extent where detection rates of existing machine learning algorithms is impacted. After parsing the original pcap file, the tool creates encapsulated virtual network information based on the provided configuration.

4.5 Hystck

Finally, the hystck framework [4] is designed to generate not only relevant network traffic, but other digital evidence, through simulating human-computer interaction. This framework is similar to TraceGen in that it uses virtualization technology in their framework. Through a provided VM template, it uses a local network interface to trigger relevant application behaviour, including Firefox and Thunderbird. If specified, it will produce a persistent disk image besides the resulting tcpdump capture.

5. CONCLUSIONS

In this seminal paper we reviewed the problem of generating digital forensic test datasets. The approach is relevant due to the drawn out nature of manual artefact creation, and the privacy concerns relating to capture of second-hand material. TraceGen is a novel system that uses virtualization technology as its approach.

Previous existing tools for emulating user actions include Forensig2, EviPlant, and the framework proposed by Yannikos et al. While these approaches are relatively successful, they each have certain drawbacks, namely a difficulty in creating artefacts resulting from complex user-system interactions. They mainly consider action emulation through system interactions alone.

The TraceGen framework instead proposes the use of VirtualBox disk images to automate GUI interactions using APIs, mouse and keyboard control, GUI interactions, and browser automation. This is implemented mainly through the use of Python scripts that execute user-generated scripts called 'stories'.

In comparison to these approaches, mobile and network forensics face slightly different issues. Due to the diverse mobile device technologies, only one complete Android image generator was found for inclusion in this paper. Network forensic tools are instead focused on generating network traces, and provide various tools for modifying or generating specific network captures.

In the future, we hope to see more work focused on evaluating and comparing existing digital forensic image generation approaches. As the number of widely used digital technologies increases, so does the need to provide increasingly diverse and scalable datasets. A unified overview can have immense value for this purpose.

6. REFERENCES

[1] E. Casey. *Digital evidence and computer crime: Forensic science, computers, and the internet.* Academic press, 2011.

[2] A. A. Ceballos Delgado, W. B. Glisson, G. Grispos, and K.-K. R. Choo. Fade: A forensic image generator for android device education. *Wiley Interdisciplinary Reviews: Forensic Science*, 4(2):e1432, 2022.

[3] X. Du, C. Hargreaves, J. Sheppard, and M. Scanlon. Tracegen: User activity emulation for digital forensic test image generation. *Forensic Science International: Digital Investigation*, 38:301133, 2021.

[4] T. Göbel, T. Schäfer, J. Hachenberger, J. Türr, and H. Baier. A novel approach for generating synthetic datasets for digital forensics. In *IFIP International Conference on Digital Forensics*, pages 73–93. Springer, 2020.

[5] P. Gonçalves, A. Attenberger, and H. Baier. A methodology to create synthetic forensic smartphone test data for research and education.

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

[7] C. Moch and F. C. Freiling. The forensic image generator generator (forensig2). In *2009 Fifth International Conference on IT Security Incident Management and IT Forensics*, pages 78–93. IEEE, 2009.

[8] L. D. Scanlon M., Du X. Eviplant: An efficient digital forensic challenge creation, manipulation, and distribution solution. In *EviPlant: An Efficient Digital Forensic Challenge Creation, Manipulation, and Distribution Solution*, pages S29–S36. Digit. Invest. 20, 2017.

[9] D. Spiekermann and J. Keller. Encapcap: Transforming network traces to virtual networks. In *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*, pages 437–442. IEEE, 2021.

[10] E. Vasilomanolakis, C. G. Cordero, N. Milanov, and M. Mühlhäuser. Towards the creation of synthetic, yet realistic, intrusion detection datasets. In *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*, pages 1209–1214. IEEE, 2016.

[11] K. Woods, C. Lee, S. Garfinkel, D. Dittrich, A. Russell, and K. Kearton. Creating realistic corpora for forensic and security education,(2011) adfsl conference on digital forensics. *Security and Law*, 2011.

[12] Y. Yannikos, L. Graner, M. Steinebach, and C. Winter. Data corpora for digital forensics education and research. In *IFIP International Conference on Digital Forensics*, pages 309–325. Springer, 2014.