INCIDENT RESPONSE MODEL FOR PROACTIVE MALWARE DETECTION IN SMART DEVICES

By

Amna Hameed

Fall 2021 – MS (IS) - 00000364734

Supervisor

Dr. Mehdi Hussain

Department of Computing

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Information Security (MS IS)

In

School of Electrical Engineering and Computer Science,

National University of Sciences and Technology (NUST),

Islamabad, Pakistan.

(December 2023)

Approval

It is certified that the contents and form of the thesis entitled "Incident Response Model for Proactive Malware Detection in Smart Devices" submitted by Amna Hameed have been found satisfactory for the requirement of the degree

Advisor : Dr. Mehdi Hussain

Signature: [illegible]

Date: 29-Nov-2023

Committee Member 1:Dr. Muhammad Zeeshan

Signature: [illegible]

29-Nov-2023

Committee Member 2:Dr. Sana Qadir

Signature: Sana Qadir

Date: 28-Nov-2023

Signature: ______________________

Date: ___________________________

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS/MPhil thesis entitled "Incident Response Model for Proactive Malware Detection in Smart Devices" written by Amna Hameed, (Registration No 364734), of SEECS has been vetted by the undersigned, found complete in all respects as per NUST Statutes/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial fulfillment for award of MS/M Phil degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have also been incorporated in the said thesis.

Signature: [illegible]

Name of Advisor: Dr. Mehdi Hussain

Date: 29-Nov-2023

HoD/Associate Dean: ____________

Date: ____________

Signature (Dean/Principal): ____________

Date: ____________

Dedication

I dedicate this thesis to my ***Parents*** and ***Siblings*** for their endless prayers, love, and encouragement.

Certificate of Originality

I hereby declare that this submission titled "Incident Response Model for Proactive Malware Detection in Smart Devices" is my own work. To the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST SEECS or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEECS or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics, which has been acknowledged. I also verified the originality of contents through plagiarism software.

Student Name:Amna Hameed

Student Signature: [illegible]

Acknowledgment

First of all, I would like to thank Allah, the Almighty for giving me the ability and strength to carry out this research. My deepest gratitude to my supervisor *Dr. Mehdi Hussain* for his continuous support and guidance during my thesis. I could not have imagined having a better supervisor and mentor for my master’s degree. I am also thankful to my teachers for providing me with an academic base, which enabled me to complete this thesis.

I am thankful to all my fellows and friends for their support and motivation. Last but not least, I would like to thank my parents for their endless prayers and support throughout.

Table of Contents

List of Tables

List of Figures

Abstract

Extensive usage of the Internet is increasing the risk of malware attacks on smart devices. Implementing security controls in these devices is challenging due to their limited processing and computation power. Different methods detect malware in smart devices through live forensics, memory analysis, and timeline reconstruction. However, these solutions provide only a limited number of artifacts and techniques. There is a need for a forensic investigation model that identify the most suitable set of paths and artifacts to detect the malware presence effectively. This study proposed an incident response model for detecting malware by employing a digital forensic methodology. The proposed model consists of three phases: proactive, reactive, and forensic process. The study extends the smart device forensic process into four modules (1) acquire & extract, (2) detect, (3) investigate and, (4) validate & report. The experiments are conducted on Android devices with the latest APKs malware. The proposed model carefully examined and identified 11 different folder paths such as /data/data, /data/app, /system/app, /system/data. These paths contain useful artifacts for investigation. The systematic examination of paths and corresponding artifacts helps to construct the timeline of APK download URI, installation, traces, activity, intent, and system permissions acquired by user-installed applications. The proposed model also correlates the changes in system paths and files made by different user-installed applications. Similarly, the proposed system is capable to identify the user-installed malware and benign applications. To prove the effectiveness of results these suspicious applications are verified by Cuckoo Sandbox for validation purposes.

Chapter 1

1. Introduction

This chapter elaborates the overview of basic concepts, significance, and history of research work. This chapter describes the road map of the thesis and briefly highlights the further organization and structure of the thesis. This chapter explains the motivation for carrying out the research work. This chapter also gives an idea about the vital contributions, prominent benefits, scope of the work, and key objectives of the thesis.

1.1. Overview

The term "Smart Devices" typically refers to hardware and other items that could potentially read, recognized, located, addressed, and/or controlled online. Over thirty billion smart devices associations, over four smart devices per person on a typical basis, and trillions of sensors linking and communicating on these devices anticipated by 2025. Each second, 127 new gadgets link to the internet, reports The McKinsey Global Institute. There are more than thousands of smart devices in existence, and defending a system with such huge attack surface is not a simple task—especially given the wide range of device kinds and standards of security. Regarding those billions of smart devices, the consensus from an information security operations perspective is that everything connected can be exploited[1].

Every smart device is a potential attack surface via which attackers might access data. Malware that targets smart devices or other connected devices is on the rise because these devices are constantly linked to mobile smartphones or other computing gadgets through the Internet. According to a recent study[2], the most susceptible gadgets involve video-streaming devices, linked cameras, PCs, cellphones, and tablets.

Additionally, the majority of smart devices have lower storage and processing capacities than smartphones and laptops. Because of this, using firewalls, anti-virus software, and various other security tools that may help safeguard them is challenging. Edge computing also makes local data an attractive target for skilled threat actors by aggregating it.

In addition to the hardware of smart devices, ransomware may attack apps and data. According to Check Point Studies, the typical daily amount of ransomware assaults rose by 50% in the third quarter of 2020 compared to the first half of the year[3].

Smart device botnets are an illustration of how vulnerabilities in devices affect users and how hackers have learned to exploit them. Mirai is well-known smart device botnet malware strains, hacked home smart devices network to launch a distributed denial of service (DDoS) operation. The introduction of smart gadgets into the home might create new entry points into a setting with questionable security, exposing staff to viruses and assaults that can infiltrate a company's network. When establishing work-from-home and BYOD policies, it is an important factor to consider.

Smart devices with known flaws can potentially be used by attackers to access inside networks. These dangers vary from DNS rebinding attacks, which enable the collecting and exfiltration of data from private networks to fresh side channel assaults, such infrared laser-initiated attacks targeting smart devices in residential and commercial settings[4].

1: Daunting challenge of securing IOT: https://www.forbes.com/sites/chuckbrooks/2021/02/07/cybersecurity-threats-the-daunting-challenge-of-securing-the-internet-of-things/

2: Report on threats found in IOT: https://www.techrepublic.com/topic/security/

3: IOT security trends: https://www.itprotoday.com/iot/iot-security-trends-2021-covid-19-casts-long-shadow

4: How IOT device vulnerabilities effect users: https://www.trendmicro.com/vinfo/us/security/news/internet-of-things

1.2. Thesis Motivation

This research is focused on malware detection using forensic investigation methodology. In literature, different malware detection approaches are employed by the various techniques [1-9, 11-14, 17-18] with their merits and demerits but there is research gap for detection of malware in smart devices. Security measures and detecting malware might, however, be difficult at times. Dohyun Kim et. al. proposed an incident response framework for smart device malware detection through digital forensic investigation [1].

The motivation behind this work is to improve the efficiency of forensic methodology by identifying useful artifacts and paths to detect malware among other benign applications. Further, the suspicious applications are validated based by cuckoo sandbox.

The proposed research aims to provide two significant perspectives. First, a method to identify and analyze multiple latest malwares by digital forensic investigation. Secondly, to suggest new artifacts and paths useful for forensic investigation for smart devices.

1.3. Research Objectives

In this study, we perform an in-depth analysis of artifacts and paths of android applications to create new insights and explore malicious behaviors. The main objectives of this research are as follows:

1. To study the existing smart device-based malwares and its forensic artifacts

2. To propose an incident response investigation forensic model for smart device.

3. Detect multiple malwares in real-time smart devices.

1.4. Research Questions

This section describes the research questions listed here are devised to perform this study:

- **Why is this research required?**

  Smart device ecosystem is continuously being threatened by malware which poses many security risks to the user's data. Since this data is usually of great value to the users, the users wanted to protect it.

  There already exists several detection methodologies, each providing their own benefits to the community. There is a need to investigate whether these can be employed with the latest or modern smart device-based devices for the detection of malware.

- **How much importance does studying have? And what procedures are followed in the study?**

  This study highlights the importance of malware detection in smart devices. The purpose of the study is to classify malicious applications from benign applications using forensic artifacts. It will help malware analysts and the research community to quickly identify the malicious applications. The study performs qualitative as well as quantitative detection of malwares. We have essentially split our process into the following four phases to conduct the study:

  1. Collection of the most recent benign and malicious samples and design environmental setup
  2. Obtain a forensic image of a smart device.
  3. Analyzing and detecting forensic paths and artefacts by autopsy.
  4. Verification of suspicious applications as malware using cuckoo sandbox.

What are the aims of this study?

The mainly focused aims are as follows:

a) Proposing forensic investigation model to identify the latest malware from benign applications.
b) Identifying the newer artifacts and paths useful for efficient malware detection.

1.5. Problem Statement

In literature, most of the existing techniques detected the malware through the digital forensic investigation in the smart devices. However, existing techniques are unable to identify multiple and latest malicious applications along with other benign applications. In addition, the existing investigation techniques are based on conventional and limited number of artifacts. There is a need for research to propose a comprehensive incident response model to readily detect the malicious applications in smart devices. The model correlates the changes in the system paths and artifacts made by malicious and benign applications in smart devices.

1.6. Solution Definition/Description

The research provides an efficient malware detection approach using forensic investigation of smart devices. In this research, the physical image of smart device is extracted and analyzed by various tools i.e. Autopsy. Furthermore, the artifacts are collected from targeted paths such as application data paths, and system data paths. This model is used to trace malware presence and distinguish them from benign application. The suspicious applications detected by this methodology are verified by Cuckoo sandbox.

1.7. Thesis Contribution

The proposed research methodology successfully explored and improved the detection of malware by introducing new artifacts and paths. The contributions of the proposed methodology are as follows:

- To **detect the presence of multiple malwares** installed in real-time smart devices.
- **Proposes most suitable forensic artifacts and paths** for effective investigation of malware activities in smart devices.
- The study will **correlate the changes** in the paths and directories of smart devices by comparing benign and malicious applications.

1.8. Thesis Organization

The thesis organization is presented as follows. Chapter 2 throws light on previous work done related to detection of malware applications. In chapter 3, the research methodology followed during the research has been discussed. The experimental setup is discussed in Chapter 4. Chapter 5 showcases the result of the experiment. This section also discusses the activities/events performed during the results collection. Chapter 6 is dedicated towards the discussion and analysis of the experimentation results. Lastly, chapter 7 sheds light on the conclusion with possible directions for the future.

1.9. Summary

In this chapter, basic concepts are discussed regarding malware analysis by digital forensic investigation for the detection of malicious applications. It provides an overview of the aim and scope of the thesis and presents the objectives of the research work with the overall thesis organization. In the next chapter, we will look at the literature review that has been conducted.

Chapter 2

2. Literature Review

Chapter 2 discusses the related work and terminologies. The related work is the research carried out by different researchers over the years which is related to the work done in this thesis and contributed towards making a new solution.

2.1. Overview of Malwares

The biggest danger to smart devices is malware, which has the potential to either damage the gadget or, in some situations, transform the system into one that is privileged and controlled by the attacker [2]. Such viruses can open a backdoor for other assaults because of their ability to function independently. Grayware and Madware both provide serious security risks in a similar way. Grayware, which among other infections include dialers and adwares, cannot be deemed harmful but can nevertheless carry out undesirable acts that impair the functionality of the device. Madware, on the opposite hand, utilizes aggressive and targeted pop-up advertisements to gather data from a user's device [2]. According to data on cyberattacks, financial malware, rootkits, logic bombs, ransomware, bots, worms, viruses, and trojans are the most well-known types of malwares. A rootkit is a form of malware that an attacker may gradually access with the aim of taking control of the system. In order to eradicate the present infection, ransomware viruses might lock the user's device or software and demand payment from them. The "screen locker ransomware" mentioned before may disable an

Android-based smart TV. Bots are intended to infiltrate a device and are a sort of malware that spreads itself. These malware threats then establish a connection with a server, commonly referred to as a "bot master," which serves as the main command and control center for infected devices. Financial malware attempts to gather details about an account from a device or through faulty banking websites. Code blocks inserted by an intruder into a network are known as logic bombs. These programmed operations have the potential to damage the system when they are activated, either by erasing data or by causing circumstances that might lead to the system's total demise. Software that runs on computers is how virus and malware software spreads and can destroy a system. The user's activity is required for a malicious program to be installed and propagated on a device (by initiating it through an executive program). In contrast to viruses, worms may propagate without the user's involvement and can function on their own. Worms, on the contrary hand, spread through networks. Trojans are a category of malware that enters the computer system by compromising user information and identity [3]. *Uapush.A, Kasandra.B*, and *SMSTracker* are three of the mobile phones malware that are most often installed in mobile devices [4]. A mobile device's data can be stolen by the malware Uapush.A via an SMS. Another virus that resembles a security program is Kasandra.B. A mobile phone's sensitive data, such as logs, passwords, history, etc., can be accessed by Kasandra.B. With the help of the Android software SMSTracker, hackers may fully observe all of a mobile device's traffic-related features, including SMS, phone calls, and other communications. A "screen locker ransomware" has also been reported to have the ability to disable an Android-based smart TV [4]. Finally, a virus known as "Mirai" has been found to have compromised a large number of smart devices, including routers, IP cameras, printers, DVRs, etc. By scanning the default usernames and passwords, it targets smart devices. [5]

2.2. General Malware Detection Approaches

Over the years smart devices have become a popular ecosystem. Owing to their extensive use and popularity, smart devices has previously been a subject of many studies from different perspectives including security research like malware detection. There is enormous literature for malware detection using different techniques, but the research lacks the detection of malware from digital forensic investigation. In this section, we will examine the currently available literature of malicious app detection under such analysis schemes.

Malicious application detection methods are mainly categorized into three types: static analysis, dynamic analysis, and hybrid analysis.

a) *Static Analysis*

Static analysis is carried out in a non-runtime context and focuses on looking at the source code, byte code, or application binaries, as well as on looking at the meta data and supplementary information for any indications of security flaws [10]. A wide range of methodologies and approaches are used in static analysis to identify a software's runtime characteristics before it is executed. In a security setting, the goal is obviously to separate out dangerous or repackaged programs before they are installed and used.

Since it requires far fewer resources and time, static analysis is frequently used as a malware detection method and is seen as an effective mechanism for market protection. The design is a fairly quick detection approach that is advantageous for Android smartphones with limited resources [50].

b) Dynamic Analysis

The study's dynamic component examines how programs behave during their execution phases when tested against certain test cases. The study seeks to spot harmful actions that occur after applications are deployed and run on actual or simulated devices. The hidden goals of malware software can be retrieved through dynamic analysis. To differentiate among dangerous and benign applications, this analysis frequently necessitates some human or automated engagement with apps and gathers data on network activity, processor execution, system calls, SMS sent/received, phone calls, etc.

The data obtained through dynamic analysis accurately reflects the real purposes of the program. Nevertheless, although being a useful tool, the execution of dynamic analysis consumes extensive resources [50].

c) Hybrid Analysis

Hybrid Analysis utilizes the mixture of static analysis and dynamic to perform the study thereby increasing the detection accuracy. Given that it examines both the installation files and the behaviors of the apps, it is regarded as the most thorough analysis since it combines the benefits and drawbacks of both analysis kinds. However, like dynamic analysis, hybrid analysis is also subjected to extensive resource utilizations.

Static analysis is beneficial for time and resource constrained environments. On the other hand, although accurate for detection, the dynamic methodology requires extensive application study and has a significant processing cost. Furthermore, unlike the static technique, the analysis is done after the APKs have been run. For this purpose, static analysis is quicker and useful in creating a preliminary understanding of the APKs depending on their anticipated behaviors.

2.3. Smart-device Malware Detection Approaches

Forensic investigation methodologies are followed to detect malware from desktops, mobile and smart device. In [1] authors aimed to develop a digital forensic incident response framework for the detection and analysis of the malwares (i.e., phishing, smishing, vishing or APT attack) for Android. The experiment is carried out to investigate Malware of Smishing and Vishing and Malware of Phishing and APT. Targeted activates include detection of Malware, invade method of malware, malicious activities performed by the malwares and their command-and-control server. List of Devices used are IPhone 6, Samsung galaxy S3, wireless router and, List of OS used are Linux, android, IOS. List of Malwares investigated are SPAp.APK GMS.APK, V3Plus.APK, 23983JJF.APK Tools: Taig, Clutch. iTools, JEB, IDA ProAndroid image extractor.

In [9], authors targeted to examine the Mirai botnet server through forensic examination while acquiring the remote access of the server. The investigators set up the Mirai botnet network architecture to retrieve the list of the infected IoT devices, the past statistics for the DDoS attacks, and retrieved as numerous login credentials as possible. It was required to gather the forensic artefacts left on the attacker's terminal, scan receiver, database server, loader command and control (CNC) server, as well as the network packets. Therefore, disk and memory image were acquired and also the author reverse engineered the live processes and service executable from the control servers of the Mirai botnet. This study outlined how a forensic investigator can access these artefacts remotely and to gather artefacts which target machine can provide beneficial information. The forensic examination of compromised IoT devices and DDoS attack victims is out of scope of this study. Instead, they concentrate on the attacker’s-controlled devices. 'Vulnerable IoT device' and 'Infected IoT device' models are created on Raspberry Pi 3 Model B computers. The operating system is a forensic workstation running 64-Bit Kali Linux. For acquisition and analysis of memory (RAM), the forensic tool Linux Memory Extractor (LiME) and Volatility 2.6 were used, respectively. DD 8.3 was used to obtain the disk image. Data recovery and file system analysis were enabled in Autopsy 4.11.0. PCAP analysis and monitoring of network traffic was carried out with Wireshark 3.0.3. To extract network packets from the RAM dump, Bulk Extractor was used. The executable files are reverse engineering by using the Ghidra 9.0.4 tool from the National Security Agency (NSA).

2.4. Smart Device Forensic Frameworks

Kebande et al. [11] have proposed a general-purpose, ISO/IEC 27043: 2015-based Digital Forensic Investigation Framework for the Internet of Things (DFIF-IoT) that may reasonably accommodate emerging IoT investigative capabilities. Three separate components are combined into the framework, and they comprise: (1) “Proactive process” which deals with activities aimed at rendering the environment of IoT forensically prepared to use. (2) “Reactive process” is represented by the digital forensic investigation process which could be initiated after a potential security incident becomes apparent. The (3) “IoT forensics” represents various forensic strategies where IoT evidence can be obtained.

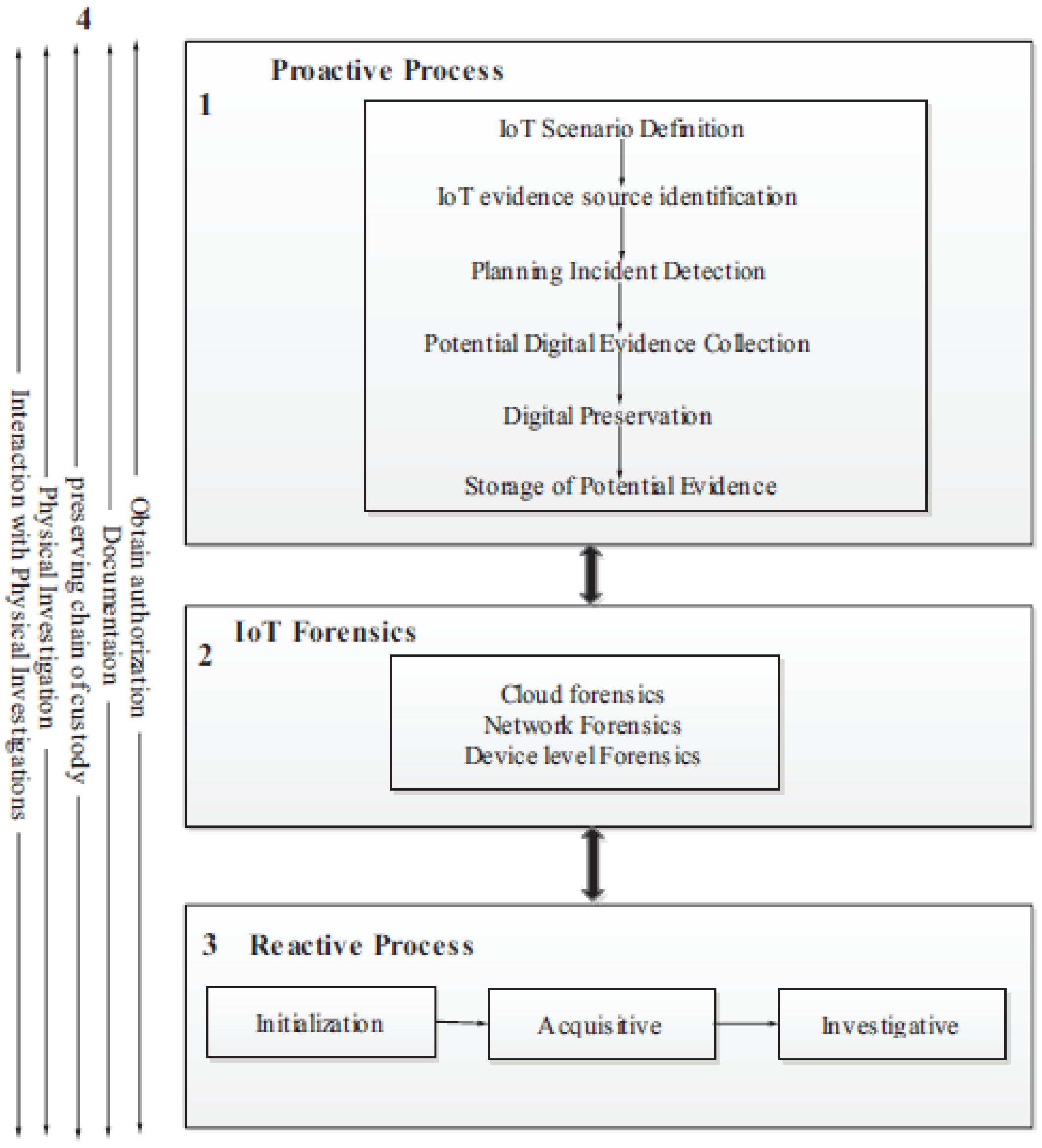


Figure 1: Digital Forensic Investigation Framework for IoT [11]

Kebande et al. [12] proposed an IoT-based ecosystem Integrated Digital Forensic framework which can analyze IOT Digital Evidence and in addition to it, this framework defines IOT management platform. It defines IOT policies and standards from organizational aspect. The generic Digital Forensic Investigation Framework for IoT environment (DFIF-IoT), which was first presented, is expanded upon in the IDFIF-IoT framework. In figure 3, IDFIF-IoT has been shown utilizing nine different subprocesses, including, Things (1), Device Connectivity and Communication Network (2), Readiness Process Groups (3), IoT Forensics

(4), Digital Investigation Process (5), Concurrent Processes (6), IoT Management Platform (7), IoT Policy (8) and IoT Standards (9). The DFIF-IoT and IDFIF-IoT differ primarily in that the earlier one was universal and governed by the ISO/IEC 27043 international standard, whereas the latter has integration of organizational aspects. IDFIF-IoT is also more policy-oriented because post-event response processes, readiness, and the "things" themselves are all direction oriented.

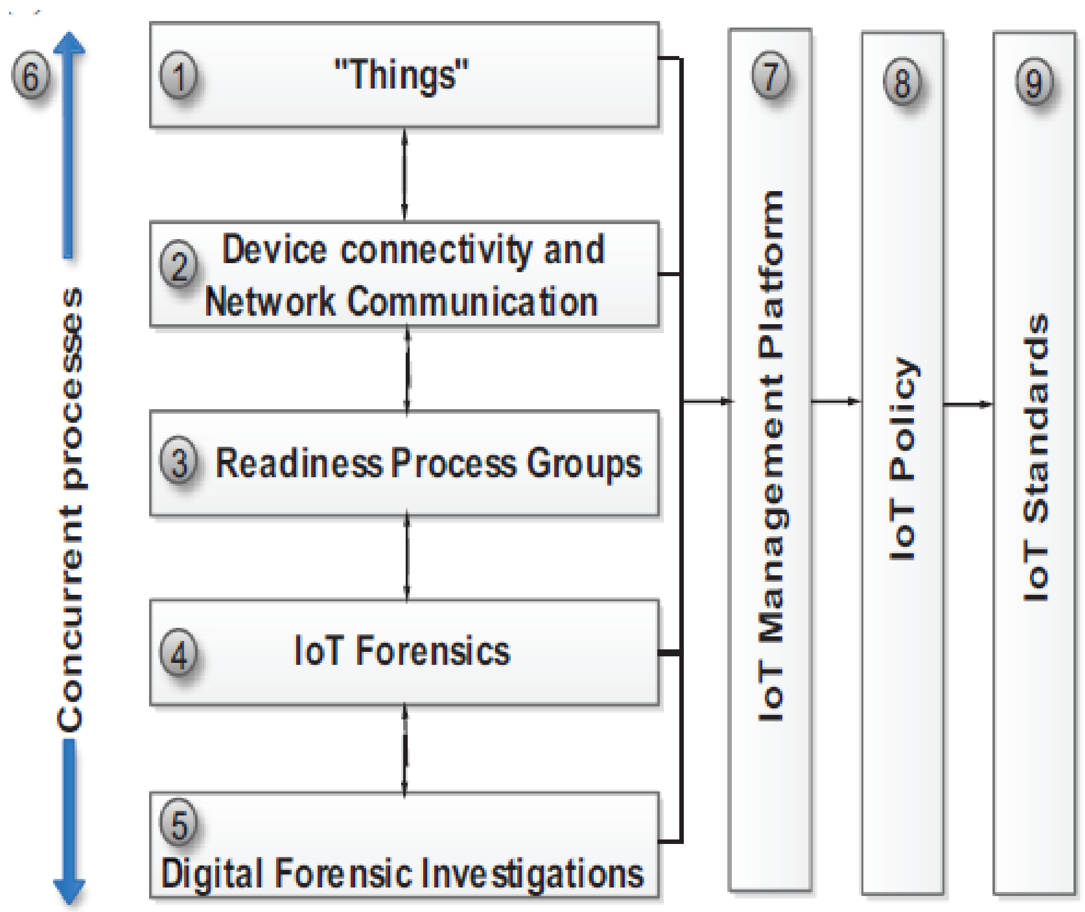

6
Concurrent processes
1 "Things"
2 Device connectivity and Network Communication
3 Readiness Process Groups
4 IoT Forensics
5 Digital Forensic Investigations
7 IoT Management Platform
8 IoT Policy
9 IoT Standards

Figure 2: Integrated Digital Forensic Investigation Framework for IoT [12]

In [13], to deal with the primary problems with digital IoT forensics, Al-Masri et al. presented the fog based IoT forensic framework (FoBI). Fog based IOT forensic framework makes use of the IOT utilizing the fog computing paradigm, which aids in pushing intelligence to the outer edges of a network through a gateway. Fog-based computing is appropriate for IoT systems with lots of installed IoT devices and high data volumes. Such a fog node may save the last known position of a connected device, the framework can extract log files linked to the broken device, and the fog node will alert other Internet of Thing's devices or networks of a potential danger. Then, using FoBI, it is feasible to recover forensic evidence.

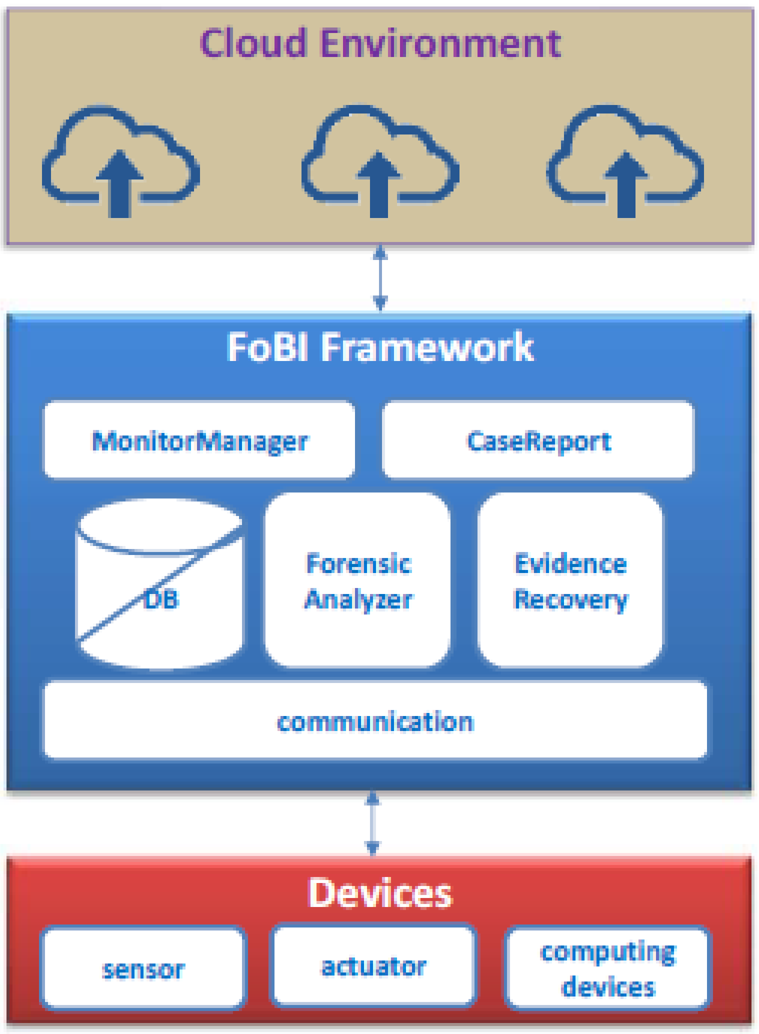


Figure 3: Fog-based Digital Forensic Investigation Framework for IoT [13]

2.5. Related Work

These days, system log analysis is used to undertake numerous investigations and research in a variety of sectors. Designers and investigators can determine the current state of the system and identify any odd conditions by analyzing the logs. For analyzing the Android log, utilize Logcat [14]. Even while the Logcat has the benefit of being simple to use, it has the drawback of just being able to inspect the basic logs that the Logcat provides. In this article, we demonstrate the examination of Application Installation Log files on Android Systems and create an installation log management solution. Using Python and Android Debug Bridge (ADB), the setup of the log management program gathers log files that are kept locally and can only be accessed by root users. Important forensic artifacts found are androidMenifest.xml file to get installation information about the APK, Localappstate.db: records information of all installed apps, Library.db contains ownership of installed apps and Frosting.db does not detect the installation time but contains traces of APK. When a normal app and malicious app is installed the change in APK path can be clearly seen in frosting.db record.

The researchers [15] presented a thorough forensic examination of Cisco WebEx, one of the top three videoconferencing programs on the market right now. More specifically, we provide the findings of the forensic examination of the web, Android, and Cisco WebEx desktop client apps. Memory, disc space, and network forensics are the three elements of digital forensics that we concentrate on. It is clear from the collected artefacts that useful user data may be obtained from many data locations. The Advanced Encryption Standard (AES) keys, contact information, emails, user IDs, profile images, chat messages, shared media, meeting information, including meeting passwords, keywords searches, timestamps, and phone logs are among them. We use the retrieved artefacts as the foundation for creating a memory interpreting tool for Cisco WebEx. The anti-forensic artefacts we find also include deleted conversation messages. Despite the fact that network connections are encrypted, researchers are able to gain access to important artefacts such the IP addresses of host devices and server domains as well as message/event timestamps. Andriller CE tool is used to conduct analysis and important artifacts found are SQLite database. *\data\apps\com.android.vending\db* folder contains frosting.db which has the APK path, other SQLite databases including install_source.db, install_queue.db, suggestions.db, localappstate.db, verify_apps.db, and xternal_referrer_status.db, provided some digital evidence of application's usage. *\data\apps\com.android.vending\db* folder also contains library.db which lists the email address against WebEx Meetings and the certifcate hash of the application. *\data\apps\com.google.android.googlequicksearchbox\r\app_webview* folder contains Cookies.db that contains the WebEx meetings web cookies. *\data\apps\com.android.vending\db* folder also contains SQLite databases like auto_update.db, and data_usage.db which indicates traces of usage/installation.

To address the security threats brought on by the widespread distribution of smishing malware, multiple studies have been carried out. Some of them employed the Naive Bayesian classifier [16, 17], which analyzes the properties of smishing characters and finds smishing characters employing rule-based techniques [18]. Additionally, research was done to identify malware using app network traffic analysis [19] and integrating API call and authorization information [20]. Taint analysis has been used in studies to proactively identify malware using Android malware [22, 23, 24]. Similar research has been done to identify malware by examining an app's behavior using data flow analysis [25, 26, 27]. Another study [28] compared the investigation's findings of the previous data flow analysis. PACE has been offered as a comprehensive solution for malware analysis that offers machine learning-based Android malware detection technologies via REST API, web interface, and ADB interface [29]. Additionally, experiments on dynamic analysis utilizing machine learning for malware detection on actual devices were undertaken [30], [31], to address the drawbacks of malware detection in Android emulators. Studies have suggested a way of identifying malware that combines a signature-based, motion-based detection with data mining approaches [32]. Another research detected the malware by analyzing android system permissions required by malicious applications [33]. Significant Permission Identification (SigPID) was created in this investigation. Because of the tests, SigPID efficiently identifies new malware by mining permissions data to identify malware.

Additionally, a study that combined features of static analysis and dynamic analysis of Android apps with deep learning technology created an engine called DroidDetector that enhances the ability to identify about malware through a successful extraction of specific features of malware [34]. Additionally, research named ToR-SIM Platform [35] proposed a mobile forensic platform for Android malware analysis and detection.

Similarly, Nisha et al. [36] suggested utilizing mutual information and chi-square approaches for the identification of characteristics to identify repackaged Android malware. Random forest classifier, among other used classifiers, was able to attain the best accuracy of 91.76% for assessment. The 88 uniquely recognized permissions for the study are the primary goal of their approach, which may be further condensed to include harmful ones.

Sandeep HR [37] did exploratory data analysis (EDA) and extracted data from the apps. The suggested method concentrated on employing deep learning techniques to detect malware during installation. Their detection architecture made use of a variety of tools, including permissions, to mimic the actions taken by the programs. They are successful in classifying with 94.6% accuracy using Random Forest. Their method excludes mimicry attacks, cloning of apps, and adware and instead employs 331 characteristics for categorization that may be further optimized.

In Multilevel Permission Extraction (MPE) technique, Zhen Wang et al. [38] concentrated on finding the permissions automatically that aids in differentiating among the good and bad apps. Their dataset consisted of 9736 apps from each of the sets of categories, malicious and benign, and experimental findings indicate that the detection rate of 97.88% is reached. In a different study, Ming Fan et al. [39] developed a method for creating frequent subgraphs, or "fregraphs," to describe the typical behaviors of viruses from the same family. They suggested FalDroid, a technique for fregraphs based detection. As per preliminary findings from their testing, FalDroid can categorize samples of malware up-to 94.2% of into the appropriate categories in approximately per program with 4.6 seconds. Without an objection, both of these methods accomplish the ultimate objective well, but at a considerable cost in terms of more computations as well as the time required.

Wang et. al. [40] designed a permission-based detection approach which uses contrasting permission patterns to differentiate malicious and benign applications. They extract information regarding required and used permissions for mining permission patterns which they later use to detect Android Malwares. The dataset used by Wang et. al. consisted of 2454 Android applications (1227 applications for each malicious and benign class) comprising of different application categories such as games, entertainment etc. With their analysis, they were able to achieve a high accuracy of 94%, with 5% false positives and 1% false negatives.

By evaluating the behavior of the virus and utilizing the cuckoo sandbox to investigate its behavior, this sandboxing technique can identify malware samples whose source code is not trusted. A malicious code examination tool called Cuckoo looks at malware in greater depth and offers thorough results depending on the tests it runs. Its objective is to offer a method for automatically analyzing files and to present all of the links between the system and the files being analyzed. Windows executables, PDF files, DLL files, Internet URLs, Office documents, and Java files are the primary targets. [43]

For the research's categorization of harmful programs, most of the approaches used a collection of permission-based characteristics. Since they offer quick and almost precise detection, permissions-based approaches are typically used for the detection of maliciousness.

Since the examination is done before the app is installed, there is no risk of the device being harmed. For this reason, permissions may play a crucial part in the quicker identification of malware. Additionally, minimizing useless permission characteristics might simplify computations.

To apply a computation-effective and rapid detection strategy, a solution that makes use of android system permissions is provided.

2.6. Research Gap

The literature review demonstrates that considerable advancements have been achieved in the creation of frameworks and approaches [18-20] for malware detection on various smart devices. However, there is still room for improvements that must be addressed. Most of the study targeted the analysis and detection of malware on certain smart devices, such as Mirai botnet servers and Android smartphones. There is no generic framework or approach for the diverse nature of smart platforms and devices. Numerous studies use smart environments that are emulated or simulated, which could not accurately reflect the richness and diversity of smart device ecosystems in the real world. Validating forensic methodologies requires running experiments on real smart devices. The majority of existing methods concentrate on static malware analysis. The identification and analysis of malware that might not leave traces in memory might be improved by using dynamic analysis approaches. The acquisition and analysis of forensic artefacts from smart devices requires the development of more effective forensic investigation techniques. A study is required to recreate the sequence of events and pinpoint the underlying causes of breaches of privacy in smart networks. In order to maintain the privacy and security of smart devices and ecosystems, the proposed study would address a few of these gaps such as: physical mobile device investigation, identifying multiple real time malwares from a bunch of other benign applications, propose a comprehensive incident response model using forensic investigation methodology. The proposed forensic investigation methodology comprises of the most suitable artifacts and paths to provide efficient output.

Summary

This chapter covers the background and the related work of the smart malware detection approaches. The related literature has been presented along with a critical analysis of the studies. Existing research work and schemes used in literature help in formulating the solution to the identified problem.

Chapter 3

3. Research Methodology

This research methodology will be explained that is followed to carry out this thesis research. A brief description of the methods that are used in our research methodology along with the phases followed in the research process, *i.e.*, acquiring digital image of smart device, investigating the artifacts and paths, and malware detection using digital forensic investigation are given in this chapter.

3.1. Introduction

Here is a brief overview of research methodology that improved Dohyun Kim et. al. [9] malware detection methodology by identifying the artifacts and new location that are useful and beneficial for malware detection investigation.

To meet the research objectives the proposed methodology studied the generic Digital Forensic Investigation Framework for IoT (DFIF-IoT) based on the ISO/IEC 27043: 2015 and added the research contribution to design incident response methodology by only extending the smart forensics phase of above framework. The overview of proposed model to conduct smart device forensic is below:

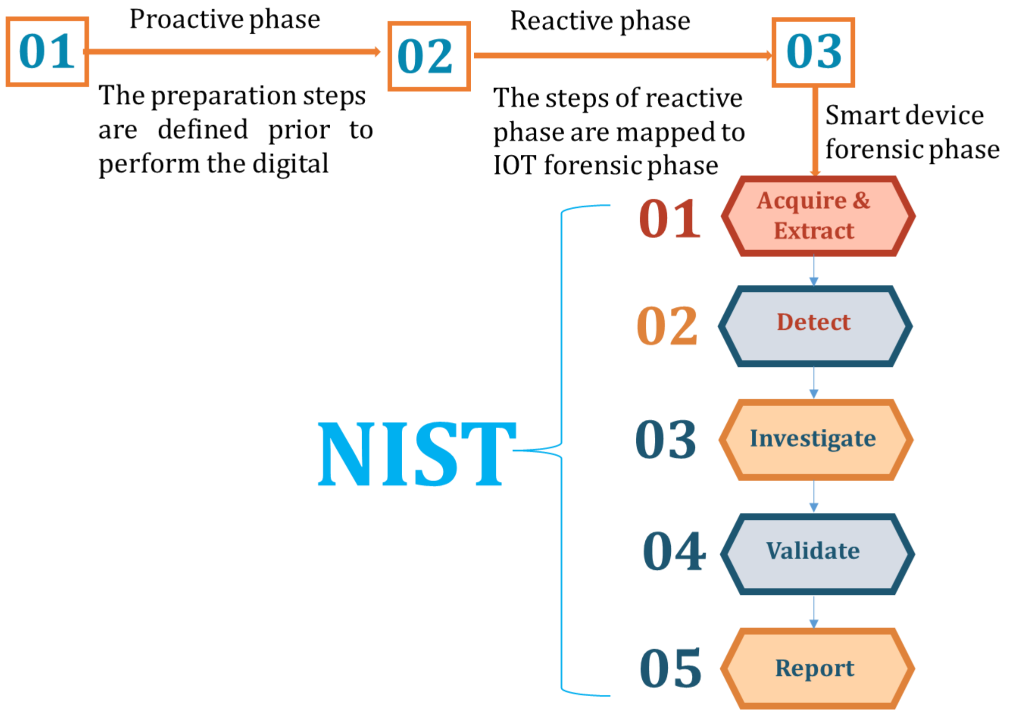

01
Proactive phase
The preparation steps are defined prior to perform the digital
02
Reactive phase
The steps of reactive phase are mapped to IOT forensic phase
03
Smart device forensic phase
NIST
01
Acquire & Extract
02
Detect
03
Investigate
04
Validate
05
Report

Figure 4: Forensic investigation Steps Overview

The steps how forensic investigation will be carried out is below:

A. Using the ADB bridge and DD to obtain the physical image of a smart device.

B. Forensic image examination of the devices and autopsy-based artifact discovery.

C. Based on findings, construct the methodology that contains the most suitable artifacts and path for malware detection.

D. Analyzing the applications of device by following constructed methodology.

E. Comparing the benign and suspicious applications to see changes in the paths and artifacts.

F. After comparison identify vital paths and artifacts that can be useful for forensic investigation

G. Analyzing all device applications by the newly identified paths

H. Final suspicious applications are validated by the cuckoo sandbox to verify if the identified applications are real malware.

After performing the stages, a report of the malware detection summary is generated which can then, later, be used to carry out further evaluations.

3.2. Thesis Research Methodology

The system architecture of the methodology is shown in Figure 4. It depicts the proactive, reactive and smart device forensics process. In this research we are extending only the smart device forensic process part. The NIST framework based forensic process is being followed and steps are acquired, extract, investigate and detect. These processes are mapped to smart device forensics and steps are defined for each phase. It depicts an abstract level view of proposed methodology to detect malware at device level evaluation process. How APKs will be evaluated by forensic investigation to distinguish benign and malicious. The key components are discussed in the below topics.

For efficient incident response, the research proposed incident response framework that can filter the benign and malicious applications and it can detect multiple malwares at the same time. The proposed incident response framework methodology is shown in Fig. 4, and that comprises of three phases:

(1) Proactive

In the proactive phase the steps are defined prior to performing the smart device forensic. While, in reactive phase when real time forensic examination is started.

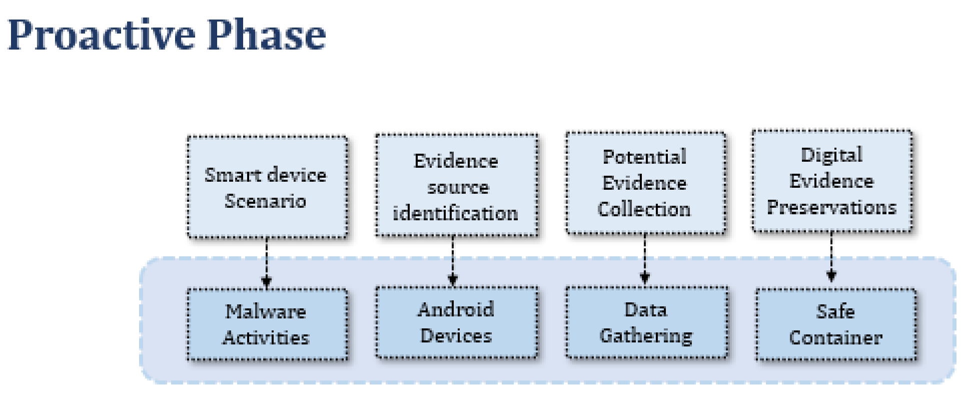


Figure 5: Proactive process phase-Digital forensic preparation

(2) Reactive

The steps of reactive phase are mapped to smart device forensic phase.

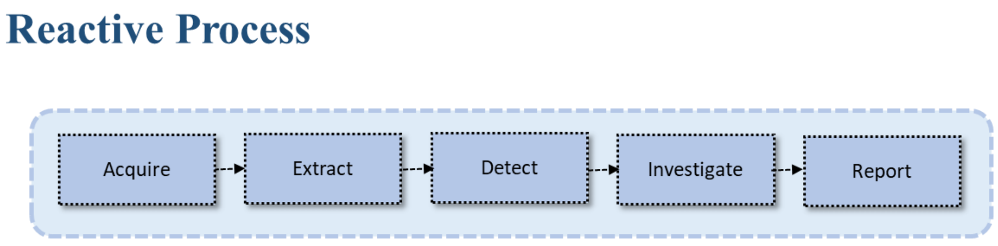


Figure 6: Reactive process phase-Defined steps to perform digital Forensic.

(3) Smart device forensics

The smart device forensic phase is following the reactive process to detect the malware by performing several steps like path traversal, examining invade method of malware, detection, and reporting.

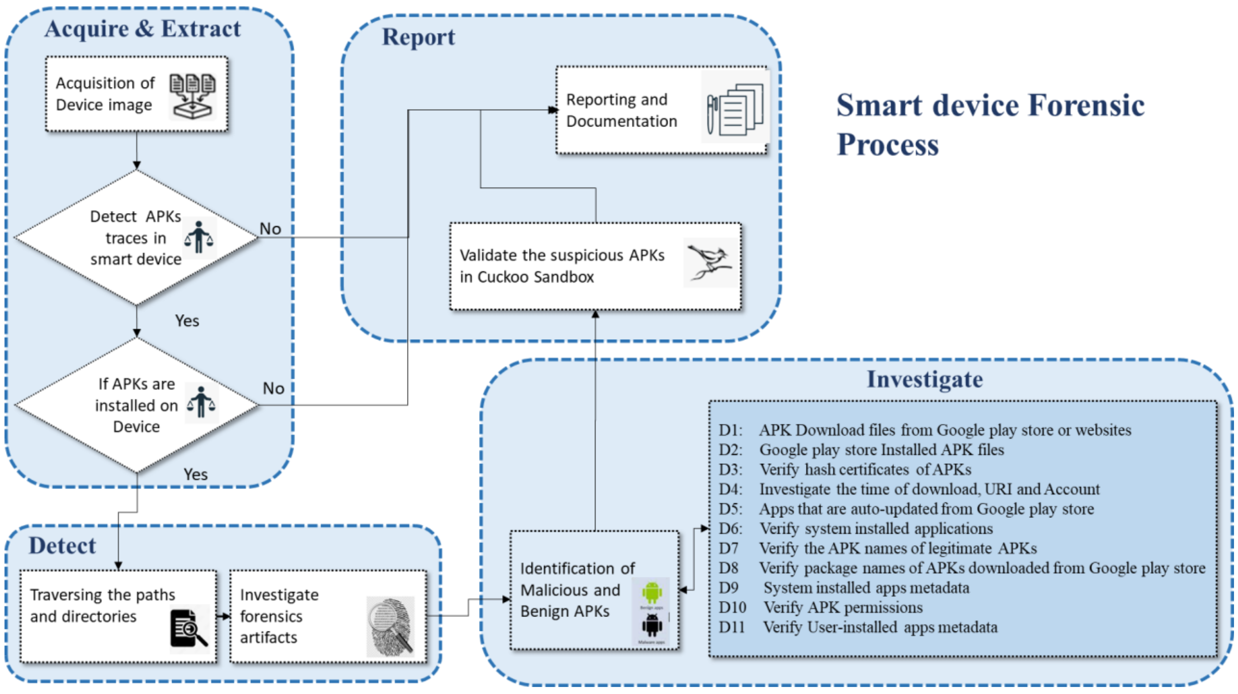

Acquire & Extract
Acquisition of Device image
Detect APKs traces in smart device
No
Yes
If APKs are installed on Device
No
Yes
Report
Reporting and Documentation
Validate the suspicious APKs in Cuckoo Sandbox
Smart device Forensic Process
Investigate
D1: APK Download files from Google play store or websites
D2: Google play store Installed APK files
D3: Verify hash certificates of APKs
D4: Investigate the time of download, URI and Account
D5: Apps that are auto-updated from Google play store
D6: Verify system installed applications
D7 Verify the APK names of legitimate APKs
D8 Verify package names of APKs downloaded from Google play store
D9 System installed apps metadata
D10 Verify APK permissions
D11 Verify User-installed apps metadata
Identification of Malicious and Benign APKs
Detect
Traversing the paths and directories
Investigate forensics artifacts

Figure 7: Proposed Model for Incident response using Digital Forensic Investigation for smart device.

When analyzing Android device digital forensic image suspected of being infected with malware, there are several important paths and artifacts that investigators should look for. Some of the key areas to focus on include:

1. Application data and logs: Malware typically leaves traces of its activities in application data and logs. Investigate the data and logs of installed applications, attentions required to suspicious or abnormal behavior, such as excessive network connections or unusual file activity.

2. System logs: System logs contain information about the device's operation and can be a valuable source of information when investigating malware. Check the system logs for any unusual activity or errors, such as repeated crashes, unusual network activity, or unexpected changes to system files.

3. File system artifacts: Malware often creates files or modifies existing files on the device. Investigate the file system for any files that look suspicious, such as executables, hidden files, or files with unusual names.

4. Malware binaries: If it is suspected that the device is infected with malware, then extract the binary files of the malware and analyze them further. These files can often be found in the application data or system directories of the device.

For evaluation, the following directories should be investigated to find useful forensic artifacts:

1. /system/app: Contains system apps that are pre-installed on the device. Malware may be disguised as a legitimate system app, therefore it's important to check for any suspicious apps that may have been added.
2. /data/app: Contains user-installed apps. Malware may be installed as a legitimate-looking app, so it's important to check for any unfamiliar or suspicious apps.
3. /sdcard: Contains user data, including photos, videos, and files. Malware may be disguised as a file, such as a PDF or document, so it's important to scan for any suspicious files.
4. */data/data*: User-installed application data, including databases and cache files. Malware may store data here, so it's important to check for any suspicious data associated with unfamiliar or suspicious apps.

No	**Paths**	**Description**	**Artifacts**
D1	*Android/providers/media/external.db*	SD card Filesystem Information	Data, size, format,parent, data_addded, data_modified, Mime_type, Title, bucket_id, bucket_display_name, media_type, storage_id,
D2	*Android/vending/databases/installqeue.db*	Google play store App Trace	Reason,package
D3	*Android/vending/databases/library.db*	Certificate hash of the application	Account, doc_id, document_hash, app_certificate_hash
D4	*Android/vending/databases/localappstate.db*	Information of all installed apps	Package name, download uri, account, title, download and update timestamp, app name
D5	*Android/vending/databases/frosting.db*	All APK paths and pkg names	Apk_path, package name
D6	*Android/providers/media/internal.db*	Contains data of internal system	Data added or modified time, app name, title, size
D7	*Android/vending/databases/ verifyapp.db*	Contains only APK name	APK name
D8	*Android/vending/databases/autoupdate.db*	Information about the auto update of apps	Package name

D9	*/system/app/**	Pre-installed system apps on the device	System installed apps metadata
D10	*data/system/package_cache/1*	System permissions for all installed apps	Intent, activity and APK permissions
D11	*Data/app/*	Contains data of user-installed apps	User-installed apps metadata

Table 1: Target artifacts and paths for analysis

Table 1. contains the targeted paths and directories which are traversed from D1 to D11 to analyze the necessary information about the device applications.

- **D1(external.db):** This file resides in the internal memory of a device. It contains the file system metadata for all existing files in the /sdcard area. After invading into the device through smishing or phishing, the malicious applications download its configuration files to the /sdcard area.. Therefore, to check the app installation items in the /sdcard directory, this file is necessary. This pertains to the file that is linked to D1 in Table 1.
- **D2 (installqeue.db):** The file corresponding to D2 in Table 1, contains forensic information related to app installation and update activities on Android devices. This information may include package names, installation timestamps, update version numbers, and installation source details. It can be useful for digital forensics investigations to track app installation history and identify potential malicious or unauthorized installations.
- **D3 (library.db):** File corresponding to D3 in Table 1. contains ownership of installed apps, lists the email address against installed apps, and the certificate hash of the

application. This file is required to verify the authenticity and hash of installed applications.

- **D4 (localappstate.db):** A file storing the app installation metadata on smart devices. The file that relates to D4 in Table 1 for Android includes details on the application. These details include its name, latest update date, installation date, package name, and Google play account used to download the app.
- **D5 (frosting.db):** The file corresponding to D5 in Table 1. does not detect the installation time but contains traces of APK. When a normal app and malicious app is installed the change in APK path is detected.
- **D6 (internal.db):** The file corresponding to D6 in Table 1. contains the application data downloaded from official vendors or Google play store. Its analysis is necessary for classification and comparison of behavior among user-installed apps from third party or Google play store.
- **D7 (verifyapp.db):** The file corresponding to D7 in Table 1. contains forensic information related to app verification and licensing on Android devices. This information may include app package names, version numbers, licensing status, timestamps of app installations, and verification tokens. It can be valuable for digital forensics investigations and analyzing app usage patterns on a device.
- **D8 (autoupdate.db):** The file corresponding to D8 in Table 1. contains traces of apps packages which are downloaded from official vendors like Google play store and contain information about the automatic update of apps. It is necessary to analyze this file to figure out which app gets update from Google play store even after installation as some apps downloaded from Google play store also contains malicious codes.

- **D9 (*/system/app*):** The files corresponding to D9 in Table 1. contains system apps that are pre-installed on the device. Malware may be disguised as a legitimate system app, so it's important to check for any suspicious apps that may have been added.
- **D10 (*/package_cache/1*):** The file corresponding to D10 in Table 1. contains the installation package of all apps, and permissions in system for all installed apps.
- **D11 (*/Data/app*):** All user-installed app installation files are located inside the IOT device. From among the files suspected of being harmful, the investigator chooses and carefully examines the files matched to D11 of Table 1.

Only the smart device forensic process part of Fig. 1 has been extended to achieve research outcomes. The NIST framework based forensic process is being followed and steps are acquired, extract, detect, investigate, and validate. Fig. 2 depicts the technical perspective of proposed methodology for device level forensic to detect multiple malwares at the same time, it shows how APKs will be evaluated by forensic investigation to distinguish between benign and malicious applications. Smart device forensic phase comprises of following steps as depicted in Figure 4:

3.2.1. Acquire

An Android-based device's root-privilege shell may be opened with the use of the Android Debug Bridge (ADB) and a rootkit. From there, a trusted 'dd' software can be run to capture an image of the device's memory, both removable and internal [40]. Throughout this work, the "rootkit method"—a technique employed by several professional mobile phone forensics programs—will be referred to. Installing a rootkit requires some sort of modification to the device, even if it's just a very little one. The most valuable digital evidence is likely to exist on the user data partition, hence Vidas et al. advise against modifying it and instead recommend re-flashing the recovery partition of the smartphone and replacing it with a forensic acquiring setting [41]. Restarting the device into "recovery mode" results in the collection of a picture utilizing the reliable forensic acquisition platform. In this study, this tactic is known as the "recovery mode" technique. [42]

3.2.2. Extract

It refers to the artifact finding stage. The forensic image acquired in first phase is loaded into sleuth kit autopsy. Forensic image is then analyzed to find forensic artifacts and directories by traversing through the paths comprising of application data and logs, system data and logs, filesystem artifacts and malware binaries. The path traversal helped into the detailed insights of the system data and artifacts. In this phase, changes have been detected into the system paths made by the applications and forensically useful paths were identified. Based on findings, the forensic investigation methodology has been constructed that contains targeted artifacts and paths for malware detection as mentioned in Table 1.

3.2.3. Detection of installed APKs

It is important to detect the presence of applications as the first step of investigation. The applications usually present are user installed apps and system installed apps. The user-installed apps are downloaded from Google play store or from untrusted source or third-party. The presence of user installed APKs are detected at the D1 and D6 paths. The basic information about APKs e.g., the download method, URI, installation timing etc. are gathered. This information is useful only when these APKs are installed in the device. To detect the installed applications D9 and D11 paths are analyzed. Forensic artifacts for the installed applications e.g., APK path, package name, installation files etc. are found which is useful to move forward with forensic investigation for malware detection.

3.2.4. Investigate

The installed applications of device can be analyzed and investigated by following constructed methodology. Installed applications are investigated by traversing through all the paths mentioned in Table 1. During the path traversal important artifacts are collected and applications are evaluated based on these artifacts. After evaluation, the applications are classified as benign and malicious. These benign and suspicious applications are compared to detect changes in the system files e.g., the installation files and paths, application packages, application behaviors, intent, activities, system permissions required by benign and malicious APKs etc.

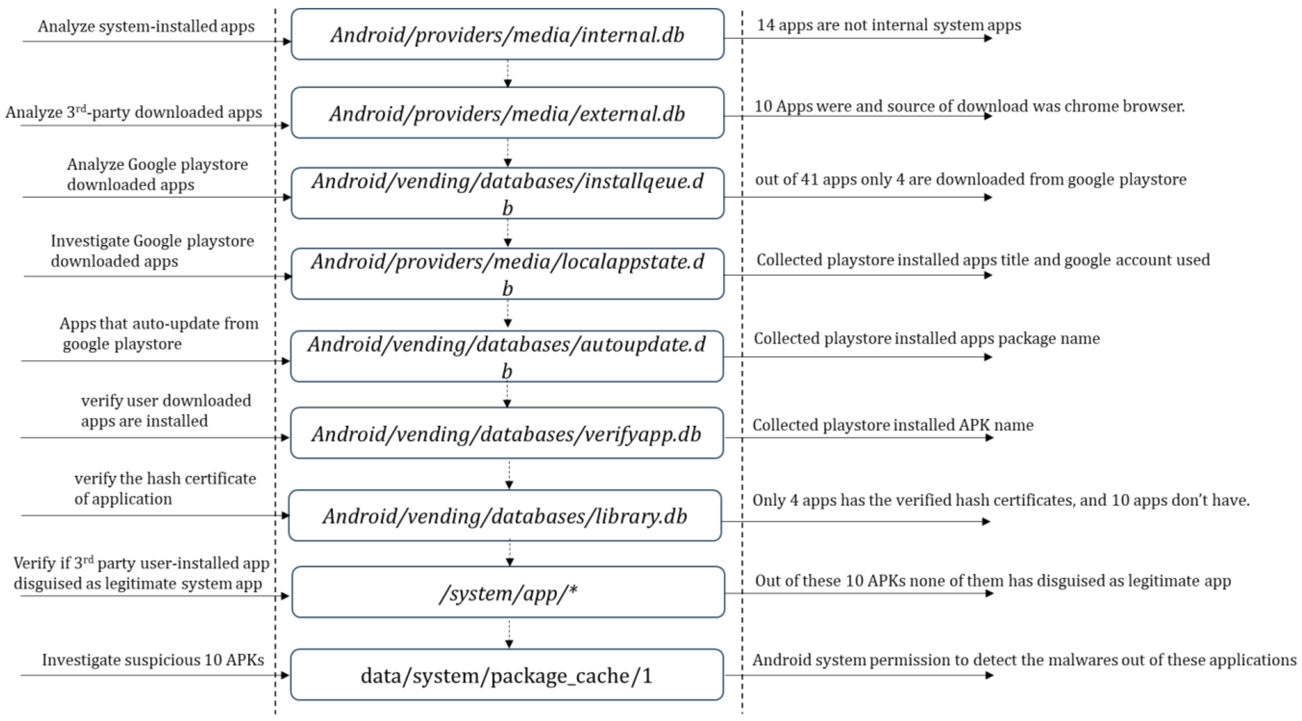

Analyze system-installed apps
Android/providers/media/internal.db
14 apps are not internal system apps
Analyze 3rd-party downloaded apps
Android/providers/media/external.db
10 Apps were and source of download was chrome browser.
Analyze Google playstore downloaded apps
Android/vending/databases/installqeue.db
out of 41 apps only 4 are downloaded from google playstore
Investigate Google playstore downloaded apps
Android/providers/media/localappstate.db
Collected playstore installed apps title and google account used
Apps that auto-update from google playstore
Android/vending/databases/autoupdate.db
Collected playstore installed apps package name
verify user downloaded apps are installed
Android/vending/databases/verifyapp.db
Collected playstore installed APK name
verify the hash certificate of application
Android/vending/databases/library.db
Only 4 apps has the verified hash certificates, and 10 apps don't have.
Verify if 3rd party user-installed app disguised as legitimate system app
/system/app/*
Out of these 10 APKs none of them has disguised as legitimate app
Investigate suspicious 10 APKs
data/system/package_cache/1
Android system permission to detect the malwares out of these applications

Figure 8: Smart device Investigation Process

3.2.5. Validate

The suspicious applications which are filtered out as a result of investigation are fed into the cuckoo sandbox to verify whether the detected suspicious APKs are real malware or benign. This step proves the effectiveness of the methodology that is proposed and verifies the results of investigation by identifying all suspicious applications as dangerous APKs. At the final step, the results are reported.

3.3. Summary

In this chapter, we have discussed different methodologies that have been used in the research and can be followed to achieve similar results. The overall view involves are acquired, extract, investigate and detect to detect malware by forensic investigation. In the next chapter, we will look at the experimental setup designed to perform this specific analysis.

Chapter 4

4. Experimental Setup

This chapter explains the experimental setup that has been designed to create and set up an environment to conduct the research. This chapter also justifies why some of the processes have been followed. System configurations are also provided in this chapter.

4.1. Overview

The experimental setup includes an Android Redmi Go device consisting of malicious and benign applications and a PC for consisting of tools like ADB Bridge, Autopsy and Cuckoo sandbox to acquire the physical image of device, to analyze the forensic image for filtering benign and malicious applications and validate the malicious applications as malwares respectively.

4.2. Setting up Environment

For carrying out experimentation, a windows-based machine has been used. The specifications of the system have been shown in below table 1.

Property	**Description**
Manufacturer	HP
Model	Pavillion
Architecture	x64 based
Operating System	Windows 11
Processor	Intel(R) Core (TM) i7-8550U CPU @ 1.80GHz 1.99 GH GHz
RAM	8 GB
Storage	1 TB

Table 2: System Specification

Property	**Description**
Manufacturer	Redmi
Model	Redmi Go
OS	Android 8.1 Oreo
Processor	Quad-core 1.4 GHz Cortex-A53
RAM	1 GB
Storage	16 GB

Table 3: Smart device specification

4.3. Malicious and benign APK Sample collection

To construct the dataset, we have collected different samples of Android applications containing applications from two distinct sets of android families i.e., malicious and benign. Both types of samples were collected from different sources. Benign samples were collected from the official Play Store. The collected benign samples represent applications from different application categories such as business, entertainment, Finance, and games, etc. to provide as much diversity as possible to the samples.

While for the malicious samples, we have collected from GitHub Android malware database. The samples on GitHub are available in the form of zip files and can be downloaded [44].

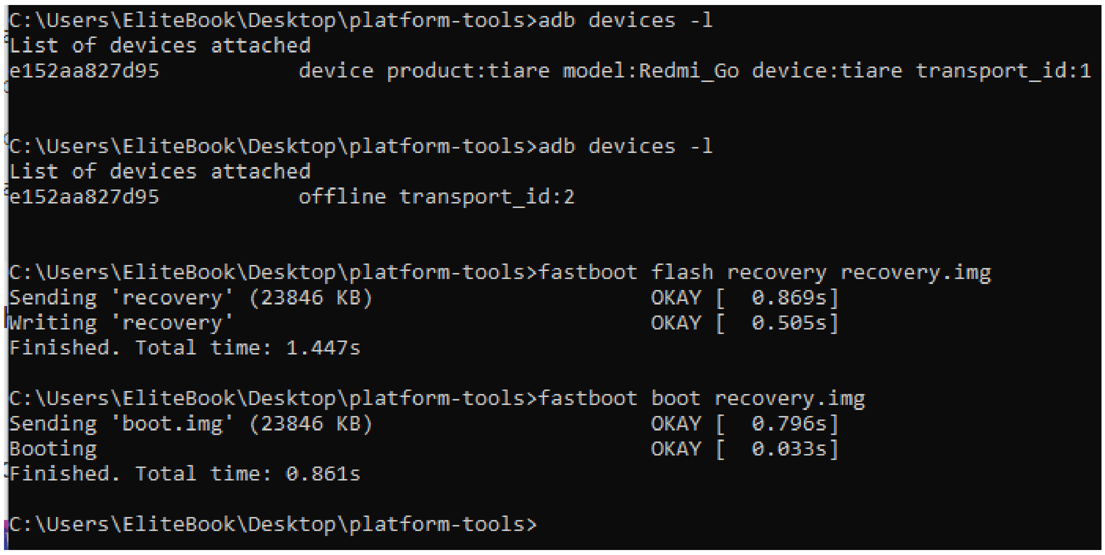
Malware	**Quantity**
Rootnik: E5E22B357893BC15A50DC35B702DD5FCDFEAFC6FFEC7DAA0D313C724D72EC854.APK	1
Krept banking: krep.itmtd.ywtjexf-1.APK	1
Candycorn: 14d9f1a92dd984d6040cc41ed06e273e.APK	1
Nimaz ka waqt: 1514376339e4a0b4727c6897640c7c3e.APK	1
Xbot: 1264C25D67D41F52102573D3C528BCDDDA42129DF5052881F7E98B4A9	1
Zip extractor: com.zip.unzip.zipextractor.raropener.zipfile	1
Rubbish cleaner: com.snt.rubbishcleaner	1
Photo processing: 263b0851156f7d77fb43368ce13bede1	1
Lockkeeper: 0e8805b683bc0fd8a6d49b07205f1a4b	1
Oscorp: 20230307/f73ebc6f645926bf8566220b14173df8.APK	1

Table 4: Malware Samples

4.4. Setting up the Android device

- Redmi Phone with Andriod version 8.1 is used for this process.
- Rooting of device was performed using TWPR & Magisk
- Download TWPR for Redmi Phone & Magisk installer

- Copy the Magisk through MTP/file transfer on Phone internal storage.
- Reboot into the bootloader
- Flash TWPR through Fastboot mode
- Boot the phone in RWPR recovery mode (fastboot reboot)
- Install the Magisc from the phone internal storage
- After Magisc installation rebooted the phone
- After reboot Magisc Manager App was there, simply run it to verify Magisc has been installed.
- Magisc Manager will control the root access, Magisc will monitor the root access for every app and will allow or deny the access.
- Installed "Root Checker" to verify rooting status



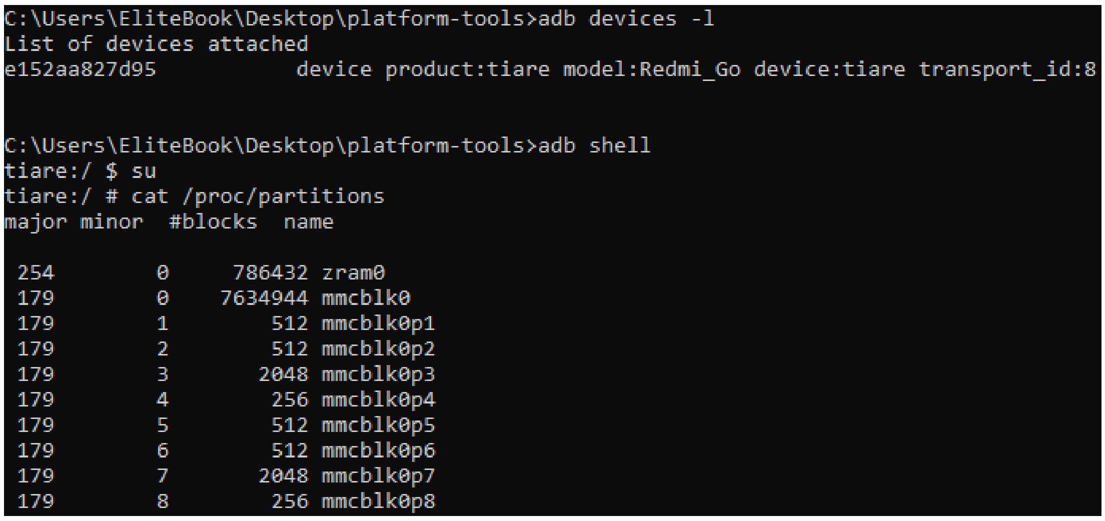
```
C:\Users\EliteBook\Desktop\platform-tools>adb devices -l
List of devices attached
e152aa827d95           device product:tiare model:Redmi_Go device:tiare transport_id:1


C:\Users\EliteBook\Desktop\platform-tools>adb devices -l
List of devices attached
e152aa827d95           offline transport_id:2


C:\Users\EliteBook\Desktop\platform-tools>fastboot flash recovery recovery.img
Sending 'recovery' (23846 KB)                      OKAY [  0.869s]
Writing 'recovery'                                 OKAY [  0.505s]
Finished. Total time: 1.447s

C:\Users\EliteBook\Desktop\platform-tools>fastboot boot recovery.img
Sending 'boot.img' (23846 KB)                      OKAY [  0.796s]
Booting                                            OKAY [  0.033s]
Finished. Total time: 0.861s

C:\Users\EliteBook\Desktop\platform-tools>
```

Figure 9: Fastboot flash recovery process on ADB

Installed BusyBox on Phone, it requires root privileges for installation. BusyBox was installed to have "dd" utility. We used "dd method" for physical image acquisition.

Installed "Netcat" on PC, this utility is used for network connection through TCP.

Establishing ADB connection to phone from PC and switched to phone root access and viewed all the disks & partitions details in the in "/proc/partitions"



```
C:\Users\EliteBook\Desktop\platform-tools>adb devices -l
List of devices attached
e152aa827d95           device product:tiare model:Redmi_Go device:tiare transport_id:8


C:\Users\EliteBook\Desktop\platform-tools>adb shell
tiare:/ $ su
tiare:/ # cat /proc/partitions
major minor  #blocks  name

 254        0     786432 zram0
 179        0    7634944 mmcblk0
 179        1        512 mmcblk0p1
 179        2        512 mmcblk0p2
 179        3       2048 mmcblk0p3
 179        4        256 mmcblk0p4
 179        5        512 mmcblk0p5
 179        6        512 mmcblk0p6
 179        7       2048 mmcblk0p7
 179        8        256 mmcblk0p8
```

Figure 10: ADB to connect with mobile device and getting root access.

4.5. Pre-requisite Software Installation

Following Software need to be installed before the experimentation process in windows can be followed:

1. An Archiving tool such as WinRAR; *for extracting the application samples [45]*
2. Installing the ADB bridge; *for connecting with android and for logical and physical device image acquisition [46]*
3. Netcat; *to start a connection in PC that can connect with the android device with TCP based connection at port 'P' [47]*
4. Autopsy: for device logical & physical analysis of the acquired image to find forensic artifacts and paths [48]

Following Software need to be installed before the experimentation process in Ubuntu 18.04 can be followed.

5. Cuckoo sandbox: It is the leading open-source sandbox to automat malware analysis system for Windows, Linux, Mac or android [49]

4.6. Summary

In this chapter, we have covered the experimental setup that has been proposed to carry out the analysis. The process of collecting the required applications sample and setting the necessary environment and the related tools has been discussed in the chapter. Moreover, details about the installation of pre-requisite software for the analysis process and their sources have also been provided in this section.

Chapter 5

5. Experimental Results

This chapter explains the achieved results and their analysis in the form of detection results. The results are compared with the benchmark approach [9] and their achievements have been discussed in this chapter.

5.1. Overview

Android applications use permissions to provide the functionality to the users, which are exploited by malware developers for conducting cybercrimes. In this study, extensive analysis has been carried out on an Android application representing benign and malicious applications. We further investigated different forensic artifacts while performing the analysis to measure the effectiveness of the approach.

5.2. Important Forensic Artifacts

As described in methodology the useful forensic artifacts which are suggested:

1. *Android/providers/media/external.db*	
Description	Traces to SD card used in the device. This is stored on the phone. But the device doesn’t have SD card, the apps downloaded without google play store save their installation files here.
Artifacts	*Data, size, format, parent, data_addded, data_modified, Mime_type, Title, bucket_id, bucket_display_name, media_type, storage_id*
2. *Android/vending/databases/installqeue.db*	
Description	Traces of apps package which are downloaded from official vendors like google playstore and contain information about the auto update of apps.
Artifacts	Reason,package
3. *Android/vending/databases/library.db*	
Description	Library.db contains ownership of installed apps and lists the email address against installed apps, and the certifcate hash of the application
Artifacts	Account, doc_id, document_hash, app_certificate_hash
4. *Android/vending/databases/localappstate.db*	
Description	Records information of all installed apps

Artifacts	Package name, download uri, account, title, download and update timestamp, app name
5. *Android/vending/databases/frosting.db*	
Description	Does not detect the installation time but contains Traces of APK. When a normal app and malicious app is installed the change in APK path is detected in frosting.db
Artifacts	APK_path, package name
6. *Android/providers/media/internal.db*	
Description	Contains data of internal system
Artifacts	Data added or modified time, app name, title, size
7. *Android/vending/databases/veifyapp.db*	
Description	Contains Google play store downloaded APKs
Artifacts	APK package name
8. *Android/vending/databases/autoupdate.db*	
Description	Contains Google play store downloaded APKs
Artifacts	APK package name
9. /system/*app*	

Description	Contains system apps that are pre-installed on the device. Malware may be disguised as a legitimate system app, so it's important to check for any suspicious apps that may have been added
Artifacts	APK package name
10. data/system/package_cache/1	
Description	Contains permissions in system for malicious activities
Artifacts	APK package name
11. Data/app/	
Description	Contains the installation package of all apps
Artifacts	APK package name

Table 5: Targeted Forensic artifacts and paths

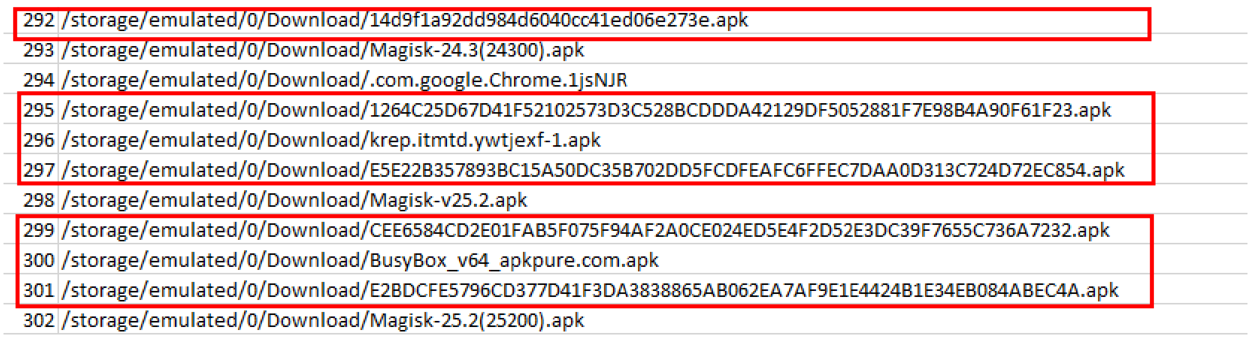
The forensic image which was acquired is analyzed on the basis of above table and paths are compared on the basis of found evidence to filter out more useful artifacts and figure out which type of applications are found in different paths.

1. *Android/ providers/ media/ external.db*	
Artifacts	*Data, size, format, parent, bucket_display_name, data_addded, data_modified, Mime_type, Title, bucket_id, media_type, storage_id*
Found evidence	List all of the APK files are obtained without using the Google Play Store as in figure 8
2. *Android/ vending/ databases/ installqeue.db*	
Artifacts	Reason,package
Found evidence	As seen in figure 9, the APK packages located in external.db are absent from installqeue.db.
3. *Android/ vending/ databases/ library.db*	
Artifacts	Account, doc_id, document_hash, app_certificate_hash
Found evidence	Figure 8 shows that does not include the hash certificate of the discovered APKs.
4. *Android/vending/databases/localappstate.db*	
Artifacts	Package name, download uri, account, title, download and update timestamp, app name
Found evidence	Figure 10 shows that does not include the package names of the discovered APKs.

5. Android/ vending/ databases/ frosting.db	
Artifacts	APK_path, package name
Found evidence	discovered the package names and APK paths for every program, whether it had been downloaded through Google or another source. Some potentially suspicious package names were discovered, and the path for these APKs was data/app/, as seen in figure 11.
6. Android/providers/media/internal.db	
Artifacts	Data added or modified time, app name, title, size
Found evidence	Found no information on APKs obtained outside of the Google Play Store; just APKs installed on the system
7. Android/ vending/ databases/ veifyapp.db	
Artifacts	Application package name
Found evidence	Only contain APK names that are found in figure 8
8. Android/vending/databases/autoupdate.db	
Artifacts	APK package name
Found evidence	Can't find APK data

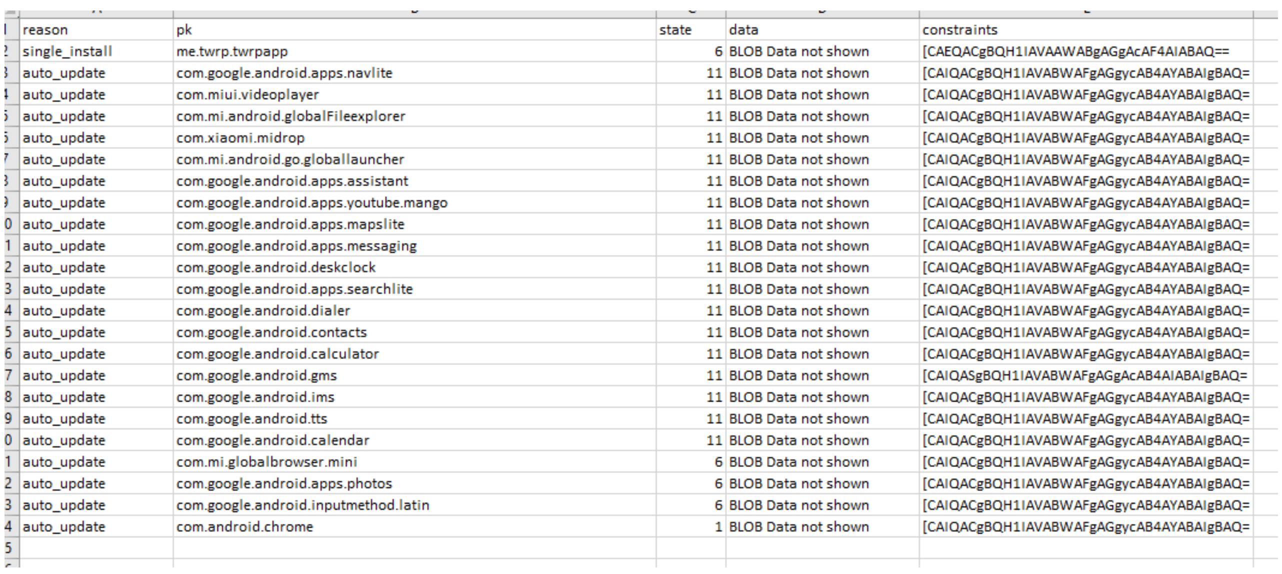
9. /system/app	
Artifacts	APK package name
Found evidence	Can’t find APK data
10. data/system/package_cache/1	
Artifacts	APK package name
Found evidence	Can’t find APK data
11. Data/app/	
Artifacts	APK package name
Found evidence	Can’t find APK data

Table 6: Analysis of forensic image based on table 1.



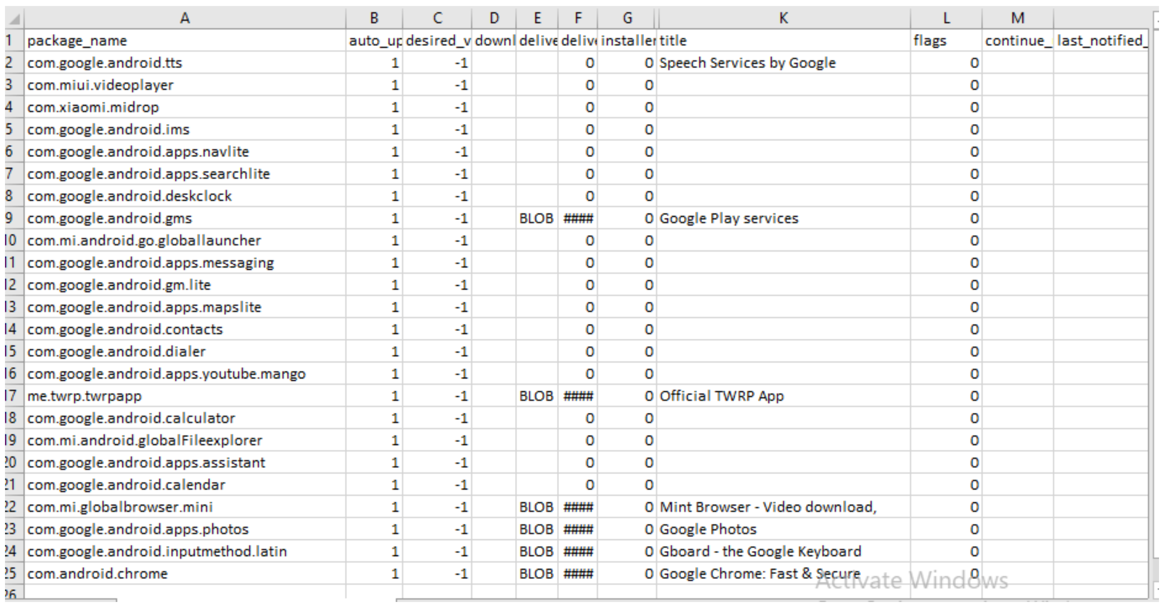
```
292 /storage/emulated/0/Download/14d9f1a92dd984d6040cc41ed06e273e.apk
293 /storage/emulated/0/Download/Magisk-24.3(24300).apk
294 /storage/emulated/0/Download/.com.google.Chrome.1jsNJR
295 /storage/emulated/0/Download/1264C25D67D41F52102573D3C528BCDDDA42129DF5052881F7E98B4A90F61F23.apk
296 /storage/emulated/0/Download/krep.itmtd.ywtjexf-1.apk
297 /storage/emulated/0/Download/E5E22B357893BC15A50DC35B702DD5FCDFEAFC6FFEC7DAA0D313C724D72EC854.apk
298 /storage/emulated/0/Download/Magisk-v25.2.apk
299 /storage/emulated/0/Download/CEE6584CD2E01FAB5F075F94AF2A0CE024ED5E4F2D52E3DC39F7655C736A7232.apk
300 /storage/emulated/0/Download/BusyBox_v64_apkpure.com.apk
301 /storage/emulated/0/Download/E2BDCFE5796CD377D41F3DA3838865AB062EA7AF9E1E4424B1E34EB084ABEC4A.apk
302 /storage/emulated/0/Download/Magisk-25.2(25200).apk
```

Figure 11: *External.db* database artifacts



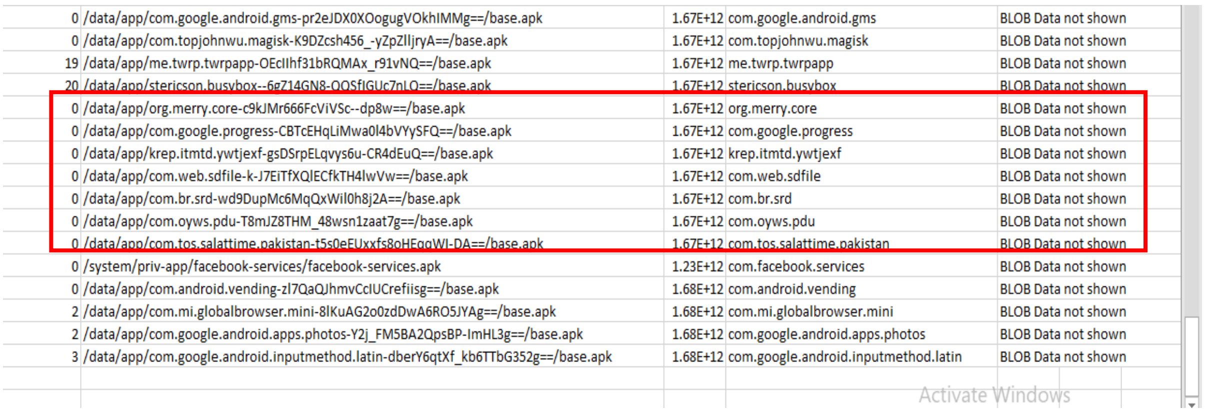
reason	pk	state	data	constraints
single_install	me.twrp.twrpapp	6	BLOB Data not shown	[CAEQACgBQH1IAVAAWABgAGgAcAF4AIABAQ==
auto_update	com.google.android.apps.navlite	11	BLOB Data not shown	[CAIQACgBQH1IAVABWAFgAGgycAB4AYABAIgBAQ=
auto_update	com.miui.videoplayer	11	BLOB Data not shown	[CAIQACgBQH1IAVABWAFgAGgycAB4AYABAIgBAQ=
auto_update	com.mi.android.globalFileexplorer	11	BLOB Data not shown	[CAIQACgBQH1IAVABWAFgAGgycAB4AYABAIgBAQ=
auto_update	com.xiaomi.midrop	11	BLOB Data not shown	[CAIQACgBQH1IAVABWAFgAGgycAB4AYABAIgBAQ=
auto_update	com.mi.android.go.globallauncher	11	BLOB Data not shown	[CAIQACgBQH1IAVABWAFgAGgycAB4AYABAIgBAQ=
auto_update	com.google.android.apps.assistant	11	BLOB Data not shown	[CAIQACgBQH1IAVABWAFgAGgycAB4AYABAIgBAQ=
auto_update	com.google.android.apps.youtube.mango	11	BLOB Data not shown	[CAIQACgBQH1IAVABWAFgAGgycAB4AYABAIgBAQ=
auto_update	com.google.android.apps.mapslite	11	BLOB Data not shown	[CAIQACgBQH1IAVABWAFgAGgycAB4AYABAIgBAQ=
auto_update	com.google.android.apps.messaging	11	BLOB Data not shown	[CAIQACgBQH1IAVABWAFgAGgycAB4AYABAIgBAQ=
auto_update	com.google.android.deskclock	11	BLOB Data not shown	[CAIQACgBQH1IAVABWAFgAGgycAB4AYABAIgBAQ=
auto_update	com.google.android.apps.searchlite	11	BLOB Data not shown	[CAIQACgBQH1IAVABWAFgAGgycAB4AYABAIgBAQ=
auto_update	com.google.android.dialer	11	BLOB Data not shown	[CAIQACgBQH1IAVABWAFgAGgycAB4AYABAIgBAQ=
auto_update	com.google.android.contacts	11	BLOB Data not shown	[CAIQACgBQH1IAVABWAFgAGgycAB4AYABAIgBAQ=
auto_update	com.google.android.calculator	11	BLOB Data not shown	[CAIQACgBQH1IAVABWAFgAGgycAB4AYABAIgBAQ=
auto_update	com.google.android.gms	11	BLOB Data not shown	[CAIQASgBQH1IAVABWAFgAGgAcAB4AIABAIgBAQ=
auto_update	com.google.android.ims	11	BLOB Data not shown	[CAIQACgBQH1IAVABWAFgAGgycAB4AYABAIgBAQ=
auto_update	com.google.android.tts	11	BLOB Data not shown	[CAIQACgBQH1IAVABWAFgAGgycAB4AYABAIgBAQ=
auto_update	com.google.android.calendar	11	BLOB Data not shown	[CAIQACgBQH1IAVABWAFgAGgycAB4AYABAIgBAQ=
auto_update	com.mi.globalbrowser.mini	6	BLOB Data not shown	[CAIQACgBQH1IAVABWAFgAGgycAB4AYABAIgBAQ=
auto_update	com.google.android.apps.photos	6	BLOB Data not shown	[CAIQACgBQH1IAVABWAFgAGgycAB4AYABAIgBAQ=
auto_update	com.google.android.inputmethod.latin	6	BLOB Data not shown	[CAIQACgBQH1IAVABWAFgAGgycAB4AYABAIgBAQ=
auto_update	com.android.chrome	1	BLOB Data not shown	[CAIQACgBQH1IAVABWAFgAGgycAB4AYABAIgBAQ=

Figure 12: *Installqeue.db* database artifacts



A	B	C	D	E	F	G	K	L	M	
package_name	auto_up	desired_v	downl	delive	deliv	installer	title	flags	continue_	last_notified_
com.google.android.tts	1	-1			0	0	Speech Services by Google	0		
com.miui.videoplayer	1	-1			0	0		0		
com.xiaomi.midrop	1	-1			0	0		0		
com.google.android.ims	1	-1			0	0		0		
com.google.android.apps.navlite	1	-1			0	0		0		
com.google.android.apps.searchlite	1	-1			0	0		0		
com.google.android.deskclock	1	-1			0	0		0		
com.google.android.gms	1	-1		BLOB	####	0	Google Play services	0		
com.mi.android.go.globallauncher	1	-1			0	0		0		
com.google.android.apps.messaging	1	-1			0	0		0		
com.google.android.gm.lite	1	-1			0	0		0		
com.google.android.apps.mapslite	1	-1			0	0		0		
com.google.android.contacts	1	-1			0	0		0		
com.google.android.dialer	1	-1			0	0		0		
com.google.android.apps.youtube.mango	1	-1			0	0		0		
me.twrp.twrpapp	1	-1		BLOB	####	0	Official TWRP App	0		
com.google.android.calculator	1	-1			0	0		0		
com.mi.android.globalFileexplorer	1	-1			0	0		0		
com.google.android.apps.assistant	1	-1			0	0		0		
com.google.android.calendar	1	-1			0	0		0		
com.mi.globalbrowser.mini	1	-1		BLOB	####	0	Mint Browser - Video download,	0		
com.google.android.apps.photos	1	-1		BLOB	####	0	Google Photos	0		
com.google.android.inputmethod.latin	1	-1		BLOB	####	0	Gboard - the Google Keyboard	0		
com.android.chrome	1	-1		BLOB	####	0	Google Chrome: Fast & Secure	0		

Figure 13: *Local_appstate.db* database artifacts



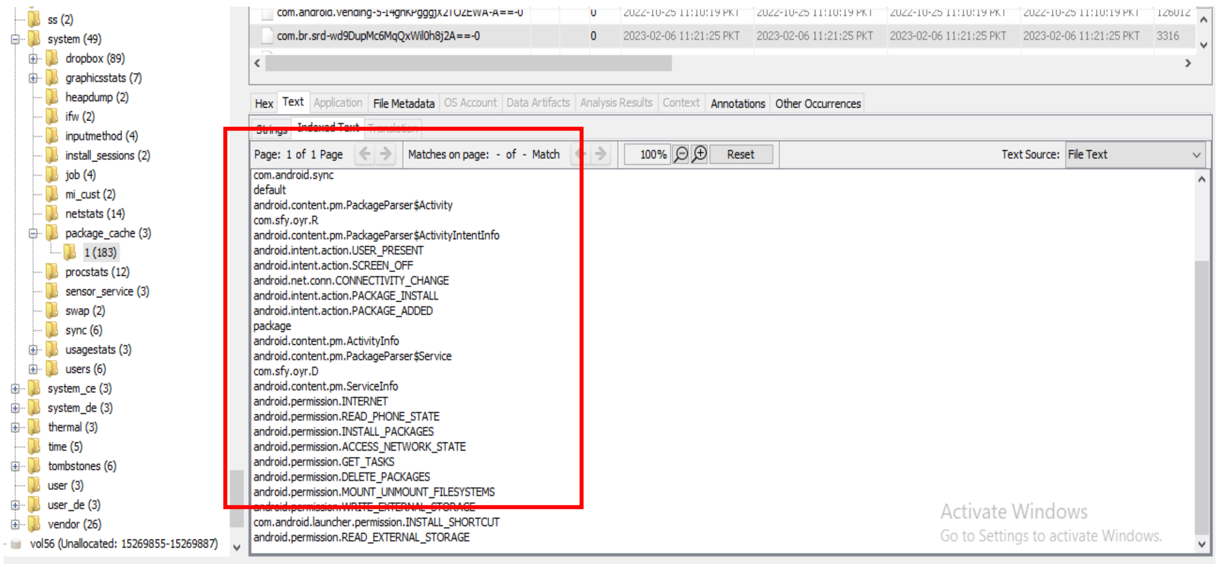
0	/data/app/com.google.android.gms-pr2eJDX0XOogugVOkhIMMg==/base.apk	1.67E+12	com.google.android.gms	BLOB Data not shown
0	/data/app/com.topjohnwu.magisk-K9DZcsh456_-yZpZlljryA==/base.apk	1.67E+12	com.topjohnwu.magisk	BLOB Data not shown
19	/data/app/me.twrp.twrpapp-OEclIhf31bRQMAx_r91vNQ==/base.apk	1.67E+12	me.twrp.twrpapp	BLOB Data not shown
20	/data/app/stericson.busybox--6gZ14GN8-QOSfIGUc7nLQ==/base.apk	1.67E+12	stericson.busybox	BLOB Data not shown
0	/data/app/org.merry.core-c9kJMr666FcViVSc--dp8w==/base.apk	1.67E+12	org.merry.core	BLOB Data not shown
0	/data/app/com.google.progress-CBTcEHqLiMwa0l4bVYySFQ==/base.apk	1.67E+12	com.google.progress	BLOB Data not shown
0	/data/app/krep.itmtd.ywtjexf-gsDSrpELqvys6u-CR4dEuQ==/base.apk	1.67E+12	krep.itmtd.ywtjexf	BLOB Data not shown
0	/data/app/com.web.sdfile-k-J7EiTfXQlECfkTH4lwVw==/base.apk	1.67E+12	com.web.sdfile	BLOB Data not shown
0	/data/app/com.br.srd-wd9DupMc6MqQxWil0h8j2A==/base.apk	1.67E+12	com.br.srd	BLOB Data not shown
0	/data/app/com.oyws.pdu-T8mJZ8THM_48wsn1zaat7g==/base.apk	1.67E+12	com.oyws.pdu	BLOB Data not shown
0	/data/app/com.tos.salattime.pakistan-t5s0eEUxxfs8oHEqqWI-DA==/base.apk	1.67E+12	com.tos.salattime.pakistan	BLOB Data not shown
0	/system/priv-app/facebook-services/facebook-services.apk	1.23E+12	com.facebook.services	BLOB Data not shown
0	/data/app/com.android.vending-zl7QaQJhmvCclUCrefiisg==/base.apk	1.68E+12	com.android.vending	BLOB Data not shown
2	/data/app/com.mi.globalbrowser.mini-8lKuAG2o0zdDwA6RO5JYAg==/base.apk	1.68E+12	com.mi.globalbrowser.mini	BLOB Data not shown
2	/data/app/com.google.android.apps.photos-Y2j_FM5BA2QpsBP-ImHL3g==/base.apk	1.68E+12	com.google.android.apps.photos	BLOB Data not shown
3	/data/app/com.google.android.inputmethod.latin-dberY6qtXf_kb6TTbG352g==/base.apk	1.68E+12	com.google.android.inputmethod.latin	BLOB Data not shown

Figure 14: *Frosting.db* database artifacts

From the above analysis it was figured out that *external.db* only contains the application information which is downloaded from third party and without google play store. While the *internal.db* comprises of that list of APKs which are downloaded from official vendor or google play store. As from the above defined paths we cannot label applications as malicious or benign so there is a need to find artifacts about the permissions required by the applications and their intent, so more artifacts are found in this case.

Path to find intent and permission of these apps:

/img_mobileimage1.dd/vol_vol55/system/package_cache/1/



```
com.br.srd-wd9DupMc6MqQxWil0h8j2A==-0    0    2023-02-06 11:21:25 PKT    2023-02-06 11:21:25 PKT    2023-02-06 11:21:25 PKT    2023-02-06 11:21:25 PKT    3316
```

```
com.android.sync
default
android.content.pm.PackageParser$Activity
com.sfy.oyr.R
android.content.pm.PackageParser$ActivityIntentInfo
android.intent.action.USER_PRESENT
android.intent.action.SCREEN_OFF
android.net.conn.CONNECTIVITY_CHANGE
android.intent.action.PACKAGE_INSTALL
android.intent.action.PACKAGE_ADDED
package
android.content.pm.ActivityInfo
android.content.pm.PackageParser$Service
com.sfy.oyr.D
android.content.pm.ServiceInfo
android.permission.INTERNET
android.permission.READ_PHONE_STATE
android.permission.INSTALL_PACKAGES
android.permission.ACCESS_NETWORK_STATE
android.permission.GET_TASKS
android.permission.DELETE_PACKAGES
android.permission.MOUNT_UNMOUNT_FILESYSTEMS
android.permission.WRITE_EXTERNAL_STORAGE
com.android.launcher.permission.INSTALL_SHORTCUT
android.permission.READ_EXTERNAL_STORAGE
```

Figure 15: investigate application permissions from *package_cache* path.

5.3. Evaluating Research Effectiveness

To evaluate the effectiveness of the research, we have employed various forensic investigation phases and concluded the most useful forensic artifacts and paths to illustrate the generality of the research. In experiments, from the prospect of suspicious application detection, the forensic investigation technique is employed. The benign and malicious applications was compared. The final paths and artifacts are displayed in table below that we will use to conduct forensic investigation.

Based on above framework, the forensic image is acquired, and findings are below:

Paths	Found artifact
Android/providers/media/external.db	1. /storage/emulated/0/WhatsApp/Media/WhatsApp Documents/Sent/1514376339e4a0b4727c6897640c7c3e.APK 2. /storage/emulated/0/Download/14d9f1a92dd984d6040cc41ed06e273e.APK 3. /storage/emulated/0/Download/1264C25D67D41F52102573D3C528BCDDDA42129DF5052881F7E98B4A90F61F23.APK 4. /storage/emulated/0/Download/krep.itmtd.ywtjexf-1.APK 5. /storage/emulated/0/Download/E5E22B357893BC15A50DC35B702DD5FCDFEAFC6FFEC7DAA0D313C724D72EC854.APK 6. /storage/emulated/0/Download/CEE6584CD2E01FAB5F075F94AF2A0CE024ED5E4F2D52E3DC39F7655C736A7232.APK

	7. /storage/emulated/0/Download/E2BDCFE5796CD377D41F3DA3838865AB062EA7AF9E1E4424B1E34EB084ABEC4A.APK 8. /storage/emulated/0/CMA_Zip/Decompressed/janOscorp_20230307/f73ebc6f645926bf8566220b14173df8.APK 9. /storage/emulated/0/Download/julyFacebookCredSteal.zip
Android/vending/databases/installqeue.db	The packages for APK which are found in external.db are not present in installqeue.db 1. com.google.android.apps.youtube.mango 2. com.prisbank.app 3. com.gamma.scan2 4. com.winzip.android 5. com.whatsapp 6. com.zip.unzip.zipextractor.raropener.zipfile
Android/vending/databases/library.db	It does not contain hash certificate of found APKs, it contains the APKs: 1. com.google.android.apps.youtube.mango 2. com.prisbank.app 3. com.gamma.scan2 4. com.winzip.android 5. com.whatsapp 6. com.zip.unzip.zipextractor.raropener.zipfile but there is same pkg of com.gamma.scan2 again at the last but it doesnot contain hash

Android/vending/databases/localappstate.db	It does not contain discovered applications packages APKs in appendix 1
Android/vending/databases/frosting.db	discovered the package names and APK locations for every program, whether it had been downloaded through Google or another source. Some potentially suspicious package names were discovered, and the path for these APKs was data/app/.
Android/providers/media/internal.db	Cant find any data for APKs found in media/external.db
Android/vending/databases/ verifyapp.db	krep.itmtd.ywtjexf com.web.sdfile com.oyws.pdu com.br.srd org.merry.core com.tos.salattime.pakistan com.prisbank.app com.gamma.scan2 com.zip.unzip.zipextractor.raropener.zipfile com.whatsapp com.pcnts.splicingpp com.facebook.system com.enab.lockkeep com.cosmos.starwarz com.facebook.appmanager se.dirac.acs

Android/vending/databases/autoupdate.db	com.google.android.apps.youtube.mango com.whatsapp com.winzip.android com.zip.unzip.zipextractor.raropener.zipfile com.gamma.scan2 com.prisbank.app
*/system/app/**	facebook-appmanager YouTubeGo Application not found there proves that these APKS not disguised as legitimate app
data/system/package_cache/1	Appendix 2
/data/app/<application package name>	com.br.srd-wd9DupMc6MqQxWil0h8j2A== com.cosmos.starwarz-2IB61gP3toiK44zR21mgoQ== com.enab.lockkeep-Y6AzdXU071I5NTKMgb0YWA== com.facebook.appmanager-AbEIncHcjzUsEhHY9bQ2wA== com.facebook.lite-m0tYZsnbkAK52Z_anlyvrg== com.facebook.services-gdIEIrF8IA-mm2Mu682-0g== com.facebook.system-eKOkZWxhXApD6zJ6YPMqLg== com.gamma.scan2-ckccwYHzKBUdNhuZeyqqtQ== com.oyws.pdu-T8mJZ8THM_48wsn1zaat7g== com.pcnts.splicingpp-TxJngHz3tzdzRsNPCWNcsg== com.prisbank.app-TyIM70hfj_oC2C3kDtC8ZA== com.tos.salattime.pakistan-t5s0eEUxxfs8oHEqqWI-DA==

	com.web.sdfile-k-J7EiTfXQlECfkTH4lwVw== com.whatsapp-eHcCvC4QGycO4kAifNvs9g== com.zip.unzip.zipextractor.raropener.zipfile-Wbu8XTlljC7VoMSRwCc-FQ== krep.itmtd.ywtjexf-gsDSrpELqvys6u-CR4dEuQ== org.merry.core-c9kJMr666FcViVSc--dp8w==

Table 7: Analysis of forensic image based on proposed forensic investigation methodology

For the forensic investigation of android mobile device, the total number of apps are checked in the system. The *data/app/* directory contains all the apps and their paths information, there are 41 apps installation files. Now we are interested that how many apps are installed from google play store or third party or downloaded directly from internet and how many are system app. To analyze the apps which are downloaded from google playstore lets check the *Android/vending/databases/installqeue.db* so out of 41 apps only 6 are downloaded from google playstore. In order to find out the download source of rest of the apps *Android/providers/media/external.db* is analyzed and it was revealed that 9 apps were downloaded from the internet directly and source of download was chrome browser. To verify the user- installed in the system analyze the *Android/vending/databases/verifyapp.db*. The package names of installed apps can be found there. Now check the *Android/providers/media/internal.db* to verify the system-installed apps, so it was verified that these 15 apps are not internal system apps. In order to validate whether the apps are legitimate we have to verify the hash certificate of application. Therefore, *Android/vending/databases/library.db* path is analyzed. The apps like Youtube, prisbank, barcode scanner, Whatsapp, winzip and rar opener has the verified hash certificates. And in the *Android/vending/databases/autoupdate.db* it can be verified that these packages can update from google playstore. The total installed apps found from *verifyapp.db* has only 6 apps that have verified certificates and auto-update from google playstore but rest of the 9 apps are still questionable. We can classify these 9 apps as suspicious apps and its necessary to analyze the */system/app/** because Contains system apps that are pre-installed on the device. Malware may be disguised as a legitimate system app, so it's important to check for any suspicious apps that may have been added. After the analysis of this path, it was turned out that only facebook and YouTube are system apps and out of these 9 APKs none of them has disguised as legitimate app.

These suspicious 9 APKs are further investigated on the basis of required android system permission to detect the malwares out of these applications.

5.4. Summary

In this chapter analysis and results achieved during the research is discussed. In the following chapter validation and verification of the achieved results are provided.

Chapter 6

6. Discussion and Analysis

The validation and verification of the data obtained during the experimentation against the suggested framework are covered in this chapter. The cuckoo sandbox is used throughout the investigation to validate the suspicious programs that were filtered out as previously indicated in the chapter. to verify if the apps in our findings are indeed malicious and to assess the efficacy and efficiency of the suggested technique. The outcomes are then contrasted with the benchmark method.

6.1. Overview

This section assesses the efficacy of the studies conducted to identify malware in Internet of Things devices. The device's forensic picture has been obtained and examined using autopsy. The suggested technique was used for the analysis. Nine apps have been eliminated for analysis because of the trial. Ultimately, Table 8's filtered-out programs may be identified from one another by their obtained system permissions. The system permission acquired by malicious and benign malware are different and the study in this paper H. J. Zhu et al. [9] conducted the research on the system permissions acquired by malicious applications. The dangerous system permissions required by the malicious applications are marked in red font. Based on findings of this research paper the applications that have malicious behavior can be separated from benign ones.

APK No.	**Package name**	**APK path**
APK1	com.web.sdfile	*/data/app/com.web.sdfile-k-J7EiTfXQlECfkTH4lwVw==/base.apk*
APK2	com.br.srd	*/data/app/com.br.srd-wd9DupMc6MqQxWil0h8j2A==/base.apk*
APK3	com.oyws.pdu	*/data/app/com.oyws.pdu-T8mJZ8THM_48wsn1zaat7g==/base.apk*
APK4	krep.itmtd.ywtjexf	*/data/app/krep.itmtd.ywtjexf-gsDSrpELqvys6u-CR4dEuQ==/base.apk*
APK5	org.merry.core	*/data/app/org.merry.core-c9kJMr666FcViVSc--dp8w==/base.apk*
APK6	com.tos.salattime	*/data/app/com.tos.salattime.pakistan-t5s0eEUxxfs8oHEqqWI-DA==/base.apk*
APK7	com.facebook.system	*/data/app/com.facebook.system-eKOkZWxhXApD6zJ6YPMqLg==*
APK8	com.enab.lockkeep	*/data/app/com.enab.lockkeep-Y6AzdXU071I5NTKMgb0YWA==/base.apk*
APK9	com.pcnts.splicingpp	*/data/app/com.pcnts.splicingpp-TxJngHz3tzdzRsNPCWNcsg==/base.apk*

APK10	com.snt.rubbishcleaner	*/data/app/com.snt.rubbishcleaner*
APK11	com.cosmos.starwarz	*/data/app/com.cosmos.starwarz-2IB61gP3toiK44zR21mgoQ==/base.apk*
APK12	com.gamma.scan2	*/data/app/com.gamma.scan2-ckccwYHzKBUdNhuZeyqqtQ==/base.apk*
APK13	com.zip.unzip.zipextractor.raropener.zipfile	*/data/app/com.zip.unzip.zipextractor.raropener.zipfile-Wbu8XTlljC7VoMSRwCcFQ==/base*
APK14	com.whatsapp	*/data/app/com.whatsapp-eHcCvC4QGycO4kAifNvs9g==/base.apk*

Table 8:User-installed applications

Permissions	APK1	APK2	APK3	APK4	APK5	APK6	APK7	APK8	APK9	APK10	APK11
MOUNT_UNMOUNT_FILESYSTEMS	✓	✓	✓								
READ_PHONE_STATE	✓	✓	✓	✓	✓	✓					✓
READ_EXTERNAL_STORAGE	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓
ACCESS_NETWORK_STATE	✓	✓	✓	✓	✓	✓	✓			✓	✓
CHANGE_NETWORK_STATE	✓										
ACCESS_WIFI_STATE	✓					✓	✓				
RESTART_PACKAGES	✓			✓							
READ_LOGS	✓			✓							
CHANGE_WIFI_STATE	✓										
RECORD_AUDIO	✓					✓					✓
CAPTURE_AUDIO_OUTPUT						✓					
DISABLE_KEYGUARD	✓										✓
WAKE_LOCK	✓				✓	✓	✓				✓
BLUETOOTH	✓										
GET_PACKAGE_SIZE	✓						✓			✓	

ACCESS_COARSE_LOCATION	✓										
WRITE_SETTINGS	✓										
WRITE_EXTERNAL_STORAGE	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓
WRITE_MEDIA_STORAGE	✓										
READ_CONTACTS	✓			✓	✓	✓					
UNINSTALL_SHORTCUT	✓										
INSTALL_SHORTCUT	✓	✓									
SYSTEM_ALERT_WINDOW	✓			✓				✓			✓
KILL_BACKGROUND_PROCESSES	✓			✓						✓	
CLEAR_APP_CACHE	✓									✓	
RECEIVE_BOOT_COMPLETED	✓		✓				✓				✓
GET_TASKS	✓	✓		✓	✓		✓	✓		✓	
ACTIVITY_RECOGNITION	✓										
READ_SETTINGS	✓										
INSTALL_PACKAGES	✓	✓					✓				
DELETE_PACKAGES	✓	✓					✓				

accelerometer	✓										
FORCE_STOP_PACKAGES	✓										
ACCESS_FINE_LOCATION	✓				✓	✓					
READ_OWNER_DATA	✓		✓								
INTERNET		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
READ_SMS				✓	✓	✓					✓
SEND_SMS				✓	✓						✓
WRITE_SMS				✓							✓
READ_CALL_LOG				✓	✓	✓					
READ_HISTORY_BOOKMARKS				✓							
READ_SYNC_SETTINGS				✓							
READ_CALENDAR				✓							
READ_PROFILE											
SET_ALARM				✓		✓					
RECEIVE_SMS				✓	✓	✓					✓
RECEIVE_MMS											✓
RECEIVE						✓					
VIBRATE				✓		✓		✓			
CALL_PHONE				✓							✓

ACCESS_MOCK_LOCATION					✓						
ACCESS_LOCATION_EXTRA_COMMANDS					✓						
BIND_JOB_SERVICE						✓					
FOREGROUND_SERVICE						✓		✓			
SET_WALLPAPER						✓					
READ_GMAIL						✓					
GET_ACCOUNTS						✓					
AUTHENTICATE_ACCOUNTS						✓					
USE_CREDENTIALS						✓					
ACCESS_NOTIFICATION_POLICY						✓					
STORAGE						✓					
BIND_GET_INSTALL_REFERRER_SERVICE						✓		✓	✓	✓	
DOWNLOAD_WITHOUT_NOTIFICATION							✓				
CHANGE_COMPONENT_ENABLED_STATE							✓				
REAL_GET_TASKS							✓				

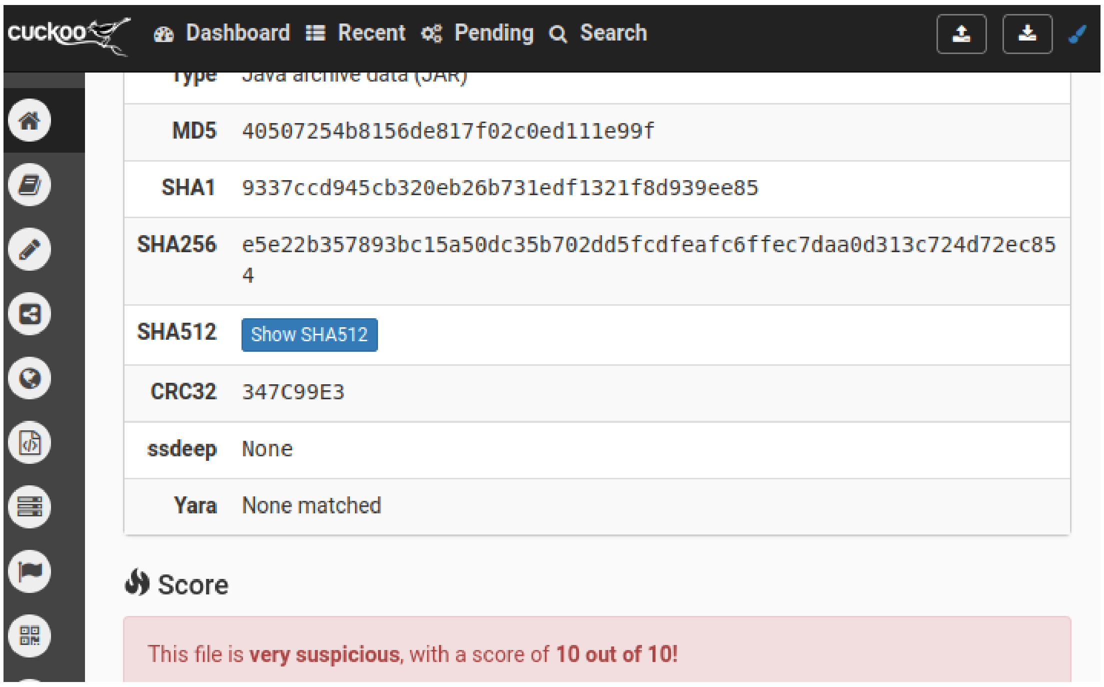
SYSTEM_OVERLAY_WINDOW								✓			
CAMERA								✓			✓
PACKAGE_USAGE_STATS								✓		✓	✓
FLASHLIGHT								✓			
MODIFY_AUDIO_SETTINGS								✓			✓
REQUEST_DELETE_PACKAGES											✓
READ_PRIVILEGED_PHONE_STATE											✓
REQUEST_IGNORE_BATTERY_OPTIMIZATIONS											✓
INJECT_EVENTS											✓
ACCESS_SUPERUSER											✓
REQUEST_INSTALL_PACKAGES											✓

Table 9: System permissions required by APKs

Table 7 contains the android system permissions required by all the applications mentioned in Table 6. These tables are systematically analyzed based on the dangerous and non-dangerous android permissions they require. The dangerous android system permissions include READ_PHONE_STATE, SET_ALARM, READ_EXTERNAL_STORAGE, INSTALL_PACKAGES, RECEIVE_SMS, SET_ALARM, ACCESS_FINE_LOCATION, WRITE_SECURE_SETTINGS, GET_ACCOUNTS, UPDATE_DEVICE_STATS, READ_CONTACTS, READ_HISTORY_BOOKMARKS,GET_ACCOUNTS, READ_SMS, ACCESS_COARSE_LOCATION, SEND_SMS READ_CALL_LOG, WRITE_HISTORY_BOOKMARKS, and ACCESS_NOTIFICATION_POLICY. and. In the above Table 7, the APKs which contain dangerous android permissions are classified as malicious. The APKs which do not require dangerous permissions, or no permissions are classified as benign. The APKs12, APK13 and APK14 do not require system permissions so they can be classified as benign. Out of the 14 APKs the APK7, APK12, APK13 and APK14 are labelled as benign and the rest of 10 APKs are malicious.

6.2. Validation with Cuckoo Sandbox

The 10 APKs are found suspicious after the above analysis. To prove the results of methodology, detected APKs are validated through Cuckoo sandbox. The validation phase will prove the effectiveness of results by the proposed methodology.



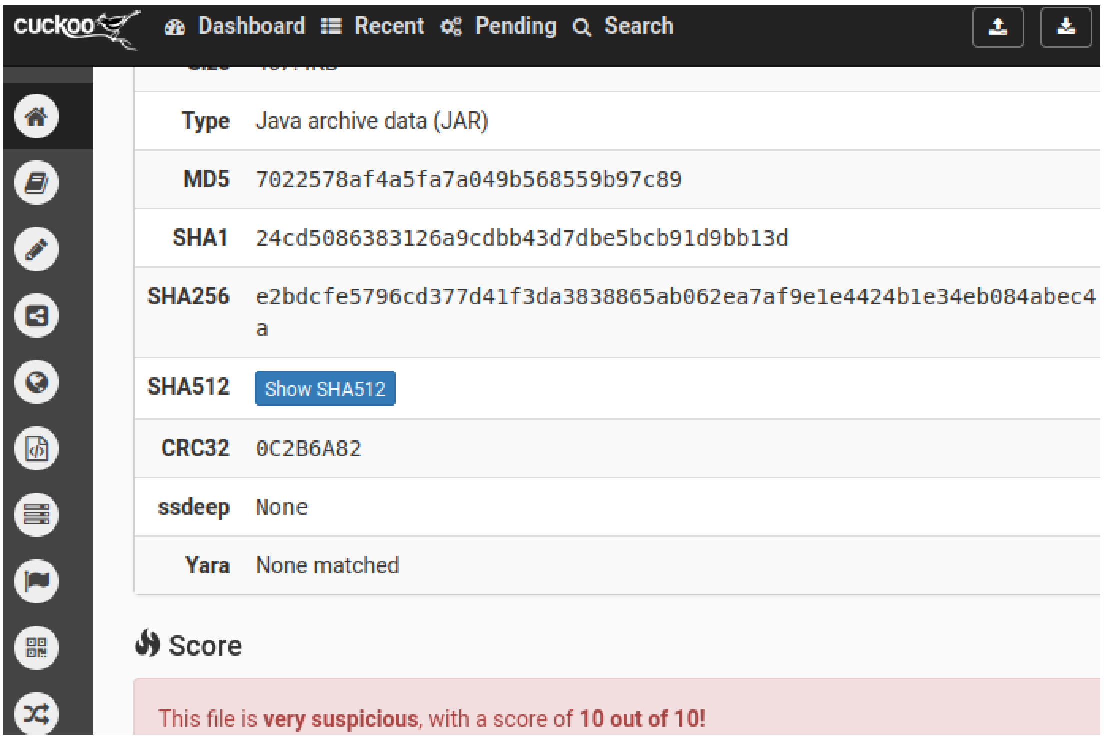
Type	Java archive data (JAR)
MD5	40507254b8156de817f02c0ed111e99f
SHA1	9337ccd945cb320eb26b731edf1321f8d939ee85
SHA256	e5e22b357893bc15a50dc35b702dd5fcdfeafc6ffec7daa0d313c724d72ec854
SHA512	Show SHA512
CRC32	347C99E3
ssdeep	None
Yara	None matched

Score

This file is **very suspicious**, with a score of **10 out of 10!**

Figure 16: Cuckoo Sandbox suspicious APK score validation of APK1

The above APK is classified as malicious by cuckoo sandbox as well.



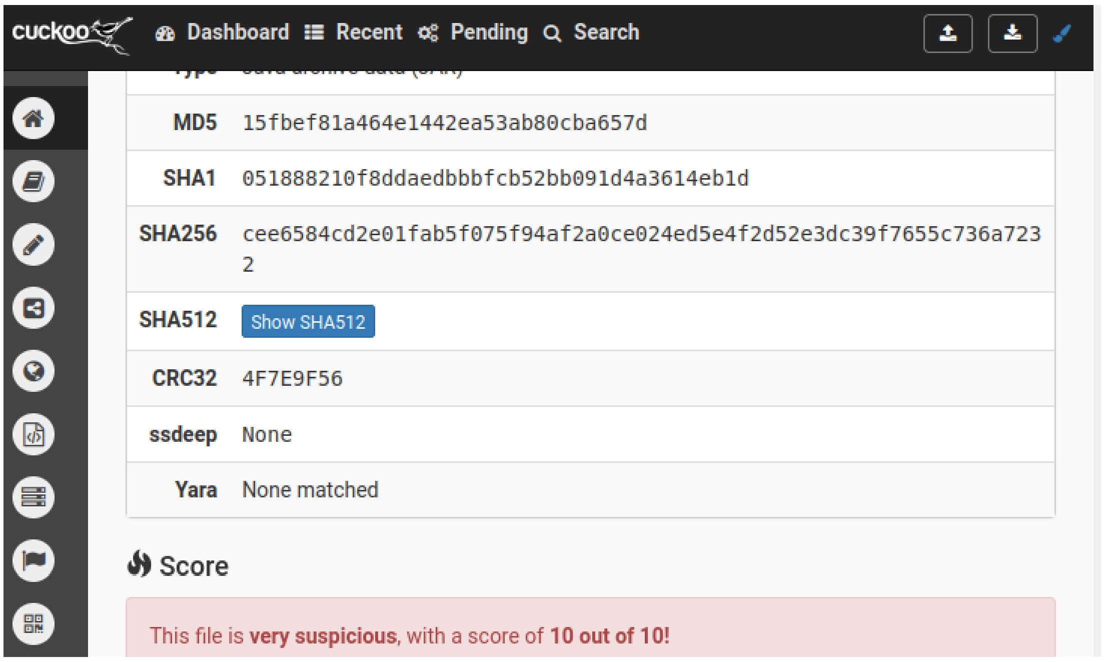
Type	Java archive data (JAR)
MD5	7022578af4a5fa7a049b568559b97c89
SHA1	24cd5086383126a9cdbb43d7dbe5bcb91d9bb13d
SHA256	e2bdcfe5796cd377d41f3da3838865ab062ea7af9e1e4424b1e34eb084abec4a
SHA512	Show SHA512
CRC32	0C2B6A82
ssdeep	None
Yara	None matched

Score

This file is **very suspicious**, with a score of **10 out of 10!**

Figure 17: Validation score of APK2 by Cuckoo Sandbox

The APK validation score classified APK2 as malicious.



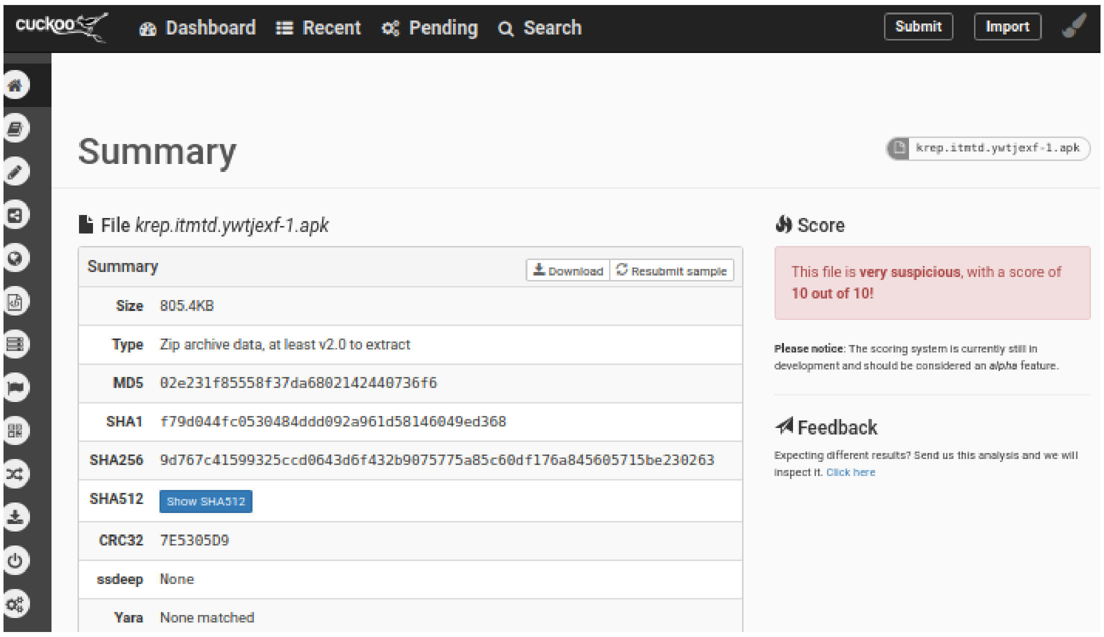
MD5	15fbef81a464e1442ea53ab80cba657d
SHA1	051888210f8ddaedbbbfcb52bb091d4a3614eb1d
SHA256	cee6584cd2e01fab5f075f94af2a0ce024ed5e4f2d52e3dc39f7655c736a7232
SHA512	Show SHA512
CRC32	4F7E9F56
ssdeep	None
Yara	None matched

Score

This file is **very suspicious**, with a score of **10 out of 10!**

Figure 18: Cuckoo Sandbox Validation of APK3

The APK3 is identified as malicious by cuckoo sandbox as well.



Summary

krep.itmtd.ywtjexf-1.apk

File *krep.itmtd.ywtjexf-1.apk*

Summary

Size	805.4KB
Type	Zip archive data, at least v2.0 to extract
MD5	02e231f85558f37da6802142440736f6
SHA1	f79d044fc0530484ddd092a961d58146049ed368
SHA256	9d767c41599325ccd0643d6f432b9075775a85c60df176a845605715be230263
SHA512	Show SHA512
CRC32	7E5305D9
ssdeep	None
Yara	None matched

Score

This file is **very suspicious**, with a score of **10 out of 10!**

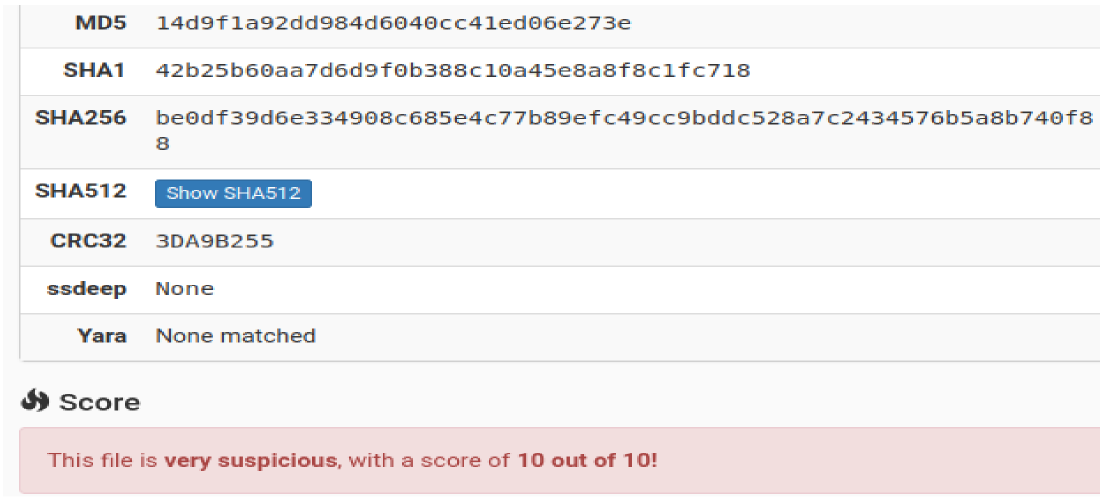
Please notice: The scoring system is currently still in development and should be considered an *alpha* feature.

Feedback

Expecting different results? Send us this analysis and we will inspect it. Click here

Figure 19: Cuckoo Sandbox validation of APK4 as malicious

Classification of APK4 as malicious by cuckoo sandbox.



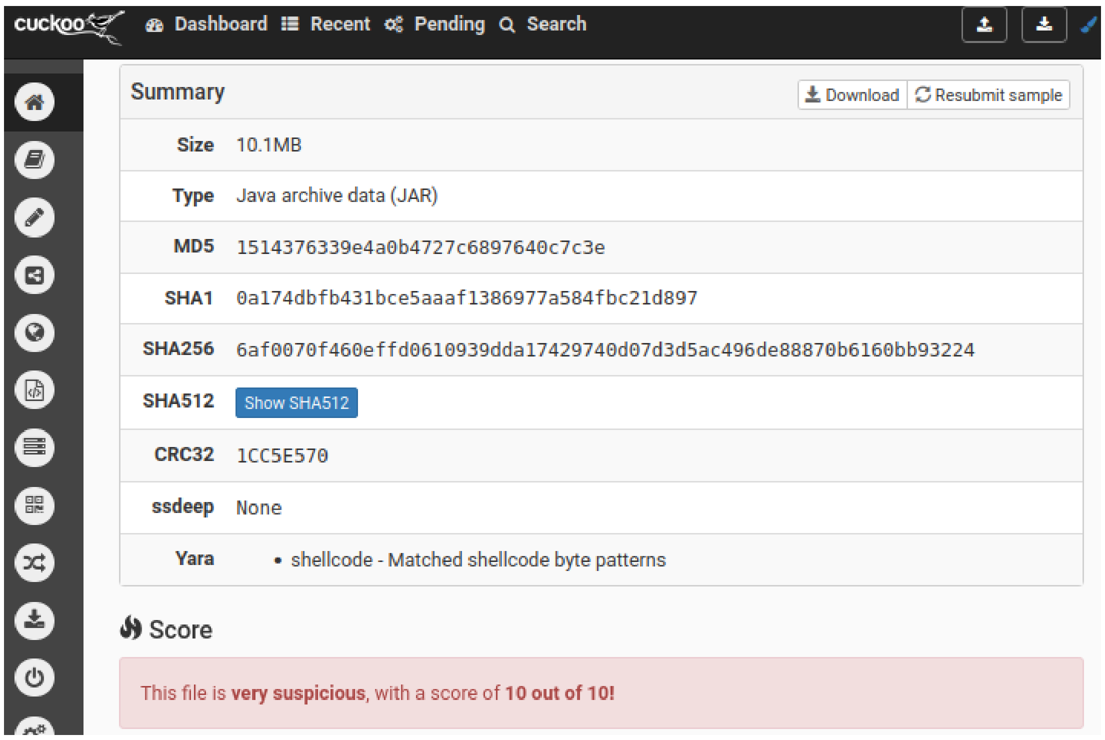
MD5	14d9f1a92dd984d6040cc41ed06e273e
SHA1	42b25b60aa7d6d9f0b388c10a45e8a8f8c1fc718
SHA256	be0df39d6e334908c685e4c77b89efc49cc9bddc528a7c2434576b5a8b740f8 8
SHA512	Show SHA512
CRC32	3DA9B255
ssdeep	None
Yara	None matched

Score

This file is **very suspicious**, with a score of **10 out of 10!**

Figure 20: APK5 identified suspicious by Cuckoo Sandbox

The APK5 is classified as malicious by cuckoo sandbox as well.



Summary

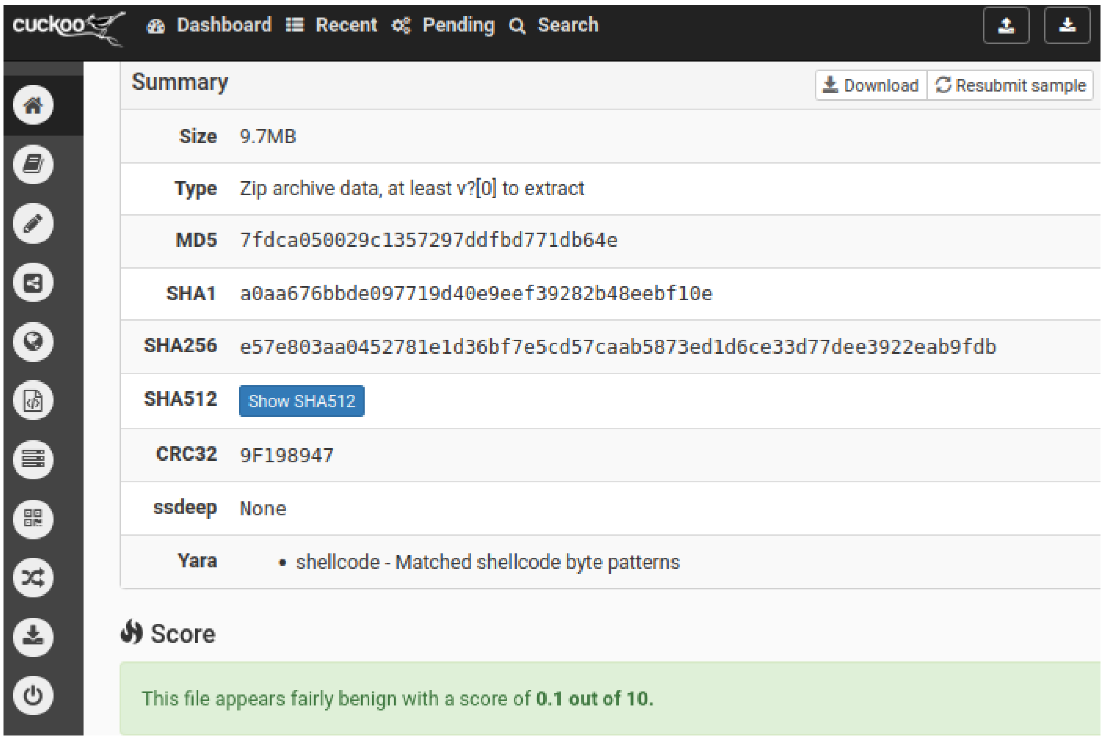
Size	10.1MB
Type	Java archive data (JAR)
MD5	1514376339e4a0b4727c6897640c7c3e
SHA1	0a174dbfb431bce5aaaf1386977a584fbc21d897
SHA256	6af0070f460effd0610939dda17429740d07d3d5ac496de88870b6160bb93224
SHA512	Show SHA512
CRC32	1CC5E570
ssdeep	None
Yara	• shellcode - Matched shellcode byte patterns

Score

This file is **very suspicious**, with a score of **10 out of 10!**

Figure 21: APK6 validation by Cuckoo Sandbox

The APK5 is validated as malicious by cuckoo sandbox.



Summary

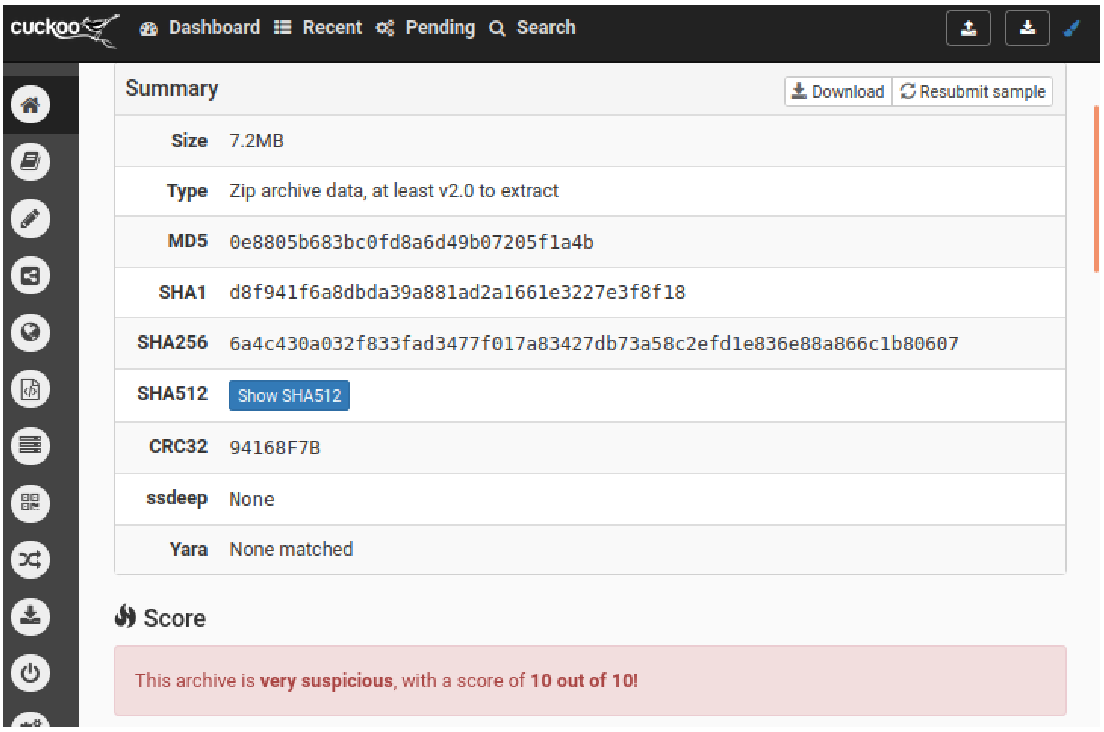
Size	9.7MB
Type	Zip archive data, at least v?[0] to extract
MD5	7fdca050029c1357297ddfbd771db64e
SHA1	a0aa676bbde097719d40e9eef39282b48eebf10e
SHA256	e57e803aa0452781e1d36bf7e5cd57caab5873ed1d6ce33d77dee3922eab9fdb
SHA512	Show SHA512
CRC32	9F198947
ssdeep	None
Yara	• shellcode - Matched shellcode byte patterns

Score

This file appears fairly benign with a score of **0.1 out of 10.**

Figure 22: Validation of APK7 by Cuckoo Sandbox

The APK7 is classified as benign because its malicious score is 0.1/10 by cuckoo sandbox as well.



Summary

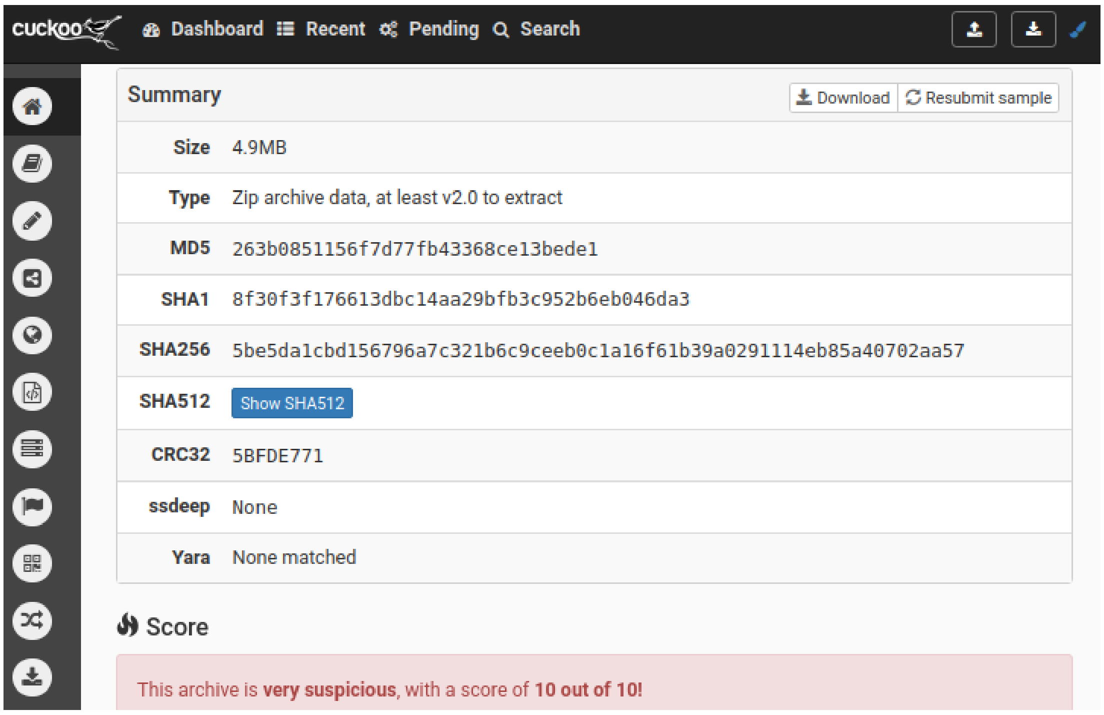
Size	7.2MB
Type	Zip archive data, at least v2.0 to extract
MD5	0e8805b683bc0fd8a6d49b07205f1a4b
SHA1	d8f941f6a8dbda39a881ad2a1661e3227e3f8f18
SHA256	6a4c430a032f833fad3477f017a83427db73a58c2efd1e836e88a866c1b80607
SHA512	Show SHA512
CRC32	94168F7B
ssdeep	None
Yara	None matched

Score

This archive is **very suspicious**, with a score of **10 out of 10!**

Figure 23: Validation by Cuckoo Sandbox for APK8

The APK8 is categorized as malicious by cuckoo sandbox.



Summary

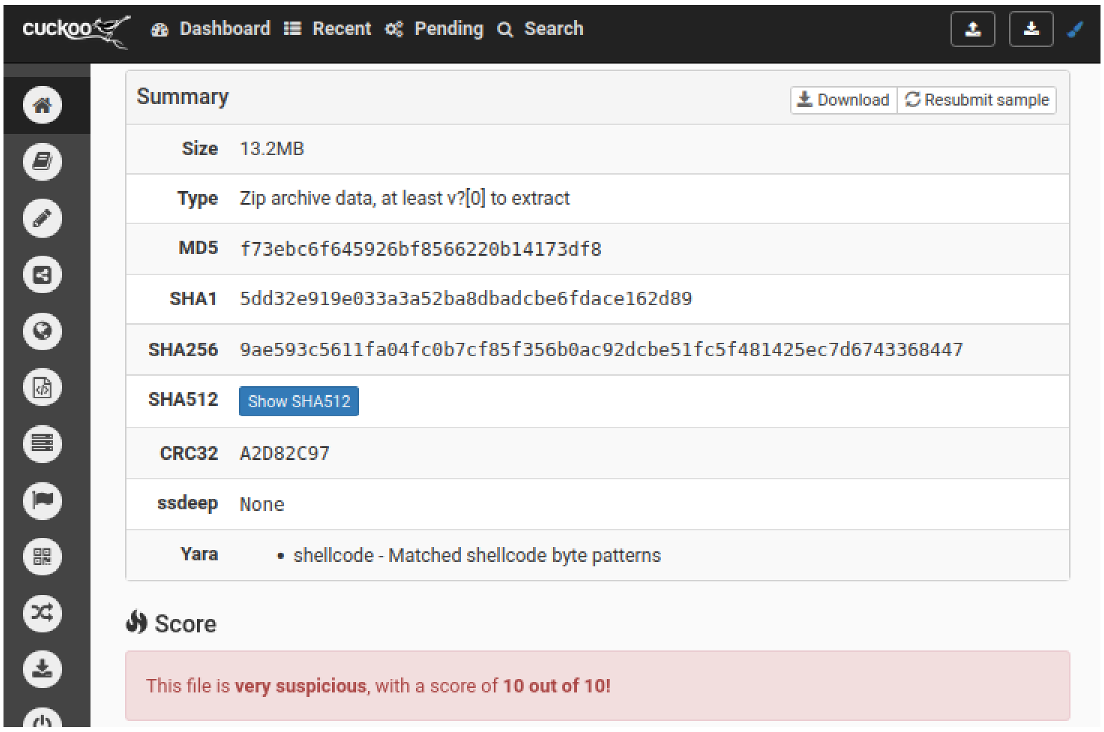
Size	4.9MB
Type	Zip archive data, at least v2.0 to extract
MD5	263b0851156f7d77fb43368ce13bede1
SHA1	8f30f3f176613dbc14aa29bfb3c952b6eb046da3
SHA256	5be5da1cbd156796a7c321b6c9ceeb0c1a16f61b39a0291114eb85a40702aa57
SHA512	Show SHA512
CRC32	5BFDE771
ssdeep	None
Yara	None matched

Score

This archive is **very suspicious**, with a score of **10 out of 10!**

Figure 24: APK9 validated by Cuckoo Sandbox

The APK9 malicious score is 10/10 by cuckoo sandbox.



Summary

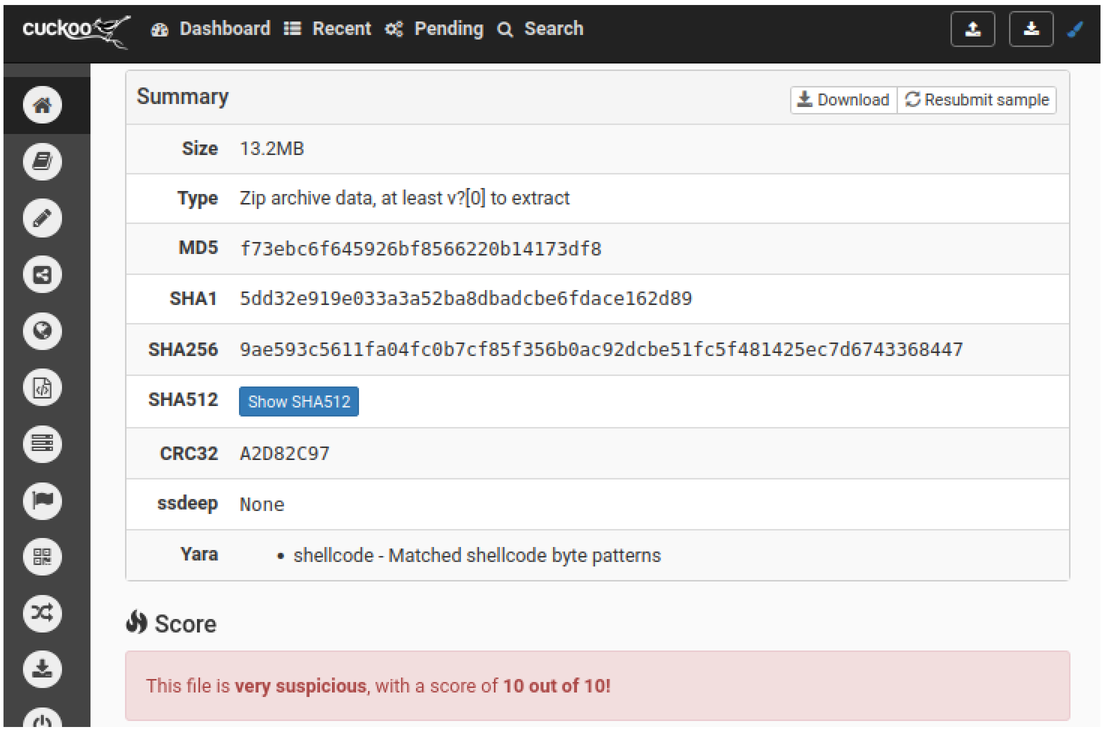
Size	13.2MB
Type	Zip archive data, at least v?[0] to extract
MD5	f73ebc6f645926bf8566220b14173df8
SHA1	5dd32e919e033a3a52ba8dbadcbe6fdace162d89
SHA256	9ae593c5611fa04fc0b7cf85f356b0ac92dcbe51fc5f481425ec7d6743368447
SHA512	Show SHA512
CRC32	A2D82C97
ssdeep	None
Yara	• shellcode - Matched shellcode byte patterns

Score

This file is **very suspicious**, with a score of **10 out of 10!**

Figure 25: Validation of APK10 by Cuckoo Sandbox

The APK10 malicious score is 10/10 by cuckoo sandbox.



Summary

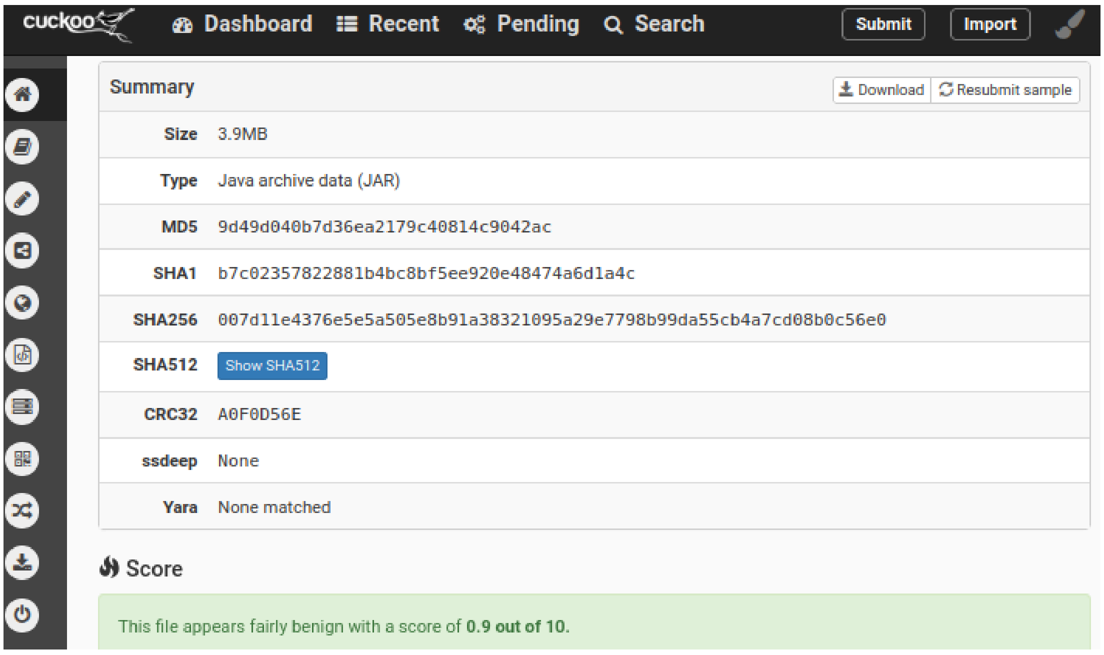
Size	13.2MB
Type	Zip archive data, at least v?[0] to extract
MD5	f73ebc6f645926bf8566220b14173df8
SHA1	5dd32e919e033a3a52ba8dbadcbe6fdace162d89
SHA256	9ae593c5611fa04fc0b7cf85f356b0ac92dcbe51fc5f481425ec7d6743368447
SHA512	Show SHA512
CRC32	A2D82C97
ssdeep	None
Yara	• shellcode - Matched shellcode byte patterns

Score

This file is **very suspicious**, with a score of **10 out of 10!**

Figure 26: APK11 Cuckoo Sandbox validation

Cuckoo sandbox identified APK11 as malicious.



Summary

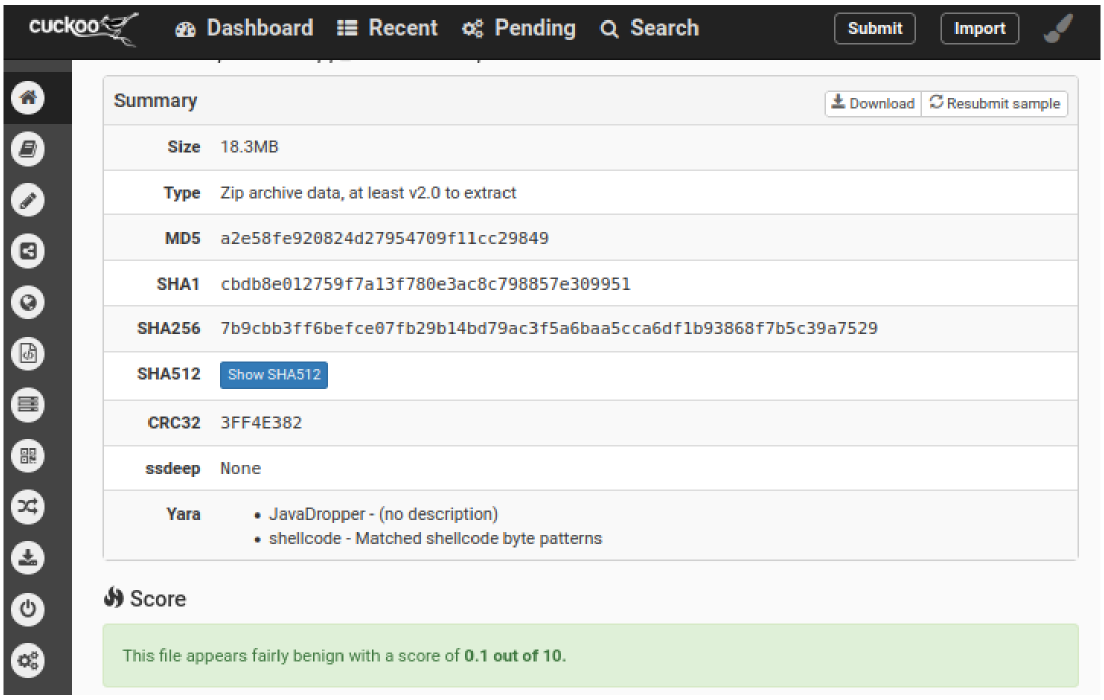
Size	3.9MB
Type	Java archive data (JAR)
MD5	9d49d040b7d36ea2179c40814c9042ac
SHA1	b7c02357822881b4bc8bf5ee920e48474a6d1a4c
SHA256	007d11e4376e5e5a505e8b91a38321095a29e7798b99da55cb4a7cd08b0c56e0
SHA512	Show SHA512
CRC32	A0F0D56E
ssdeep	None
Yara	None matched

Score

This file appears fairly benign with a score of **0.9 out of 10.**

Figure 27: APK12 validated by Cuckoo Sandbox

The APK12 is classified as benign cuckoo sandbox.



Summary

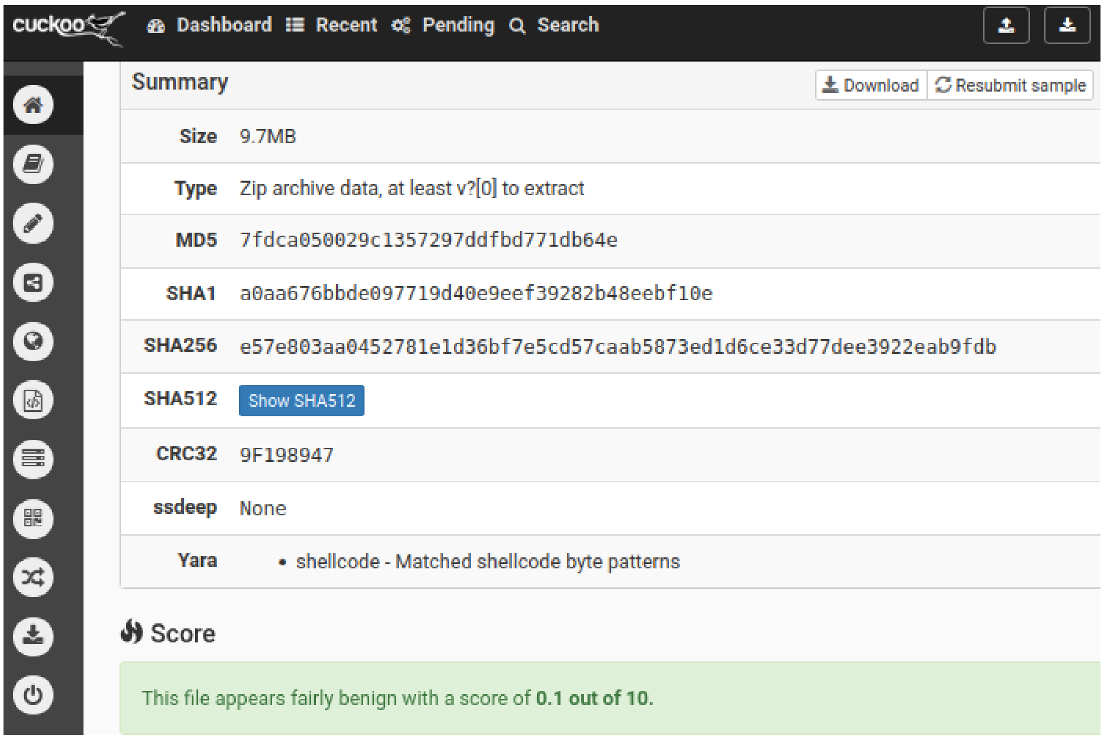
Size	18.3MB
Type	Zip archive data, at least v2.0 to extract
MD5	a2e58fe920824d27954709f11cc29849
SHA1	cbdb8e012759f7a13f780e3ac8c798857e309951
SHA256	7b9cbb3ff6befce07fb29b14bd79ac3f5a6baa5cca6df1b93868f7b5c39a7529
SHA512	Show SHA512
CRC32	3FF4E382
ssdeep	None
Yara	• JavaDropper - (no description) • shellcode - Matched shellcode byte patterns

Score

This file appears fairly benign with a score of **0.1 out of 10.**

Figure 28: Cuckoo Sandbox Validation of APK13

Based on malicious score APK13 is classified as benign by cuckoo sandbox.



Summary

Size	9.7MB
Type	Zip archive data, at least v?[0] to extract
MD5	7fdca050029c1357297ddfbd771db64e
SHA1	a0aa676bbde097719d40e9eef39282b48eebf10e
SHA256	e57e803aa0452781e1d36bf7e5cd57caab5873ed1d6ce33d77dee3922eab9fdb
SHA512	Show SHA512
CRC32	9F198947
ssdeep	None
Yara	• shellcode - Matched shellcode byte patterns

Score

This file appears fairly benign with a score of **0.1 out of 10.**

Figure 29: APK14 Validation by Cuckoo Sandbox

The APK14 malicious score is 0.1/10, it is identified as benign by cuckoo sandbox as well.

APK No.	**Cuckoo Sandbox outcome**	**Proposed methodology Outcome**	**Validation**
APK1	Malicious	Malicious	Match
APK2	Malicious	Malicious	Match
APK3	Malicious	Malicious	Match
APK4	Malicious	Malicious	Match
APK5	Malicious	Malicious	Match
APK6	Malicious	Malicious	Match
APK7	Benign	Benign	Match
APK8	Malicious	Malicious	Match
APK9	Malicious	Malicious	Match
APK10	Malicious	Malicious	Match
APK11	Malicious	Malicious	Match
APK12	Benign	Benign	Match
APK13	Benign	Benign	Match
APK14	Benign	Benign	Match
Result Accuracy			**100 %**

Table 10: Summarized results of validation by Cuckoo sandbox

6.3. Discussion

The data/data, data/app, data/system and system/app provided the most important paths and artifacts. These directories are comprised of 11 important paths listed in Table 1 to construct the sequence of evidence. The identified artifacts in Table 1 helped to classify the benign and malicious applications. In the above experiment 15 user-installed apps were analyzed by following the methodology, and later on the suspicious apps were validated by the cuckoo sandbox as if they are real malwares.

The methodology presented in this thesis contributes to significant advancement in smart device security by addressing security challenges within smart devices. In this study, the malware threats in the evolving environment of smart devices are tackled by a comprehensive digital forensic investigation framework. The methodology systematically analyzes location, paths and files that contain necessary information about artifacts of installed applications in the smart device to identify and detect the real malwares traces.

A robust incident response framework is designed by implementing this methodology, which detects multiple malwares in smart devices efficiently. The framework comprises proactive, reactive, and forensic phases. In the forensic phase the real-time forensic investigation is carried out that aligns with NIST forensic process. Each phase of framework is followed strictly to ensure filtering out of benign from malicious applications.

Particularly, the approach efficiently filters out suspicious apps that require deeper investigation. In this way, the methodology addresses the issues of identifying malicious applications from numerous other applications in smart device. The effectiveness of this technique is further strengthened using tools like Autopsy to analyze the artifacts and Cuckoo sandbox to validate forensic findings.

In a nutshell the proposed technique offers a framework that provides investigators with the resources they need to effectively identify, categorize, and filter the most recent malware variants that attack smart devices. This method not only helps to improve smart security, yet it also offers a model for future incident response in the ever-changing world of connected devices. Proposed methodology will be crucial in preserving the integrity and security of the smart ecosystem as it grows.

6.4. Comparison with Benchmark Approach

For comparison, the performance of the proposed approach is compared with Dohyun Kim et. al.'s [1] approach. The Dohyun et. al. [1] analyzed the only four malwares i.e. SPAp.apk, GMS.apk, V3Plus.apk and 23983JJF.apk. On the other hand, in our proposed approach multiple latest malwares are analyzed. The most recent Android device easily implements our suggested methods for malware analysis. We also contrasted the malicious and benign programs, in contrast to the benchmark technique. The contribution is further enhanced by our suggested technique, which offers additional artifacts and routes helpful for effective malicious program identification. The system permissions that the apps that are suspected of being malicious have obtained are used to confirm their identity. Furthermore, the Cuckoo Sandbox verifies malicious programs as well.

	Dohyun et al. (2020)	Juanru LI et al. (2012)	Zainab et al. (2023)	J. Lee et al. (2019)	Proposed method
Proposed incident response model	No	No	No	No	Yes
Number of malwares investigated	4	1	0	1	10
Identified Paths	7	0	11	4	11

Comparison of benign and malicious applications	No	Yes	No	yes	Yes
System-installed APKs identification	No	Yes	No	No	Yes
User-installed APKs identification	No	Yes	No	Yes	Yes
Real-time android device forensic	Yes	No	Yes	No	Yes
Investigation of D1	Yes	No	No	No	Yes
Investigation of D2	No	No	Yes	No	Yes
Investigation of D3	No	No	Yes	Yes	Yes

Investigation of D4	Yes	No	Yes	Yes	Yes
Investigation of D5	No	No	Yes	Yes	Yes
Investigation of D6	Yes	No	No	No	No
Investigation of D7	No	No	Yes	No	Yes
Investigation of D8	No	No	Yes	No	Yes
Investigation of D9	No	No	No	No	Yes
Investigation of D10	No	Yes	No	No	Yes
Investigation of D11	Yes	No	Yes	No	Yes
Cuckoo sandbox validation of suspicious APKs	No	No	No	No	Yes

Followed NIST framework	No	No	No	No	Yes

Table 11: Comparison table proposed VS existing method

6.5. Summary

We have outlined the key findings in this chapter and used the validations to support our conclusions. The outcomes derived from our research have been contrasted with those from the benchmark. Additional uses of the suggested methodology have been considered. Future research and the conclusion are covered in the next chapter.

Chapter 7

7. Conclusion & Future Work

Chapter 7 concludes the presented thesis and highlights potential future research directions. It describes different research prospects of our research and identifies open research problems that still need to be solved by the research community.

7.1. Conclusion

The smart devices ecosystem is currently under severe security threat from smart device malware. The efficiency of forensic detection approaches must be increased to meet these problems, with a general focus on identifying malicious apps and the successful operation of the chosen methodology to contrast benign and malicious programs. In this study, we examined the smart device ecosystem to demonstrate that forensic inquiry may lead to improved outcomes. To evaluate the effectiveness of the research, various forensic investigation phases were employed and concluded the most useful forensic artifacts and paths.

With this forensic investigation methodology, we performed a comparison between benign and malicious applications. The obtained results compared with the benchmark and existing methods, it was observed that the proposed strategy achieved improved level of detections and is also capable to further improve it by suggesting more artifacts and paths. The proposed framework detected multiple malwares in the latest android version devices.

Experiment results show that the 9 malicious APKs have been separated out from 5 benign applications. This classification was conducted by following the proposed methodology. Later, the results of the experiment are validated through Cuckoo Sandbox. The suspected APKs which are validated through cuckoo sandbox turned out as very dangerous applications.

7.2. Limitation & Future Work

The major limitation of the proposed work is that we only use the android mobile device for forensic investigation to detect malware. Other smart devices such as smart TVs, smart watches, smart homes assistance devices, smart cameras, tablets, laptops etc. need to be forensically investigated. Although forensic investigation methodology can help distinguish the apps over platforms other than android as well. However, to provide a complete solution in terms of detection, malware analysis needs to handle other platforms like ROS, Tarzen as well; therefore, the forensic investigation of other platforms would be targeted in the future. We would also like to explore other methodologies for malware detection while enhancing the no. of artifacts.

Bibliography

[1] Kim, Dohyun, Yi Pan, and Jong Hyuk Park. "A study on the digital forensic investigation method of clever malware in IoT devices." *IEEE Access* 8 (2020): 224487-224499.

[2] J. Milosevic, F. Regazzoni and M. Malek, “Malware Threats and Solutions for Trustworthy Mobile Systems Design”, Hardware Security and Trust, Design and Deployment of Integrated Circuits in a Threatened Environment, Switzerland: Springer, 2017, pp. 149-157

[3] G. Kalogeridou, N. Sklavos, A.W. Moore, O Koufopavlou, “On the Hardware Trojans Detection, using Mixed-Signal ICs”, proceedings, workshop on Trustworthy Manufacturing and Utilization of Secure Devices, Conference DATE 2015, Grenoble, France, March 9-13, 2015

[4] Nokia, “Nokia Threat Intelligence Report”, 2016.

[5] Kshertri, Nir: 'Can Blockchain Strengthen the Internet of Things'. IEEE Access, vol 19. Pp. 68-72. 17 August2017

[6] Meng, Weizhi. 'When Intrusion Detection Meets Blockchain Technology: A Review. IEEE Access. Pp 1 -10. 21 January 2018.

[7] Meng, Weizhi. 'When Intrusion Detection Meets Blockchain Technology: A Review. IEEE Access. Pp 1 -10. 21 January 2018.

[8] Clincy, Victor, and Hossain Shahriar. "IoT malware analysis." *2019 IEEE 43rd annual computer software and applications conference (COMPSAC)*. Vol. 1. IEEE, 2019.

[9] Zhang, Xiaolu, et al. "Iot botnet forensics: A comprehensive digital forensic case study on mirai botnet servers." *Forensic Science International: Digital Investigation* 32 (2020): 300926.

[10] Asma Razgallah, Raphaël Khoury, Sylvain Hallé, Kobra Khanmohammadi, "A survey of malware detection in Android apps: Recommendations and perspectives for future research", Computer Science Review 39 (2021) 100358

[11] Kebande, Victor R., and Indrakshi Ray. "A generic digital forensic investigation framework for internet of things (iot)." *2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud)*. IEEE, 2016.

[12] Kebande, Victor R., et al. "Towards an integrated digital forensic investigation framework for an IoT-based ecosystem." *2018 IEEE International Conference on Smart Internet of Things (SmartIoT)*. IEEE, 2018.

[13] Al-Masri, Eyhab, Yan Bai, and Juan Li. "A fog-based digital forensics investigation framework for IoT systems." *2018 IEEE international conference on smart cloud (SmartCloud)*. IEEE, 2018.

[14] Lee, Jinwoo, et al. "Analysis of application installation logs on android systems." *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. 2019.

[15] Khalid, Zainab, et al. "Forensic investigation of Cisco WebEx desktop client, web, and Android smartphone applications." *Annals of Telecommunications* 78.3-4 (2023): 183-208.

[16] D. Goel and A. K. Jain, ``Smishing-classi_er: A novel framework for detection of smishing attack in mobile environment," in *Proc. Int. Conf. Next Gener. Comput. Technol.*, Singapore, Oct. 2017, pp. 502_512.

[17] J. W. Joo, S. Y. Moon, S. Singh, and J. H. Park, ``S-detector: An enhanced security model for detecting Smishing attack for mobile computing," Telecommun. Syst., vol. 66, no. 1, pp. 29_38, Sep. 2017.

[18] A. K. Jain and B. B. Gupta, ``Rule-based framework for detection of Smishing messages in mobile environment," Procedia Comput. Sci., vol. 125, pp. 617_623, 2018.

[19] S.Wang, Z. Chen, Q.Yan, B.Yang, L. Peng, and Z. Jia, ``Amobile malware detection method using behavior features in network traf_c," J. Netw. Comput. Appl., vol. 133, pp. 15_25, May 2019.

[20] M. Alazab, M. Alazab, A. Shalaginov, A. Mesleh, and A. Awajan, ``Intelligent mobile malware detection using permission requests and API calls," Future Gener. Comput. Syst., vol. 107, pp. 509_521, Jun. 2020.

[21] Autopsy. "Autopsy | Digital Forensics." *Autopsy*, 2023, www.autopsy.com/.

[22] Z. Xu, C. Shi, C. C.-C. Cheng, N. Z. Gong, and Y. Guan, ``A dynamic taint analysis tool for Android app forensics," in Proc. IEEE Secur. Privacy Workshops (SPW), May 2018, pp. 160_169.

[23] M. Sun, T. Wei, and J. C. S. Lui, ``TaintART: A practical multi-level information-_ow tracking system for Android RunTime," in Proc. ACM SIGSAC Conf. Comput. Commun. Secur., Oct. 2016, pp. 331_342.

[24] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, ``TaintDroid: An information-_ow tracking system for realtime privacy monitoring on smartphones," ACM Trans. Comput. Syst., vol. 32, no. 2, pp. 1_29, Jun. 2014.

[25] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, ``FlowDroid: Precise context, _ow, _eld, object-sensitive

and lifecycle-aware taint analysis for Android apps," ACM SIGPLAN Notices, vol. 49, no. 6, pp. 259_269, Jun. 2014.

[26] L. Li, A. Bartel, T. F. Bissyande, J. Klein, Y. Le Traon, S. Arzt, S. Rasthofer, E. Bodden, D. Octeau, and P. McDaniel, ``IccTA: Detecting inter-component privacy leaks in Android apps," in Proc. IEEE/ACM 37th IEEE Int. Conf. Softw. Eng., vol. 1, May 2015, pp. 280_291.

[27] M. I. Gordon, D. Kim, J. H. Perkins, L. Gilham, N. Nguyen, and M. C. Rinard, ``Information _owanalysis of Android applications in droidsafe," in Proc. NDSS, Feb. 2015, vol. 15, no. 201, p. 110.

[28] L. Qiu, Y. Wang, and J. Rubin, ``Analyzing the analyzers: Flow- Droid/IccTA, AmanDroid, and DroidSafe," in Proc. 27th ACM SIGSOFT Int. Symp. Softw. Test. Anal. (ISSTA), Jul. 2018, pp. 176_186.

[29] A. Kumar, V. Agarwal, S. Kumar Shandilya, A. Shalaginov, S. Upadhyay, and B. Yadav, ``PACER: Platform for Android malware classi_cation, performance evaluation and threat reporting," Future Internet, vol. 12, no. 4, p. 66, Apr. 2020.

[30] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, ``EMULATOR vs REAL PHONE: Android malware detection using machine learning," in Proc. 3rd ACM Int. Workshop Secur. PrivacyAnalytics (IWSPA), Mar. 2017, pp. 65_72.

[31] A. Nieto and R. Rios, ``Cybersecurity pro_les based on human-centric IoT devices," Hum.-Centric Comput. Inf. Sci., vol. 9, no. 1, p. 39, Dec. 2019.

[32] A. Souri and R. Hosseini, ``A state-of-the-art survey of malware detection approaches using data mining techniques," Hum.-Centric Comput. Inf. Sci., vol. 8, no. 1, p. 3, Dec. 2018.

[33] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-an, and H. Ye, ``Signi_cant permission identi_cation for Machine-Learning-Based Android malware detection," IEEE Trans. Ind. Informat., vol. 14, no. 7, pp. 3216_3225, Jul. 2018.

[34] Z. Yuan, Y. Lu, and Y. Xue, ``Droiddetector: Android malware characterization and detection using deep learning," Tsinghua Sci. Technol., vol. 21, no. 1, pp. 114_123, Feb. 2016.

[35] G. Suciu, C.-I. Istrate, R. I. R ducanu, M.-C. Diµu, O. Fratu, and A. Vulpe, ``Mobile devices forensic platform for malware detection," in Proc. Int. Symp. ICS SCADA Cyber Secur. Res, vol. 6, Sep. 2019, pp. 59_66.

[36] O. S. J. Nisha and S. M. S. Bhanu, "Detection of repackaged Android applications based on Apps Permissions," Proc. 4th IEEE Int. Conf. Recent Adv. Inf. Technol. RAIT 2018, pp. 1–8, 2018

[37] H. R. Sandeep, "Static analysis of android malware detection using deep learning," 2019 Int. Conf. Intell. Comput. Control Syst. ICCS 2019, no. Iciccs, pp. 841–845, 2019.

[38] Z. Wang, K. Li, Y. Hu, A. Fukuda, and W. Kong, "Multilevel permission extraction in android applications for malware detection," CITS 2019 - Proceeding 2019 Int. Conf. Comput. Inf. Telecommun. Syst., pp. 0–4, 2019

[39] M. Fan et al., "Android malware familial classification and representative sample selection via frequent subgraph analysis," IEEE Trans. Inf. Forensics Secur., vol. 13, no. 8, pp. 1890–1905, 2018

[40] Lessard, Jeff, and Gary Kessler. "Android forensics: Simplifying cell phone examinations." (2010).

[41] Vidas, Timothy, Chengye Zhang, and Nicolas Christin. "Toward a general collection methodology for Android devices." digital investigation 8 (2011): S14-S24.

[42] Al Barghouthy, Nedaa, and Andrew Marrington. "A comparison of forensic acquisition techniques for android devices: a case study investigation of orweb browsing sessions." 2014 6th International Conference on New Technologies, Mobility and Security (NTMS). IEEE, 2014.

[43] Jamalpur, Sainadh, et al. "Dynamic malware analysis using cuckoo sandbox." 2018 Second international conference on inventive communication and computational technologies (ICICCT). IEEE, 2018.

[44] Anish, & sk3ptre. (2021, January 18). Android Malware Timeline 2021. Sk3ptre. https://sk3ptre.github.io/Malware-Timeline-2021/

[45] win.rar GmbH. (2016). WinRAR download and support: Download. Win-Rar.com. https://www.win-rar.com/download.html?&L=0

[46] "Android Debug Bridge (Adb)." *Android Developers*, developer.android.com/tools/adb.

[47] "Ncat." Nmap.org, 2019, nmap.org/ncat/.

[48] "Autopsy." *Sleuthkit.org*, sleuthkit.org/autopsy/.

[49] "Download IntelliJ IDEA – the Leading Java and Kotlin IDE." JetBrains, www.jetbrains.com/idea/download. Accessed 1 June 2023.

[50] Wang, W., Zhao, M., Gao, Z., Xu, G., Xian, H., Li, Y., & Zhang, X. (2019). Constructing Features for Detecting Android Malicious Applications: Issues, Taxonomy and Directions. *IEEE Access*, *7*, 67602–67631. https://doi.org/10.1109/ACCESS.2019.2918139

[51] Ghosh, A., Majumder, K., & De, D. (2021). Android forensics using sleuth kit autopsy. In *Proceedings of the Sixth International Conference on Mathematics and Computing: ICMC 2020* (pp. 297-308). Springer Singapore

Appendix 1

Table 12:Applications downloaded from google playstore

package_name	title
com.google.android.apps.youtube.mango	YouTube Go
com.whatsapp	WhatsApp Messenger
com.google.android.tts	Speech Services by Google
com.winzip.android	WinZip – Zip UnZip Tool
com.zip.unzip.zipextractor.raropener.zipfile	Zip Extractor - RAR ZIP, UnZIP
com.gamma.scan2	QR & Barcode Scanner PRO
com.prisbank.app	Sberbank

Appendix 2

Table 13: All applications, package name and APK path

Malware	**APK path**	**Package name**
Rootnik: E5E22B357893BC15A50DC35B702DD5FCDFEAFC6FFEC7DAA0D313C724D72EC854.APK	/data/app/com.web.sdfile-k-J7EiTfXQlECfkTH4lwVw==/base.APK	com.web.sdfile
Rootnik: E2BDCFE5796CD377D41F3DA3838865AB062EA7AF9E1E4424B1E34EB084ABEC4A.APK	/data/app/com.br.srd-wd9DupMc6MqQxWil0h8j2A==/base.APK	com.br.srd
Rootnik: CEE6584CD2E01FAB5F075F94AF2A0CE024ED5E4F2D52E3DC39F7655C736A7232.APK	/data/app/com.oyws.pdu-T8mJZ8THM_48wsn1zaat7g==/base.APK	com.oyws.pdu
Krept banking: krep.itmtd.ywtjexf-1.APK	/data/app/krep.itmtd.ywtjexf-gsDSrpELqvys6u-CR4dEuQ==/base.APK	krep.itmtd.ywtjexf-1

Candycorn: 14d9f1a92dd984d6040cc41ed06e273e.APK	/data/app/org.merry.core-c9kJMr666FcViVSc--dp8w==/base.APK	org.merry.core
Nimaz ka waqt: 1514376339e4a0b4727c6897640c7c3e.APK	/data/app/com.tos.salattime.pakistan-t5s0eEUxxfs8oHEqqWI-DA==/base.APK	com.tos.salattime.pakistan
Xbot: 1264C25D67D41F52102573D3C528BCDDDA42129DF5052881F7E98B4A9		
Youtube:	/data/app/com.google.android.apps.youtube.mango-qBh30GLh7U0pwBoL6gGbgg==/base.APK	com.google.android.apps.youtube.mango
Whatsapp:	/data/app/com.whatsapp-eHcCvC4QGycO4kAifNvs9g==/base.APK	com.whatsapp
Sberbank:	/data/app/com.prisbank.app-/TyIM70hfj_oC2C3kDtC8ZA==/base.APK	com.prisbank.app
Barcode scanner	/data/app/com.gamma.scan2-	com.gamma.scan2

	ckccwYHzKBUdNhuZ eyqqtQ==/base.APK	
Winzip		com.winzip.androi d
Zip extractor	/data/app/com.zip.unzip .zipextractor.raropener. zipfile- Wbu8XTlljC7VoMSR wCc-FQ==/base.APK	com.zip.unzip.zipe xtractor.raropener. zipfile
Photo processing 263b0851156f7d77fb43368ce13bede1	/data/app/com.pcnts.spl icingpp- TxJngHz3tzdzRsNPC WNcsg==/base.APK	com.pcnts.splicing pp
Facebook		com.facebook.syst em
Lockkeeper 0e8805b683bc0fd8a6d49b07205f1a4b	/data/app/com.enab.loc kkeep- Y6AzdXU071I5NTKM gb0YWA==/base.APK	com.enab.lockkeep
janOscorp_20230307/f73ebc6f645926bf85 66220b14173df8.APK	/data/app/com.cosmos.s tarwarz- 2IB61gP3toiK44zR21 mgoQ==/base.APK	com.cosmos.starw arz
	/data/app/com.facebook .appmanager-	com.facebook.app manager

	AbEIncHcjzUsEhHY9bQ2wA==/base.APK	
		com.snt.rubbishcleaner