



**Abertay  
University**

# **Android Malware Analysis and Comparison with Associated Challenges**

Mini-Project investigation

**Nick Nesterenko**

CMP320: Ethical Hacking 3 coursework 2

**BSc (Hons) Ethical Hacking, Year 3**

2021/22

## Abstract or Executive Summary

---

This report details the procedure, findings, and evaluation of Android malware analysis of two recent samples. The goal was to deliver a report which involves comprehensive independent research and follows the industry standards of format and methodology. It was also attempted for the report to allow recreation of the findings as well as make the results understandable to a generally technical audience.

The methodology was chosen after completing an extensive research and collection of methods of standardized methods of Malware Analysis. Since there was no standard methodology for analyzing malware which was developed for the Android OS, the methodology was adapted on the basis of the needs of this investigation.

The two malware samples which were analyzed were variant of FluBot and Hydra android malwares. The samples were analyzed and compared to find the correlation in implementation of android malware.

After conducting the research and analysis, it was discovered that the topic of Android malware analysis was very limited and required spreading of awareness. The lack of interest from the enthusiasts also led to the lack of available tools for the analysis, however, at the end this challenge was overcome using industry leading tool called Joe Sandbox Cloud.

# +Contents

---

1	Introduction .....	1
1.1	Background .....	1
1.2	Aim .....	2
1.3	Overview of the Methodology .....	3
2	Procedure and Results .....	4
2.1	Initial Actions & Search for Advanced Tools .....	4
2.1.1	CuckooDroid v1.0 & v2.0.....	4
2.1.2	DroidBox Sandbox v4.1.1 .....	5
2.1.3	Ghidra v10.1.4.....	7
2.1.4	Finding Android Malware Samples .....	9
2.1.5	Final Set of Tools and Android Malware Samples.....	10
2.2	FluBot .....	11
2.2.1	Overview and information gathering.....	11
2.2.2	FluBot Analysis .....	12
2.3	Hydra (Android Malware) .....	19
2.3.1	Overview and information gathering.....	19
2.3.2	Hydra malware Analysis.....	20
	Discussion.....	26
2.4	The Results of Analysis and Comparison .....	26
2.5	Countermeasures.....	26
2.6	Conclusions and Associated Challenges.....	27
2.7	Future Work .....	27
	References .....	28
	Appendices.....	1
	Appendix A: FluBot Mitre Att&ck Matrix.....	1
	Appendix B: Hydra Mitre Att&ck Matrix.....	2

# 1 INTRODUCTION

## 1.1 BACKGROUND

Nowadays Smartphones became the main computing device for the majority of humanity. Today, Smartphones hold almost every piece of information about the owner, the history of movement logged by the Navigation software, personal information such as addresses and photographs which may include scans of legal identification documents, mobile banking confirmation applications and etc. Furthermore, Android operating system was chosen since Smartphones with preinstalled given OS dominate the market. According to StatCounter, 69.74% of all Smartphones as of January 2022. In addition to, unlike the main competitor (iOS), Android is a rather open operating system which makes it easier to develop malware and exploits for it. There is a huge variety of malware developed for the system.

In the early years of Android's lifecycle, specifically between years 2010-2014, there were already over 30 Major malware cases. For example, one of the most popular Android viruses is DroidKungFu (Also known as KungFu), the instances of this virus were first detected in 2011 and the virus still works to this day and even gets updated for the latest versions of Android. The situation worsens as the discovery of new malware developed for Android Operating System grows every year to this day. According to G DATA Software (2019) the number of newly discovered Android malware rose from 3.25 million instances to 4.18 million from 2016 to 2019 respectively, (Figure 1):

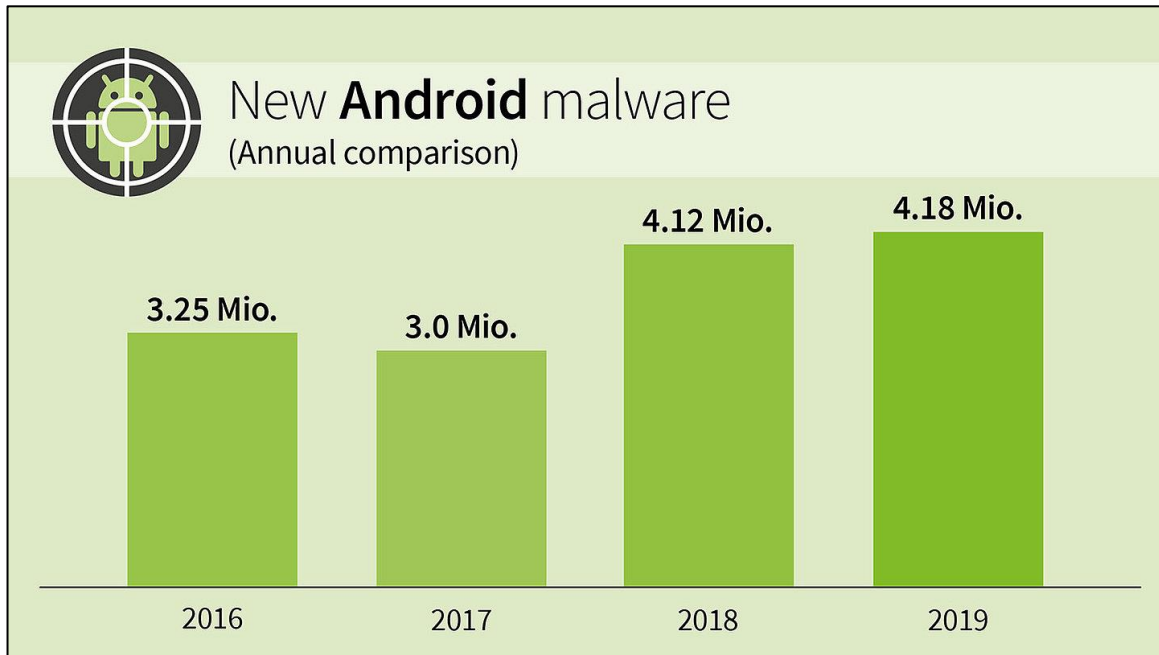


Figure 1: G DATA Software New Android malware (Annual comparison)

Furthermore, Android OS was the third largest target for malware developers following Windows and Browser malware as by Q1 2020 ranked by the percentage of malware detections according to Survey results published by Joseph Johnson at Statista (2020), (Figure 2):

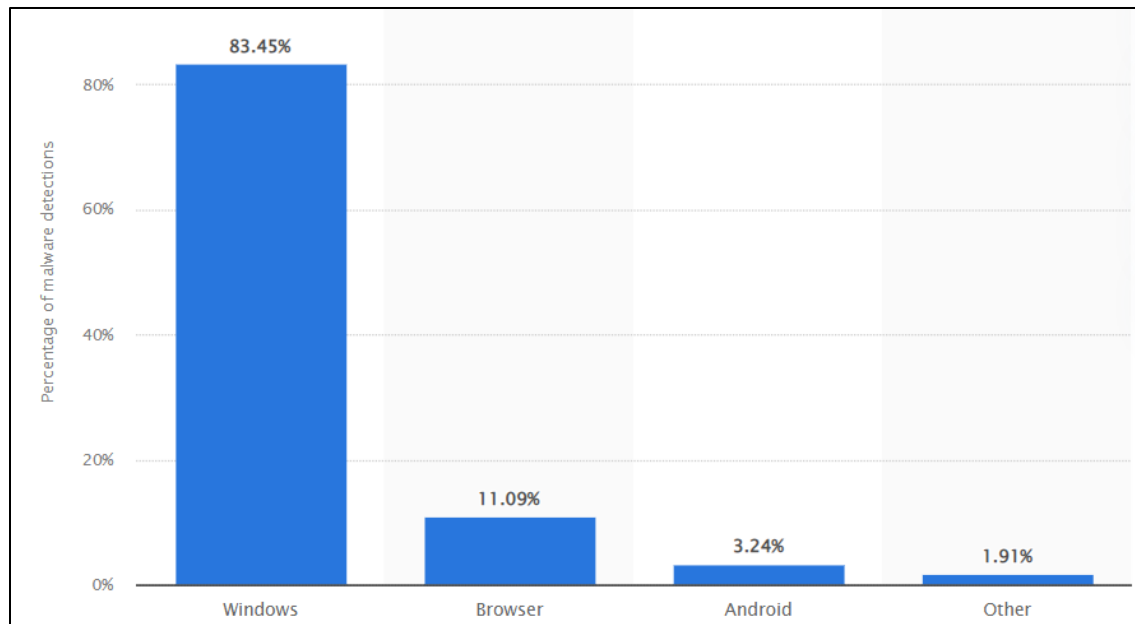


Figure 2: Statista Operating systems most affected by malware as of 1st quarter 2020

Considering the said above, the importance of understanding Android Malware can be classified as critical. In addition to that, the amounts of research and tools for analysis of Android malware is relatively low due to the fact that Android operating system is rather new as it was released in 2008 in contrast to Windows OS which was released in 1985. This collection of facts led to the decision to conduct an investigation into the topic by performing an analysis of several successors of fundamental malware which was discovered between 2011 and 2012 and now are adapted to modern technologies.

## 1.2 AIM

The aim of this project is to analyze several Android malware samples and find a correlation between the threats and behaviors of those. The aims of this project also include presenting the result of the analysis and evaluating the findings in accordance with industry methodology standards and evaluating challenges related to Android Malware Analysis.

During the analysis stage, it was required to identify the following:

1. Available tools for Android Malware Analysis.
2. The background of the chosen malware sample.
3. Analysis of the sample's suspicious functions/activities, services, resources and requested permissions during the phase of static analysis.
4. Analysis of the network traffic generated traffic by the malware sample during the phase of dynamic analysis.
5. Analysis of created/generated files by the malware samples during the phase of dynamic analysis.
6. Analysis of APK behavior during the phase of dynamic analysis.
7. Comparison of the malware samples and evaluation of the significance of the treats.
8. Evaluation of the faced challenges during Android Malware Analysis.

### 1.3 OVERVIEW OF THE METHODOLOGY

---

There are several standard methodologies for malware analysis, most of which involve the two critical steps of the investigation which include:

1. Static analysis.
2. Dynamic analysis.

In order to make the methodology more complete, additional research was conducted for finding relevant methodology practices. As per the time of conduction of this investigation, there were no standardized methodologies for analysis of malware in Android OS, methodologies of malware detection techniques were found. In the relation to the absence of the methodology of analysis, it was decided to combine the analysis techniques of Linux based systems and the Android malware detection systems.

The following methodology was used during the current Android Malware analysis and comparison investigation was inspired by ***MMALE—A Methodology for Malware Analysis in Linux Environments (2021)*** research paper which was published on ResearchGate by Mohino, José Javier & Bermejo, Javier & Higuera, Juan-Ramón & Montalvo, Juan Antonio & Rubio, Manuel & Herraiz, researchers of The International University of La Rioja, Spain. In addition to that, the ***Analysis of Android Malware Detection Techniques: A Systematic Review (2019)*** research paper by Moses Ashawa and Sarah Morris, researchers of Cranfield University, Defense Academy of the United Kingdom:

1. **Initial actions** – gathering of the evidence that all changes and modifications done by the malware are possible to track as well as making sure that there is a possibility to return easily to clean state of a victim machine and guarantee that there is no outbound network traffic generated.
2. **Information gathering** – one of the stages of the malware analysis which involves additional research about the malware sample in order to identify the proposed functionality and implementation which will then be used as a directional vector of the scope and analysis.
3. **Static analysis** – the first stage of malware analysis which involves the evaluation of the static malware code without the malicious application running, this step can be performed in multiple ways including core reverse engineering. However, for the purposes of this investigation it was decided to utilize an automated tool which would return all the notable and necessary data.
4. **Dynamic analysis** – the main phase of the malware analysis where the malware is executed in a controlled environment such as Malware Analysis Sandbox, which ensures the safety of the host machine, in order to determine the overall behavior of the malicious application and other aspects such as the generated internet traffic.

The utilized tools were described during the phase of Initial Actions in Procedure and Results section.

## 2 PROCEDURE AND RESULTS

### 2.1 INITIAL ACTIONS & SEARCH FOR ADVANCED TOOLS

---

This section described the information gathering process and setup of tools required for analysis of Android malware and ensuring the safety measures to the local host machine as well as the process of obtaining access to the infected samples through several repositories. The detailed explanation of used tools was necessary due to the fact the limited availability of such.

#### 2.1.1 CuckooDroid v1.0 & v2.0

The initial stage of investigation included the search of all the necessary tools to perform the malware analysis. At first, it was decided to search for the malware analysis tools which would allow for the malicious APK to be run on the local machine by emulating an Android device using the Android SDK on the Host machine. The first attempted tool for setup was the **CuckooDroid** which is an extension of **Cuckoo Sandbox** which was designed to work with Android malware samples, one of the most popular opensource Desktop malware analysis tools.

There were several attempts to install CuckooDroid tool which unfortunately were unsuccessful due to a set of issues. Three different versions of Ubuntu Linux were tested to get CuckooDroid in the operation state which were:

1. Ubuntu 20.04.
2. Ubuntu 18.04.
3. Ubuntu 14.04.

Multiple ways of installation were followed during the attempts of installation which led to different behaviors of the tool. However, the closest iteration was using a freshly installed Ubuntu 18.04 with Virtual Machine which was run under VMware 16 using a guide posted on **Hydrasky** website in 2017 by a user named Cloudi. It was a more detailed guide which followed the instructions of the original instruction manual on Cuckoo Sandbox and CuckooDroid installation and at first glance the tool managed to run however it was not possible to establish connection with the Guest system in the form of emulated Android device which used Android API 16 as suggested by the official instruction manual.

There were several suggested reasons as of why the tool refused to work as intended. CuckooDroid required multiple dependencies for the operation and even though the guide provided by Cloudi had the software versions suggested, the compatibility may have changed and due to the large set of required additional tools it could have led to some being changed from the previously operational/compatible state. Another reason of the unsuccessful installation would be new iterations of Android SDK which was greatly updated since the initial release of CuckooDroid 2.0 which happened in 2017, whereas the SDK is frequently updated and the platform tools for the emulation are always updated automatically upon installation of the SDK so there could be a compatibility issue in that. Lastly, Cuckoo Sandbox (latest version 2.0.7) itself is currently left unmaintained as suggested by the developers, left alone CuckooDroid which mainly utilized the Sandbox version 1.2 which may have also led to the compatibility issues, (Figure 3):

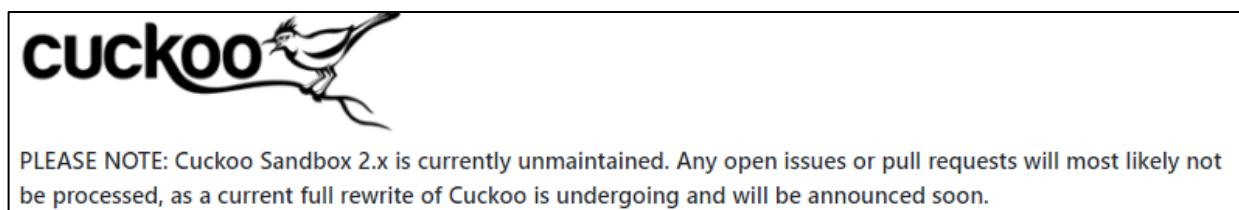


Figure 3: Cuckoo Sandbox Unmaintained

### 2.1.2 DroidBox Sandbox v4.1.1

The next Android malware analysis Sandbox tool which was attempted was **DroidBox**, another sandbox which allows for making the emulator isolated and monitored. This tool was dropped due to different reasons. Considering the advancement of reporting tools like Cuckoo Sandbox for Desktop malware and Virus total which provide a comprehensive and interactive report which allows for better understanding of by an unexperienced user, DroidBox on the other hand provides limited output by modern standards. In an overview, DroidBox provides simple “yes/no” type of analysis as in for example “The SMS sending was not done by the provided APK”. Such reports may have been useful back in 2015, this year was referred as DroidBox tool was discovered during the research process and examination of one of the only dedicated books related to the topic named **Android Malware and Analysis** (2015) by Ken Dunham, Shane Hartman, Jose Andre Morales, Manu Quintans, and Tim Strazzere. The book had a set of tools mentioned which were purposely made for Android malware analysis, however most of the tools and URLs were no longer operational except for DroidBox. The output of the tool was presented as followed, (Figure 4):

 A screenshot of a terminal window with a dark background and light-colored text. The output shows the process of installing and running an application. At the top, there is a large ASCII art logo of a revolver. Below it, the text reads: "Waiting for the device...", "Installing the application /home/revolver/Downloads/com.parental.control.v4.apk.", "...", "Running the component com.parental.control.v4/com.connect.Dendroid...", "Starting the activity com.connect.Dendroid...", "Application started", "Analyzing the application during infinite time seconds...", and finally a large JSON object containing various analysis results like apkName, permissions, network activity, and hashes.

Figure 4: DroidBox sample output

#### 2.1.2.1 Joe Sandbox Cloud Basic

After completing extensive research, an online malware sandbox called **Joe Sandbox Cloud** by Joe Security was discovered. Joe Sandbox Cloud is a deep malware analysis engine which is often referred as “industry leading” tool in the sphere of malware analysis of any sort despite being less widely known in



comparison to online-based sandboxes and malware reporting tools such as ANY.RUN or VirusTotal. Unlike most of the available online malware sandboxes, Joe Security provides a framework which allows users to analyze malware which is built for different operating systems including Android and many others which indeed require attention. The overall layout of the user interface is standard to such type of tools (alike Cuckoo Sandbox, ANY.RUN, etc.), while having the most advantageous feature which was the ability to select operating system programmatically, (Figure 5):

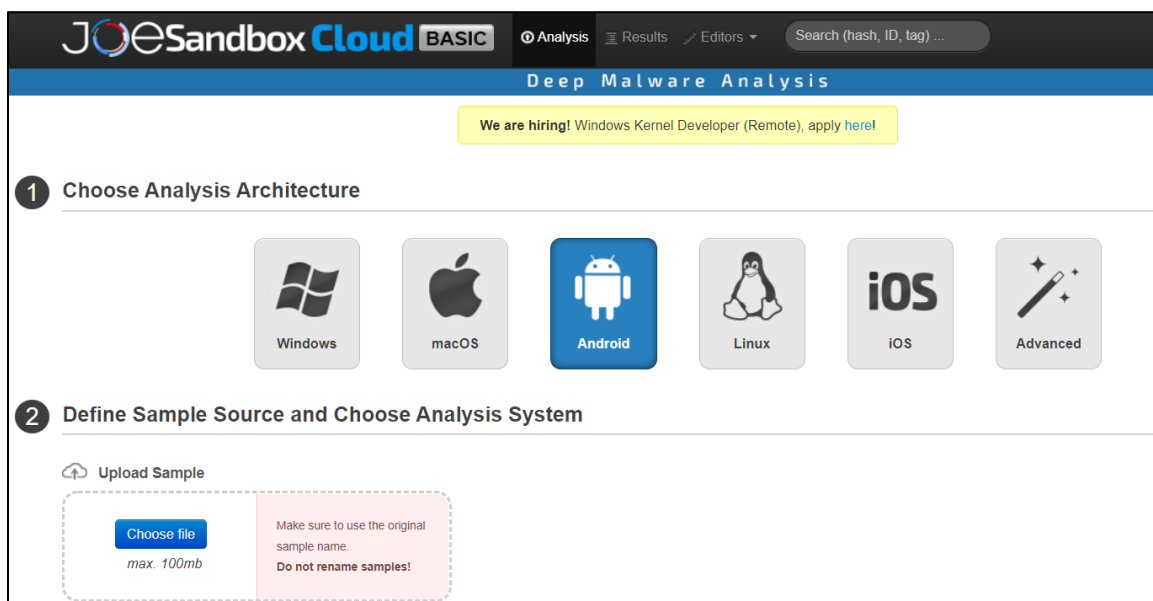


Figure 5: Joe Sandbox Cloud interface

In addition to the required functionality for the purpose of this malware analysis investigation, Joe Sandbox Cloud was considered as a superior tool to other solutions even for Desktop malware analysis by the competent parties. For example, the SC media magazine (2017) described Joe Sandbox as a “the most complete reversing and malware analysis tool” SC media “have ever seen” and added it to the “SC Lab Approved tool set”. This is due to the fact that Joe Sandbox provided a more complete and detailed malware analysis reports than its competitors while still being friendly for a less technical user.

The advanced functionality may suggest that the tool could have been paid and the most complex features such as interactive remote access to the Virtual Machines were indeed guarded under the paywall which was evaluated on the case-by-case basis. According to the community on Reddit, the prices range from \$2000-\$12000 for allowance of 250 advanced analysis per month. However, Joe Security offered the Cloud Basic plan for its malware sandbox which included all the necessary information for the purposes of the current malware analysis investigation including access to the reports conducted by the premium users. Even Basic plan was evaluated on case-by-case basis during the registration applications.

Considering the above, it was decided to use Joe Sandbox Cloud Basic as the main analysis and reversing tool for understanding Android Malware as it satisfied the criteria for safe and controlled environments for execution of malicious code in accordance with the proposed methodology of this investigation mentioned in Section 1.3. The tool is available at the following URL:

<https://www.joesandbox.com/>

### 2.1.3 Ghidra v10.1.4

Ghidra was another essential tool for any kind of malware analysis. This tool is a free and open-source reverse engineering tool implemented by the National Security Agency (NSA) of the United States. Ghidra was released to the public on 5<sup>th</sup> of March 2019 at the RSA Conference in San Francisco. As mentioned above, Ghidra is a reverse engineering tool meaning that it is used to disassemble and decompile existing applications. For the purposes of this malware analysis investigation, Ghidra was used for disassembling infected .APK files which stand for Android Package and is an alternative to .exe files in Windows operating systems.

Using Ghidra is relatively straightforward, specifically considering the complexity of the setup procedures for other tools used for Android Malware Analysis; however, it does require advanced knowledge of Java/Kotlin programming languages used for development of software dedicated for Android OS. Ghidra is considered to be superior to “any other Reverse Engineering tools with the only exception of IDA”, according to security researcher, Joxean Koret. And IDA Pro is a premium RE tool which was not ideal for current investigation, so it was decided to use Ghidra for Reverse Engineering during static analysis, (Figure 6):

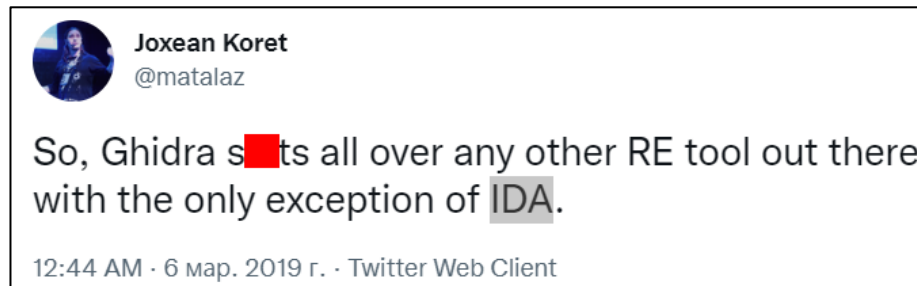


Figure 6: Joxean Koret tweet on Ghidra

In order to install Ghidra v10.1.4 on Ubuntu 20.04 VMware Virtual Machine the following commands were executed in the terminal, (Figure 7):

```
1 sudo apt install openjdk-11-jdk
2 cd ~Downloads/
3 wget https://github.com/NationalSecurityAgency/ghidra/releases/download/Ghidra_10.1.4_build/-ghidra_10.1.4_PUBLIC_20220519.zip
4 unzip ghidra_10.1.4_PUBLIC_20220519.zip
5 cd ghidra_10.1.4_PUBLIC_
6 ./ghidraRun
```

Figure 7: Ghidra install Script

After executing the script, Ghidra installed all the necessary dependencies automatically and proceeded to the setup screen, (Figure 8):



Figure 8: Ghidra launching

After the reverse engineering tool was opened, it was required to create a new project by accessing the navigation bar **File -> New Project...** in order to import the desired .APK file. The project was configured as **Non-Shared Project** and the desired directory was then set as well as suitable project name, (Figure 9):

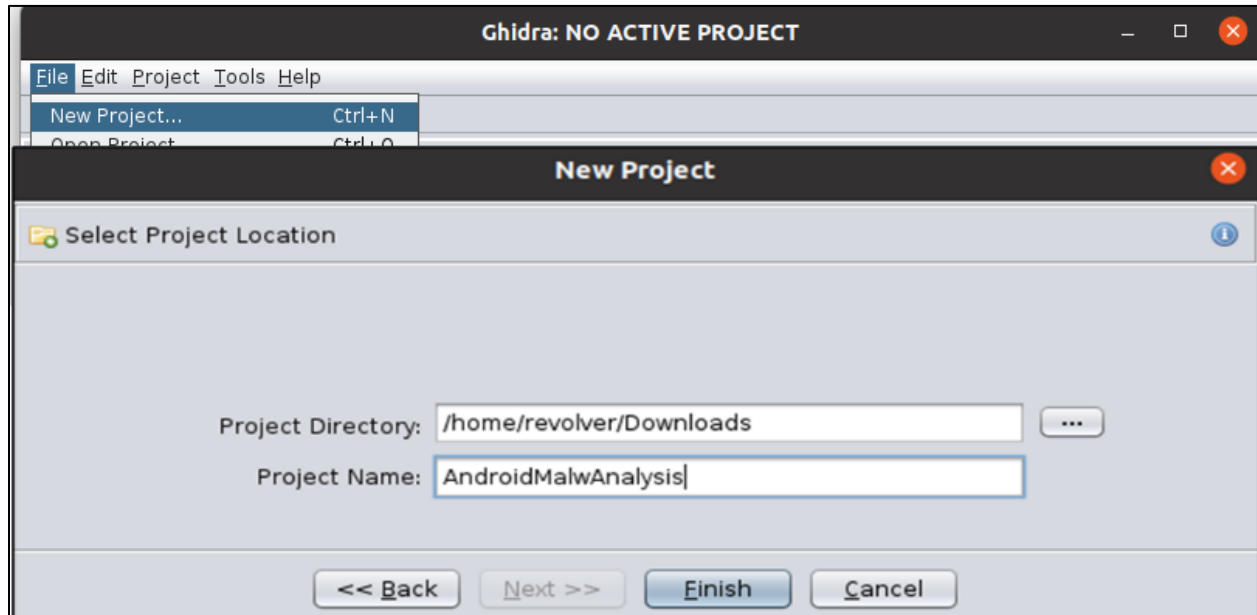


Figure 9: Ghidra (Creating new Project)

The next step was to import the infected .APK file, this was done using the navigation bar after creating the new project **File -> Import File...** after that the file was selected and the import mode must be selected as **Batch**, (Figure 10):

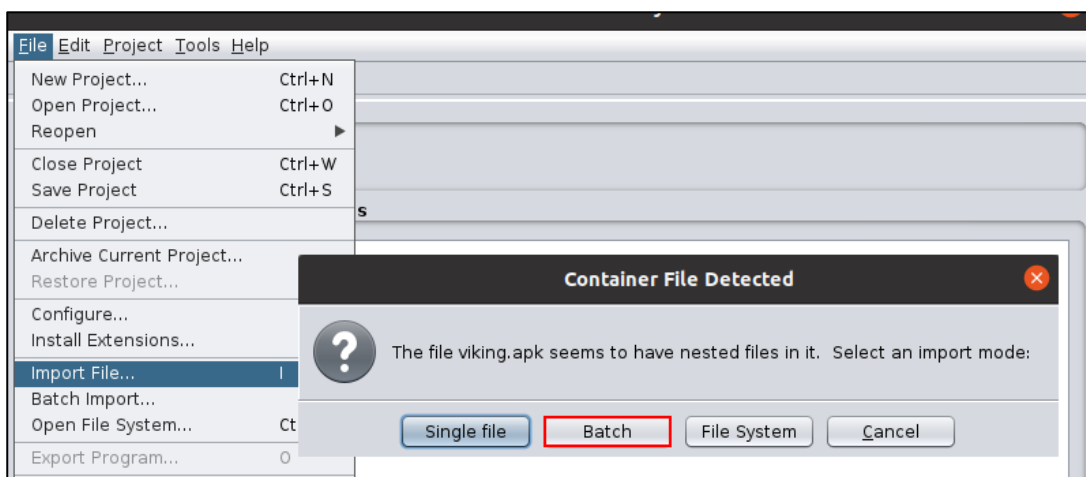


Figure 10: Importing infected .APK files into Ghidra

The repositories where the Android malware samples were taken from were described in the following section.

### 2.1.4 Finding Android Malware Samples

Several repositories were used to find Android malware samples. First of all, some of the previous Joe Sandbox Cloud reports allow the users with Basic plan to retrieve the samples which were analyzed. Secondly, a tool called **theZoo** was also used, however that repository had a very limited number of samples. Ubuntu 20.04 Virtual machine was used to setup theZoo using the following commands, (Figure 11):

```
git clone https://www.github.com/ytisf/theZoo
cd theZoo
pip install --user -r requirements.txt
python theZoo.py
```

Figure 11: Installing theZoo

After installing theZoo, the malware repository was queried for malware which was developed for the Android platform using **search android** command, (Figure 12):

```
| 100 | trojan      | apk          | arm      | android | Dendroid
| 162 | trojan      | java         | arm      | android | Mazar
| 163 | ransomware  | java         | arm      | android | android_locker
| 175 | trojan      | java         | arm      | android | Skygofree
| 192 | trojan      | multi        | arm      | android | Dendroid
| 306 | apt         | Java         | arm      | android | Pegasus
| 322 | rat         | VB           | arm      | android | SpyNote
| 326 | apt         | bin          | Android  | Android | Transparent Tri
be
| 331 | botnet      | java, c, gradle | android  | android | Cerebrus
+-----+-----+-----+-----+-----+
[+] Total records found: 11
```

Figure 12: theZoo (Finding Android malware samples)

In addition to the tools for accessing malware repositories, two additional websites were used which also held a substantial number of malwares developed for Android:

1. vx-underground, URL: <https://samples.vx-underground.org/samples/>
2. contagiominidump, URL: <http://contagiominidump.blogspot.com/>

### 2.1.5 Final Set of Tools and Android Malware Samples

After completing the extensive research of tools for Android malware analysis, the final list of tools was put together. The following set of tools was then used to analyze two malware samples:

- Virtualization software:
  - VMware Workstation 16.
- Virtual Machine Operating System:
  - Ubuntu 20.04.
- Static malware analysis tools:
  - Joe Sandbox Cloud: Basic – automated static analysis functionality.
  - Ghidra v10.1.4 – reverse engineering tool.
  - Dex2Jar v2.1 – tool to translate .dex files into .jar Java files.
- Dynamic malware analysis tools:
  - Joe Sandbox Cloud: Basic – automated dynamic analysis functionality.
  - Android Studio & SDK – used for Android emulation and profiling (Use at own risk).
- Android malware repositories:
  - Joe Sandbox Cloud: Basic – allowed the retrieval of several previously analyzed malware samples.
  - theZoo v0.6.0 – Linux based tool for accessing malware repository.
  - vx-underground – powerful malware repository with multiple instances of each malware, a collection of malware samples taken from other repositories such as VirusShare.
  - Contagiominiidump – Android malware mini repository.

For the purposes of this Android malware analysis investigation, it was decided to undertake an analysis and comparison of two malware samples. The proposed strategy was to take one sample as the control example of malware, whereas the second was used to compare and contrast the difference between malware implementations and techniques used. The samples were part of the following malware families:

- FluBot (Control Sample).
- Hydra (Used for comparison).

## 2.2 FLUBOT

### 2.2.1 Overview and information gathering

FluBot was a sophisticated piece of Android malware which was discovered in the early 2020. The malware was distributed as an SMS scam which utilized two different techniques of social engineering, either as a link for downloading of a tracking software for the popular delivery services such as DHL and FedEx or as a voice mail application. The malware would spread the using the information accessed from the already infected victims' contact lists. For the purposes of the current investigation, the voice mail version was taken for analysis. An example of the phishing SMS messages with the malicious link looked as follows, (Figure 13):

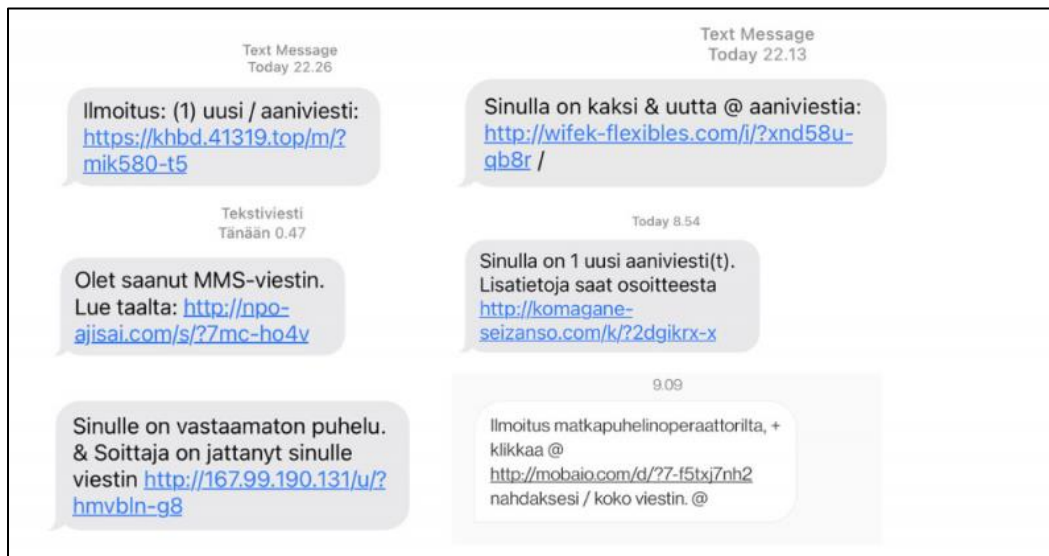


Figure 13: FluBot scam phishing SMS example (Finland)

Such links would prompt the user to download the infected application in a form of .APK file, (Figure 14):

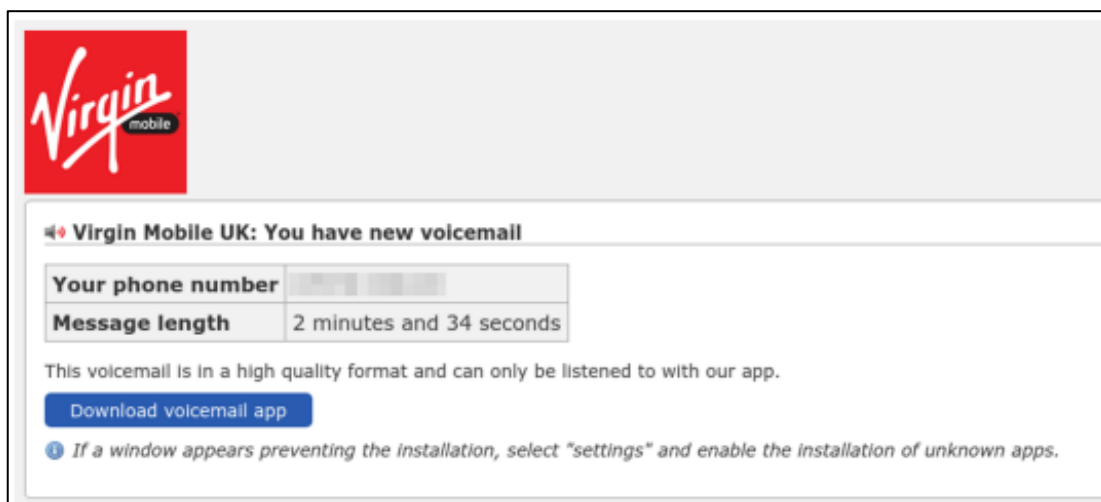


Figure 14: FluBot scam website example (Virgin media)

FluBot was a piece of malware which abused the Android's Accessibility Service, and even though it was not the first one to utilize such method of attack, FluBot gained the access to a list of malicious functionalities on the victims' mobile devices:

- Full remote control of the Device, similarly to the AndroRAT exploit which can be found in the Metasploit Framework which granted the attacked with full reverse shell of the device. Considering that Android was based on Linux, such unauthorized access can cause critical damage to users' personal information.
- Overlay functionality targeted on the mobile banking application installed on the users Android device. This is especially critical as nowadays most of the banking operations are controlled and accessed using mobile banking apps.
- Sending, intercepting, and hiding the SMS messages and notifications. Many malwares which target Android systems have such malicious functionality as it gives an opportunity to intercept two-factor authentication messages.
- Key-logging functionality which could potentially be used to steal victim's passwords and other sensitive information such as private text-based conversations.

### 2.2.2 FluBot Analysis

The next step was to prove that the malware has the functionality mentioned in the previous section by conducting an analysis. In addition to that, it was necessary to prove the reliability of Joe Sandbox Cloud automated analysis by comparing the findings against the reverse engineered code from Ghidra.

The malware sample with MD5 hash **75c0eb2c78ff31534a588ec47088b622** which was taken from the vx-underground malware repository was passed to the Joe Sandbox Cloud by dragging the file into the file submission area. After around 25 minutes, the automated malware analysis report was generated, (Figure 15):



Figure 15: FluBot analysis report was generated by Joe Sandbox Cloud

The vulnerabilities and concerns were presented by Joe Sandbox Cloud report in the form of signatures with a graphical representation of the results in the form of color (the redder the more critical). The

sandbox also provided a graphical representation of the classification of the malware using the following chart, using the located signatures of the vulnerabilities and suspicious activities which were present in the infected application file, (Figure 16):

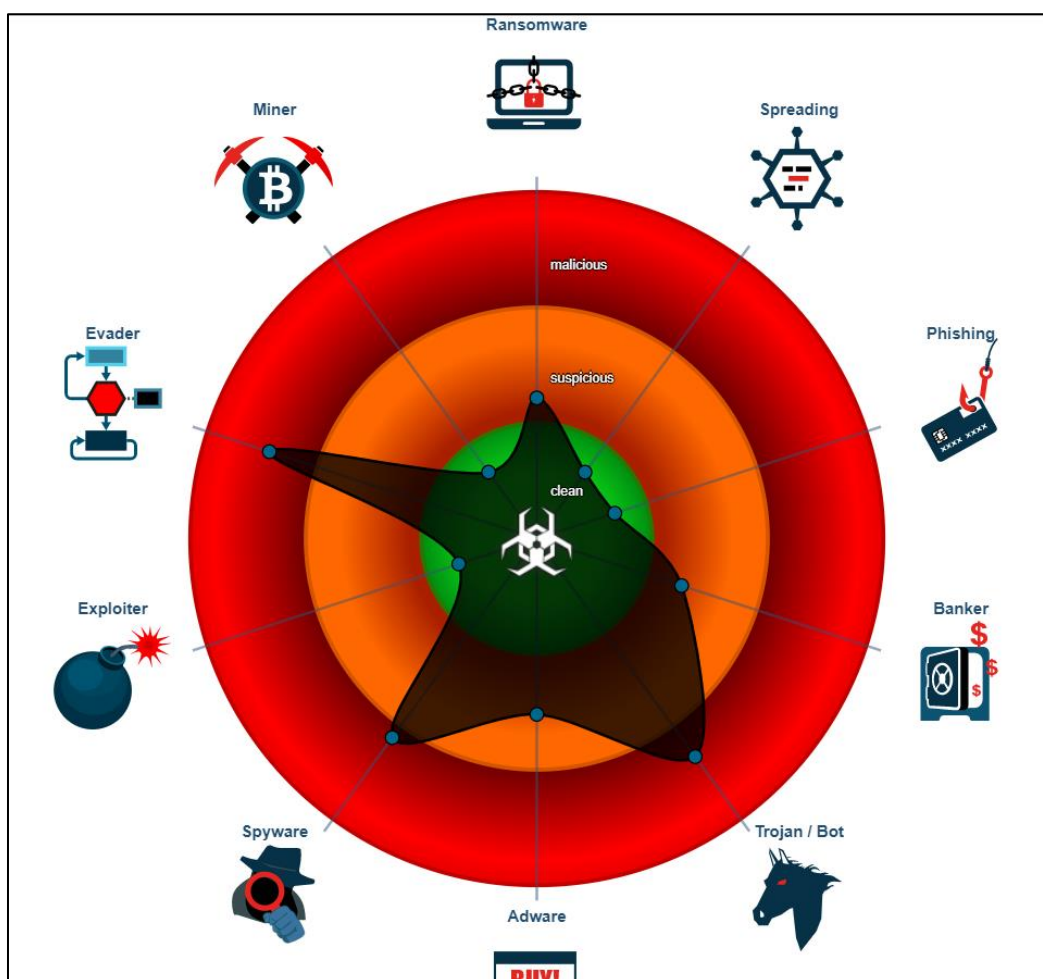


Figure 16: FluBot malware classification chart

As mentioned previously, Joe Sandbox Cloud communicates the concern using signatures which are summarized using the generated Mitre Att&ck Matrix, a table of concerns related to the malware sample. Please refer to the Appendix A for the complete Mitre Att&ck Matrix table.

There were a large number of the located signatures found by the Sandbox. For the purposes of this malware analysis investigation, only the most important signatures were discussed, however, the complete report can be accessed using the following URL:

<https://www.joesandbox.com/analysis/450959/0/html>

In order to prove the reliability of the Joe Sandbox Cloud several examples of found concerns were compared to the ones found in the process of reverse engineering with Ghidra. It was necessary in order to view the source code in Ghidra properly. By looking through the report generated by Joe Sandbox, it was discovered that the .APK file was using software called ApkProtector meaning that the .APK file was



translated into .DEX (Dalvik Executable) instead of Java files in order to conceal the malicious code for the purpose of prevention from static and dynamic analysis of the application, (Figure 17):

Sample is protected by ApkProtector	
Source: <a href="#">Android App</a>	File dump: /data/user/0/com.xunmeng.pinduoduo/app_apkprotector_dex/xCSswjIO.king
Source: <a href="#">Lcom/disneyplus/netflix/q;-&gt;a(I)Ljava/lang/String;</a>	Method string: apkprotector_dex

Figure 17: ApkProtector .dex protection detected

This challenge was solved using an opensource tool called Dex2Jar v2.1 using the following command:

**dex2jar Voicemail39.apk -> ./Voicemail39-dex2jar.jar**

After the transformation of the .APK file, it was possible to perform the import of files into Ghidra, as was mentioned in Section 2.1.3. One of the concerns related to the FluBot infected file was that the application is able to create photographs without the victim's knowledge:

Queries camera information	
Source: <a href="#">com.usabilla.sdk.ubform.screenshot.c.j.a</a> ;->t:66	API Call: android.hardware.Camera.getNumberOfCameras
Source: <a href="#">com.usabilla.sdk.ubform.screenshot.c.j.a</a> ;->t:68	API Call: android.hardware.Camera.getCameraInfo
Source: <a href="#">com.usabilla.sdk.ubform.screenshot.c.j.a</a> ;->w:91	API Call: android.hardware.Camera.open

Figure 18: FluBot querying camera information

This was letter confirmed using the Ghidra reverse engineering tool as well as other java files which proved the reliability of Joe Sandbox Cloud automated analysis, it was then decided to continue to refer to the analysis results gained from the Joe Sandbox report as the representation of the findings were more graphical and understandable:

```

7 void <clinit>_void(void)
8
9 {
10     a[] ppaVar1;
11     a objectRef;
12     a objectRef_00;
13     a objectRef_01;
14
15     objectRef = new a("CAMERA",0,"camera");
16     a.CAMERA = objectRef;
17     objectRef_00 = new a("GALLERY",1,"library");
18     a.GALLERY = objectRef_00;
19     objectRef_01 = new a("DEFAULT",2,"default");
20     a.DEFAULT = objectRef_01;
21     ppaVar1 = new a[3];
22     ppaVar1[0] = objectRef;
23     ppaVar1[1] = objectRef_00;
24     ppaVar1[2] = objectRef_01;
25     a.$VALUES = ppaVar1;
26     return;

```

Figure 19: Ghidra confirming Joe Sandbox findings

The next notable signature was related to the FluBot itself. It was discovered that the remote access functionality of FluBot was incoming from a different file name, which suggested that the infected voice mail application deployed another .APK file which contained the main FluBot malicious code proving that the analyzed Android malware sample was of Trojan type, (Figure 20):

Detected FluBot	
Source: <a href="https://lcom.xunmeng.pinduoduo/pa5234fd9;-&gt;k(Landroid/view/accessibility/AccessibilityNo delInfo;)Z">Lcom/xunmeng/pinduoduo/pa5234fd9;-&gt;k(Landroid/view/accessibility/AccessibilityNo delInfo;)Z</a>	Method: Various indicators of FluBot

Figure 20: FluBot detected in another deployed (dropped) .APK file

It was then decided to examine the Created / Dropped Files section of the Joe Sandbox report, where three created files were discovered including the ApkProtector.apk mentioned earlier, createdfiles0.csv.part, which was not classified as malicious, but it could be an integral part of the malware, and finally the base.apk which was the main FluBot Android package application, (Figure 21):

Created / dropped Files	
/data/app/com.xunmeng.pinduoduo-Y2SeKImAcbPmCU0Gtk7g6g==/base.apk	
File Type:	troff or preprocessor input, ASCII text, with very long lines
Category:	dropped
Size (bytes):	800
Entropy (8bit):	3.328394258288145

Figure 21: FluBot .APK file dropped by the malware sample

The next step was to investigate the static information about the infected .APK file. Several suspicious receivers and services were located by the Joe Sandbox, (Figure 22):

Receivers	
<ul style="list-style-type: none"> <li>com.xunmeng.pinduoduo.pb601e383</li> <li>com.xunmeng.pinduoduo.pec08273e</li> </ul>	<ul style="list-style-type: none"> <li>Intent: android.provider.Telephony.WAP_PUSH_DELIVER</li> <li>Intent: android.provider.Telephony.SMS_DELIVER</li> </ul>
Services	
<ul style="list-style-type: none"> <li>com.xunmeng.pinduoduo.p1834e5c7</li> <li>com.xunmeng.pinduoduo.p2bc32733</li> <li>com.xunmeng.pinduoduo.p5ac58bb5</li> <li>com.xunmeng.pinduoduo.pa5234fd9</li> <li>com.xunmeng.pinduoduo.pd1c4396e</li> </ul>	<ul style="list-style-type: none"> <li>Intent: android.service.notification.NotificationListenerService (Priority 0)</li> <li>Intent: android.intent.action.RESPOND_VIA_MESSAGE (Priority 0)</li> <li>Intent: android.accessibilityservice.AccessibilityService (Priority 0)</li> </ul>

Figure 22: FluBot static information (Receiver and Services)

The receivers and services that begun with the **com.xunmeng.pinduoduo** application were related to the deployed .APK which contained main FluBot malicious code. In addition to that, there were other predefined services such as the ability to check the delivery of SMS messages as well as the service of responding via SMS messages which proved that the infected Android application was able to send SMS messages in the background without the need for opening dedicated messaging application. It was also located the predefinition of Accessibility Services which FluBot utilized in order to obtain control over the currently running activities and applications, the main service which FluBot was designed to exploit.

The application also had an extensive list of requested permissions. Considering that the application was targeting Android API 28 which did not have mandatory asking for permissions after installation of side loaded applications, (Figure 23):

General	
Label:	Voicemail
Minimum SDK required:	24
Target SDK required:	28

Figure 23: FluBot sample API version

This meant that the following list of permissions was granted automatically and the list essentially portrayed the list of functionalities that the infected application was able to access without any further interruptions, (Figure 24):

Permission Requested
<ul style="list-style-type: none"> <li>• android.permission.ACCESS_NETWORK_STATE</li> <li>• android.permission.BIND_ACCESSIBILITY_SERVICE</li> <li>• android.permission.BIND_NOTIFICATION_LISTENER_SERVICE</li> <li>• android.permission.CALL_PHONE</li> <li>• android.permission.FOREGROUND_SERVICE</li> <li>• android.permission.INTERNET</li> <li>• android.permission.KILL_BACKGROUND_PROCESSES</li> <li>• android.permission.QUERY_ALL_PACKAGES</li> <li>• android.permission.READ_CONTACTS</li> <li>• android.permission.READ_PHONE_STATE</li> <li>• android.permission.READ_SMS</li> <li>• android.permission.RECEIVE_SMS</li> <li>• android.permission.REQUEST_DELETE_PACKAGES</li> <li>• android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS</li> <li>• android.permission.SEND_RESPOND_VIA_MESSAGE</li> <li>• android.permission.SEND_SMS</li> <li>• android.permission.VIBRATE</li> <li>• android.permission.WAKE_LOCK</li> <li>• android.permission.WRITE_SMS</li> </ul>

Figure 24: Requested permissions by FluBot

After examining the list of requested permissions, it was concluded that the infected Android application had several suspicious permissions predefined. To begin with, the access to the network state, meaning that the application can monitor if the user is connected to the internet and if the network is WIFI, the application can gain information about the network. Next, the application has the ability to bind the accessibility service, the functionality which FluBot exploits for the remote access. Furthermore, binding the notification listener service, this suggested that the infected application had the ability to set listeners for any type of notifications, which is even more severe than just listening to the SMS notifications, as stated during the phase of information gathering. The next item on the list was the permission to perform calls from within the application, although this could be a part of the “voice mailing” functionality, considering the rest of the requested permissions, this could be interpreted as making phone calls without the victim’s knowledge. Foreground service permission allows the application to keep the icon of the app inside of the status bar which notifies the victim that the application is currently running, which may create an impression of trust. Furthermore, the internet permission usually does not impose risk, however, in current case this suggested that the application is able to offload the information of the victim to external service as well as receive malicious connections such as remote access. The next permission was especially interesting, the infected Android application had the ability to kill background processes meaning the application could potentially stop the background processes of installed anti-virus software and other security services. Next, the application was able to query all packages, meaning that the application can look through the list of installed applications and possibly identify installed mobile banking applications. Moreover, the application had the ability to read the contacts list, which would suggest the application is using for gaining new phone numbers to send scam SMS messages as was proposed in the phase of information gathering. The application had the ability to read the phone state, meaning that it was able to read information such as the current cellular network information and the status of any ongoing calls as well. Another two of the requested permissions were the reading and sending SMS, another red flag for a voice mailing application. A critical requested permission was located, the infected application had the ability to delete packages, meaning the malware sample could delete other installed application on victim’s device which could potentially mean that the malware could delete some application and replace it with an infected malicious alternative without user’s knowledge. The infected application also had the ability to send a request to other applications to handle the respond-via-message action during incoming calls. Furthermore, the malware sample was able to ignore battery optimization, meaning that the background malware processes would not be terminated if the battery optimization mode is on. Finally, the infected application had the ability to keep the device from locking the phone (AKA dimming the screen) as well as send vibrations and write SMS messages.

Just the list of the requested permissions can suggest the overall suspiciousness of the Android application. However, to sum the analysis up, it was decided to analyze some sections of the dynamic behavior of the malware sample. First of all, Joe Sandbox was able to report the executed permissions and methods, meaning that it was possible to view the functionality which was accessed during the dynamic analysis. It was discovered that FluBot, in fact, did not execute any of the above stated permissions. This suggested that the malware sample detected that it was executed inside of the Virtual Machine, so it was decided to revisit the list of signatures generated by the sandbox in order to become sure that the malware sample had the ability to detect its environment. After inspecting the list of signatures, it was discovered that the application had multiple implementations of techniques which provide the protection of the malware against debugging and analysis, (Figure 25):

<b>Malware Analysis System Evasion:</b>
Accesses android OS build fields
Queries several sensitive phone informations
Queries the unique operating system id (ANDROID_ID)
<b>Anti Debugging:</b>
Checks if app is currently debugged

Figure 25: FluBot analysis evasion techniques

It was also discovered that the application tried to query several Google servers which were located in the United States, this could also be a part of environmental checks, (Figure 26):













IP	Domain	Country	Flag
173.194.69.188 	unknown	United States	
142.250.186.163 	unknown	United States	
8.8.4.4 	unknown	United States	
142.250.203.110 	unknown	United States	
142.250.186.42 	unknown	United States	
216.58.212.170 	unknown	United States	

Figure 26: Queried IP addresses by FluBot

Finally, it was discovered that the main methods of the malware were not executed during the analysis, proving the existence of the anti-emulation techniques, (Figure 27):

<b>13 Non-Executed Methods</b>
Method: com.xunmeng.pinduoduo.pa5234fd9->onAccessibilityEvent(Landroid/view/accessibility/AccessibilityEvent;
Method: com.xunmeng.pinduoduo.pa5234fd9->k(Landroid/view/accessibility/AccessibilityNodeInfo;) Relevance: 19.6
Method: com.xunmeng.pinduoduo.pa5234fd9->h(Ljava/lang/String;Landroid/view/accessibility/AccessibilityNodeInfo

Figure 27: FluBot's main remote access methods were not executed

## 2.3 HYDRA (ANDROID MALWARE)

### 2.3.1 Overview and information gathering

Hydra is a family of Android malware, which is considered even more advanced in contrast with FluBot. The malware was targeting Android mobile banking customers since 2019 and was still life during the time of this Android malware analysis investigation. Just like FluBot, Hydra has different variants, however, the way of spreading was found to be drastically different. The malware was distributed through a set of fake websites which included fake mobile banking websites and websites which would prompt the user to install applications which allow for installation of Flash Player, matter of fact, the Flash Player variant was used for analysis. The following screenshot shows an example of such fake website, (Figure 28):

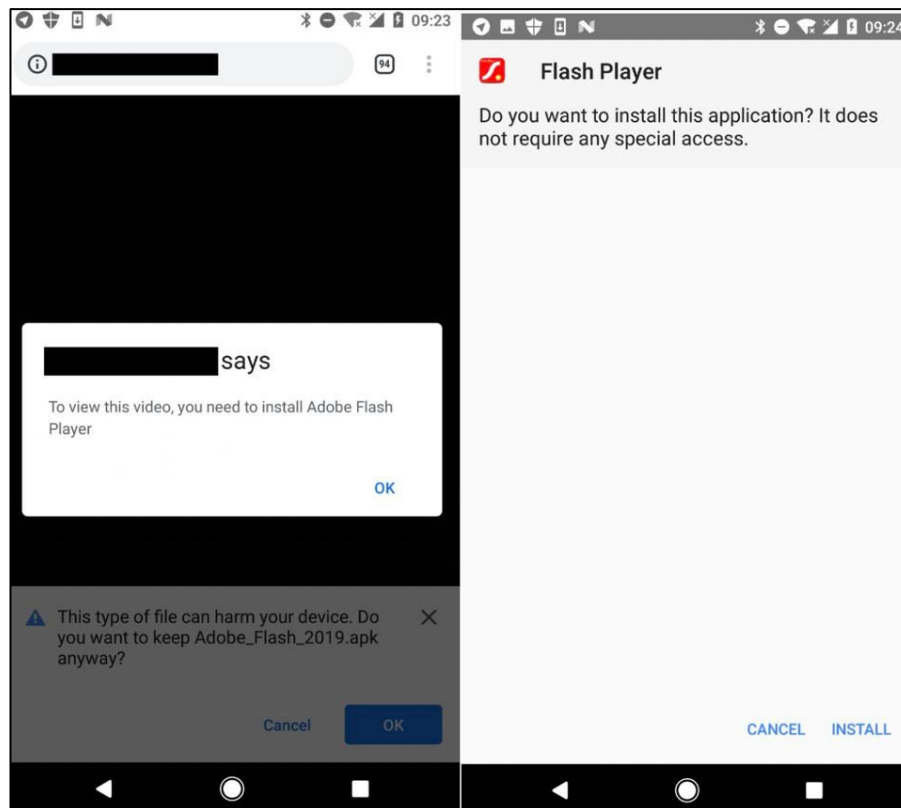


Figure 28: Hydra spreading using fake websites

Just like FluBot, Hydra exploits Android's Accessibility Service in order to obtain complete control over the victims' devices. Just like with FluBot, Hydra had a set of functionalities which the malware family was known for:

- Drops malicious files (is a trojan), like FluBot.
- Full remote control using Accessibility Service, like FluBot.
- Has a tendency to hide the launcher icon.
- Saved preferences contain a list of other banking apps' package names, meaning that the dropped files could potentially be able to communicate between each other with the aim of searching for the installed banking software.

- Can overlay other applications including key logging, like FluBot.
- The installer does not carry the malicious code, instead the initiator .APK file uses Tor (Deep Web) connection URL with the .onion extension to retrieve the main (base.apk) application which is then used for malicious intents.

### 2.3.2 Hydra malware Analysis

The malware sample with MD5 hash **90b53b42373895894c4308255d22c041** which was taken from the vx-underground malware repository was passed to the Joe Sandbox Cloud by dragging the file into the file submission area, then automated malware analysis report was generated, (Figure 29):

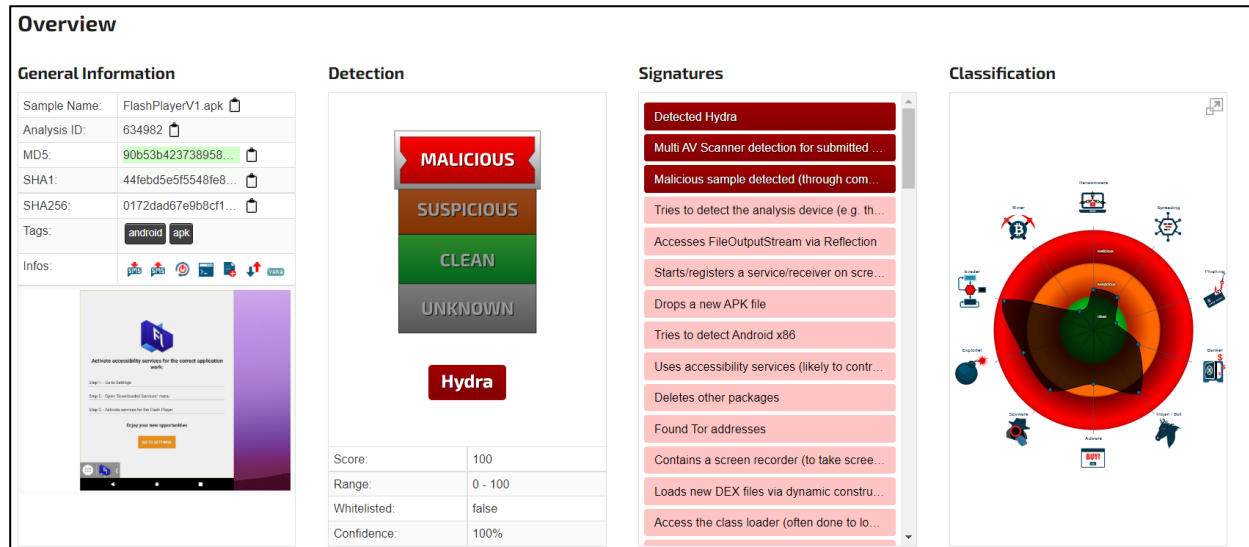


Figure 29: Generated report on Hydra by Joe Sandbox

The classification of the malware was also different from the one reported on FluBot sample. The following charts represent side by side comparison, (Figure 30):

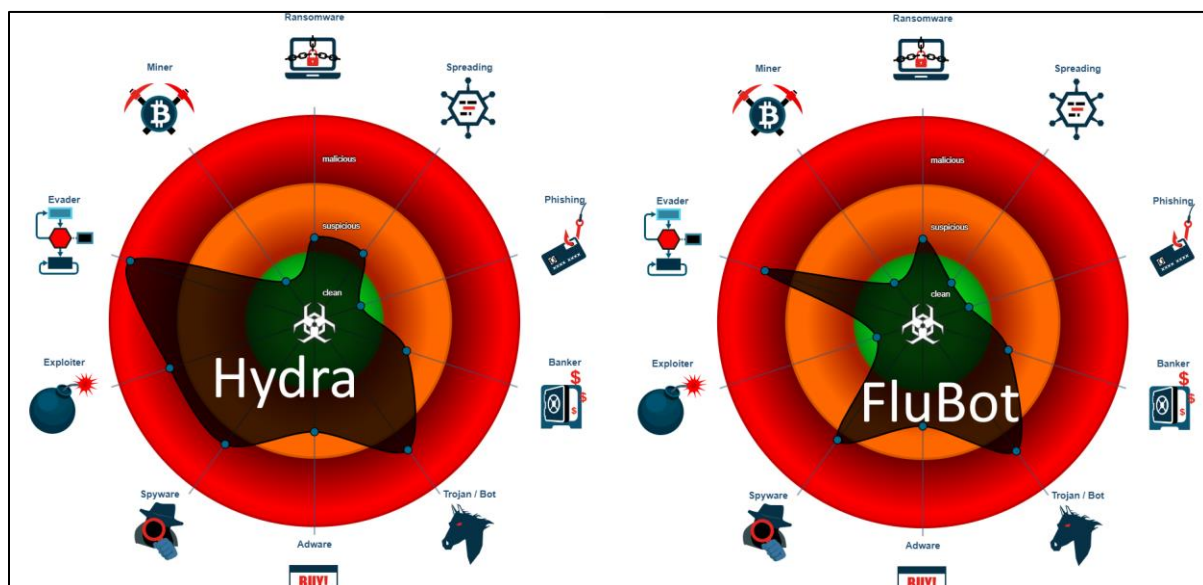


Figure 30: Hydra vs FluBot Classifications

By comparing the malware classifications, it was discovered that both malware samples were Trojans, and Spyware. It was also noted that Hydra malware had significantly more signatures than FluBot, which suggested that Hydra was significantly more complex. Please refer to the Appendix B for the complete Mitre Att&ck Matrix table.

The next step was to look through the created / dropped files to confirm the use of Tor connections as this was discovered as one of the signatures (Figure 31):

Networking	
Found Tor addresses	
Source: <a href="#">Linfo/guardianproject/netcipher/proxy/OrbotHelper;-&gt;isOnionAddress(Landroid/net/Uri;)Z</a>	Method string: ".onion"
Source: <a href="#">Linfo/guardianproject/netcipher/proxy/OrbotHelper;-&gt;isOnionAddress(Ljava/net/URL;)Z</a>	Method string: ".onion"

Figure 31: Tor connections discovered

The number of created / dropped files was also significantly higher in comparison with FluBot. In the case of Hydra, there were **eleven** dropped files which include the shared preferences file, seven Tor related files (including .zip file so possibly 1 zip and 6 files inside) and the base.apk which contained the main malware functionality, (Figure 32):

/data/user/0/com.chapter.pause/files/all_tor.zip	
File Type:	Zip archive data, at least v2.0 to extract
Category:	dropped
Size (bytes):	12390941
Entropy (8bit):	7.983098220313455
Encrypted:	false

Figure 32: Tor related dropped .zip file in Hydra sample

And the main Hydra malware code in a form of .APK file, (Figure 33):


/data/app/com.chapter.pause-9GZStfljYDk3lcZX7o6SHQ==/base.apk 	
File Type:	troff or preprocessor input, ASCII text, with very long lines
Category:	dropped
Size (bytes):	4490
Entropy (8bit):	3.3157490843773374

Figure 33: Main Hydra .APK dropped



Accessing the main malware code through Tor connection was not only an innovative way of file retrieval but also a very harmful and critical threat to the potential victims as the code can be easily updated by the attackers and become more evadable or malicious.

The application was targeting Android API 24 which was even older than the API targeted by the FluBot, the advantage of targeting older APIs was that the targeted victim audience would be even wider. That API version also did not have mandatory asking for permissions after installation of side loaded applications, (Figure 34):

Static APK Info	
General	
Label:	Flash Player
Minimum SDK required:	19
Target SDK required:	24
Version Code:	1

Figure 34: Hydra targeted Android API

During the static analysis, there were an extraordinary number of receivers, services, and permissions. It would probably be more efficient to mention services which were not in use, however, it was decided to outline the ones that stand out the most. For example, there was an interesting batch of receivers which were located in a single cell of the receivers table, (Figure 35):

Intent: android.intent.action.BOOT\_COMPLETED (Priority 999), android.intent.action.ACTION\_POWER\_CONNECTED (Priority 999), android.intent.action.BATTERY\_OKAY (Priority 999), android.net.wifi.WIFI\_STATE\_CHANGED (Priority 999), android.intent.action.SCREEN\_OFF (Priority 999), android.intent.action.DREAMING\_STOPPED (Priority 999), android.net.conn.CONNECTIVITY\_CHANGE (Priority 999), android.intent.action.ACTION\_POWER\_DISCONNECTED (Priority 999), android.intent.action.BATTERY\_LOW (Priority 999), android.intent.action.SCREEN\_ON (Priority 999), android.intent.action.BATTERY\_CHANGED (Priority 999), android.intent.action.REBOOT (Priority 999), android.intent.action.EXTERNAL\_APPLICATIONS\_AVAILABLE (Priority 999), com.htc.intent.action.QUICKBOOT\_POWERON (Priority 999), android.intent.action.USER\_PRESENT (Priority 999), android.intent.action.PACKAGE\_ADDED (Priority 999), android.intent.action.PACKAGE\_REMOVED (Priority 999), android.provider.Telephony.SMS\_RECEIVED (Priority 999), android.intent.action.SCREEN\_ON (Priority 999), android.intent.action.EXTERNAL\_APPLICATIONS\_AVAILABLE (Priority 999), android.net.conn.CONNECTIVITY\_CHANGE (Priority 999), android.net.wifi.WIFI\_STATE\_CHANGED (Priority 999), android.intent.action.DREAMING\_STOPPED (Priority 999)

Figure 35: Standing out set of receivers of Hydra malware

There were multiple receivers which were used to monitor the power state of the device such as if the device has booted or the battery state. Other receivers were monitoring the state of the device, if the device was put into the sleep state or not, also if the user was present and if the phone was unlocked. The malware sample even was listening for the rebooting of the victim's device. The priority was also set to value 999 which is very untypical for any kind of application developed for Android, even ones which are not meant to be malicious.

The services present were also different from those found in FluBot sample, a significant ration of those were dedicated to proxies and VPNs which would suggest the relationship between the Tor connections and used services by the Hydra malware sample. Other services included the Worker and Job Service which meant that the malware utilized multithreading. This suggested that Hydra was one of the most sophisticated pieces of Android malware on the market. During the static analysis it was also noted that the names used for methods and strings was rather open and understandable in contrast with the ones found in FluBot. Some of the proprietary services used for VPN connections and Notification listeners

were set to priority of value 0 which could suggest that Hydra implements its own proprietary listeners which meant to override the ones implemented by Google. The following screenshot outlines used services, (Figure 36):

Services	
• <u>com.sdktools.android.bot.components.commands.NLService</u>	• Intent: android.service.notification.NotificationListenerService (Priority 0)
• <u>com.sdktools.android.bot.components.injects.system.InjAccessibilityService</u>	• Intent: android.accessibilityservice.AccessibilityService (Priority 0)
• <u>com.sdktools.android.bot.components.screencast.ScreencastService</u>	
• <u>com.sdktools.android.bot.components.socks5.Socks5ProxyService</u>	
• <u>com.sdktools.android.bot.sms.HeadlessSmsSendService</u>	• Intent: android.intent.action.RESPOND_VIA_MESSAGE (Priority 0)
• <u>com.sdktools.android.core.PeriodicJobService</u>	
• <u>com.sdktools.android.core.injects_core.Worker</u>	
• <u>info.pluggabletransports.dispatch.service.DispatchService</u>	
• <u>info.pluggabletransports.dispatch.service.DispatchVPN</u>	• Intent: android.net.VpnService (Priority 0)
• <u>org.torproject.android.service.OrbotService</u>	

Figure 36: Strange services used by Hydra

Later, the list of requested permissions was inspected, Hydra had all the permissions as FluBot sample, however there was more than double the number of used permissions, it was decided to outline the permissions which did not match and discuss those only, (Figures 37 & 38):

Permission Requested
• android.permission.ACCESS_BACKGROUND_LOCATION
• android.permission.ACCESS_COARSE_LOCATION
• android.permission.ACCESS_FINE_LOCATION
• android.permission.ACCESS_NETWORK_STATE
• android.permission.ACCESS_NOTIFICATION_POLICY
• android.permission.ACCESS_WIFI_STATE
• android.permission.ACTION_MANAGE_OVERLAY_PERMISSION
• android.permission.BIND_ACCESSIBILITY_SERVICE
• android.permission.BIND_NOTIFICATION_LISTENER_SERVICE
• android.permission.BIND_VPN_SERVICE
• android.permission.BLUETOOTH
• android.permission.CALL_PHONE
• android.permission.CAPTURE_VIDEO_OUTPUT
• android.permission.CHANGE_WIFI_STATE
• android.permission.DISABLE_KEYGUARD
• android.permission.FOREGROUND_SERVICE
• android.permission.GET_TASKS
• android.permission.INTERNET
• android.permission.MODIFY_AUDIO_SETTINGS
• android.permission.QUERY_ALL_PACKAGES

Figure 37: Hydra requested permissions (Part 1)

• android.permission. <u>QUICKBOOT_POWERON</u>
• android.permission.READ_CONTACTS
• android.permission. <u>READ_EXTERNAL_STORAGE</u>
• android.permission.READ_PHONE_NUMBERS
• android.permission.READ_SMS
• android.permission. <u>RECEIVE_BOOT_COMPLETED</u>
• android.permission. <u>RECEIVE_LAUNCH_BROADCASTS</u>
• android.permission.RECEIVE_SMS
• android.permission. <u>RECORD_AUDIO</u>
• android.permission. <u>REORDER_TASKS</u>
• android.permission.REQUEST_DELETE_PACKAGES
• android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS
• android.permission. <u>REQUEST_INSTALL_PACKAGES</u>
• android.permission.SEND_RESPOND_VIA_MESSAGE
• android.permission.SEND_SMS
• android.permission. <u>SYSTEM_ALERT_WINDOW</u>
• android.permission. <u>USE_FINGERPRINT</u>
• android.permission.VIBRATE
• android.permission.WAKE_LOCK
• android.permission.WRITE_EXTERNAL_STORAGE
• android.permission.WRITE_SETTINGS
• android.permission.WRITE_SMS
• com.google.android.gms.permission. <u>ACTIVITY_RECOGNITION</u>

Figure 38: Hydra requested permissions (Part 2)

There were several notable requested permissions. To begin with, there were several permissions that requested the Location services, meaning the malware was able to determine the physical location of the victim's device. Furthermore, the application was able to manage the overlays on the device, this was used by the malware as it was designed to overlay mobile banking applications to steal credentials. Next, there was a permission which allows the malware to use VPN services, this was used for accessing Tor (Deep web) network in order to download the main functionality of the malware through the base.apk downloaded from .onion URL address. The Bluetooth functionality was also requested by Hydra, this could potentially be used for targeted attacks by locating the device id nearby, this could mean that the malicious potential of Hydra is much higher than it is portrayed in media. The infected application also was able to capture video output, meaning that it was possible to screen record the device. The next permission was very notable as disabling keyguard meant that the application would disable the password lock of the device if the victim has configured password unlock functionality. The infected application was also able to list the currently running tasks, which would be comparable to task manager which is part of Windows based operation systems. Hydra also had the ability to modify the audio settings, meaning that there was a possibility of the malware to intercept all audio output including private voice-over-internet communications. The permission QUICKBOOT\_POWERON meant that the malware sample had the ability to start on the booting of the device, essentially this may lead to the application being constantly running in background. Next, the application had the ability to access

external storage, in Android terms this would mean that the application had the access to all local files outside of the dedicated application storage on the device including image gallery. The application was also able to reorder tasks, which meant that the malware had the functionality to change the priorities of currently running tasks on the device. The malware was able to install other packages (application) on the device without the knowledge of the victim. Hydra was also able to create system alert messages which would mimic the actual system pop up notifications and dialog windows. Finally, the infected sample was also able to read the fingerprint sensors and it also had one custom permission which was meant to be recognized by device as genuine.

Unlike FluBot, Hydra did execute some of the requested permissions, they were mainly related to the installation of the Tor proxy services and the downloading of the main malware functionality as well as it accessed the contact list (Figure 39):

By Permission (executed)
ACCESS_NETWORK_STATE
CHANGE_COMPONENT_ENABLED_STATE
INTERNET
READ_CONTACTS
INSTALL_PACKAGES or NONE

Figure 39: List of executed permissions by Hydra

The rest of the application behavior which would differ from the one noted in FluBot analysis was related to communications using the Tor network which was out of the scope of this Android malware analysis investigation. The networking capabilities of Hydra was much more advanced which was portrayed by the number of packets send/received, (Figure 40):

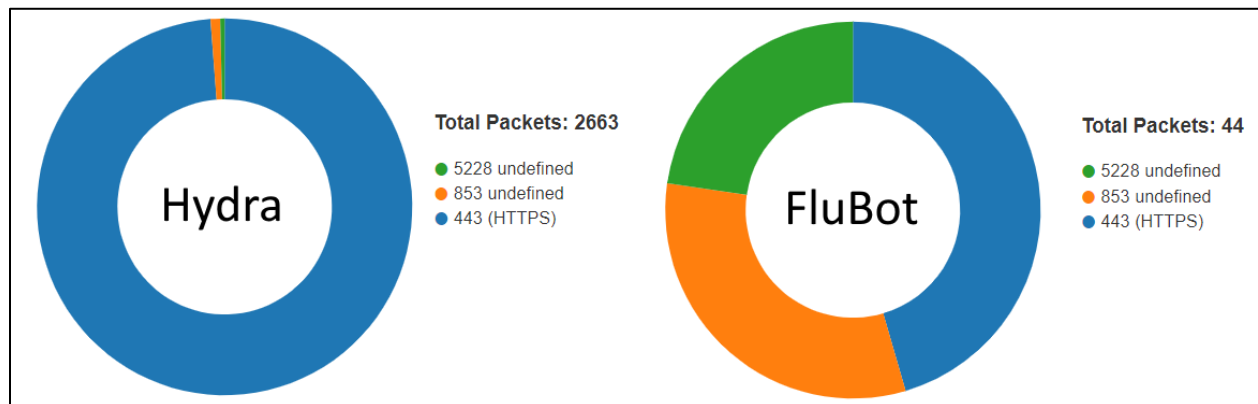


Figure 40: Networking stats, Hydra vs FluBot

The Joe Sandbox Cloud report on the Hydra variant used in this investigation could be found at the following URL:

<https://www.joesandbox.com/analysis/634982/0/html>

# DISCUSSION

## 2.4 THE RESULTS OF ANALYSIS AND COMPARISON

---

As the conclusion of the analysis of the two Android malware samples it could be proposed that the overall state of security of Android Systems were relatively low. Considering the fact that both, Hydra and FluBot utilized the exploitation of Android's Accessibility Services the standard implementation of such functionality is under a huge concern. The ability to sideload software onto any device running Android OS also adds to the fundamental security constraints. It was important to emphasize that the majority of Android devices are not sold with reliable antivirus software which means that the average user could potentially suffer from being infected with malware if any of their forms.

Speaking of the results of the analysis as per individualized approach per sample, the FluBot malware proved to be an advanced piece of malware which utilized most of the functionality of the Android OS functionality. The malware sample had a set of evasion techniques which were effective enough to assume that an average user would not notice the effects of the malware until critical moment such as the loss of funds from their bank account.

On the other hand, Hydra malware proved to be an example of one of the most complicated pieces of software designed for Android operating systems. During the analysis it was difficult to imagine that a Mobile operating system can perform such actions which bring it closer to the Desktop malware. All in all, Hydra was an example of true coding art and the level of advancement of it would be as high as potential risks to the proposed victims.

## 2.5 COUNTERMEASURES

---

Due to the fact that the vulnerability of Android to malware being fundamental, meaning that the attacker does not even have to implement proprietary algorithms for seizure of sensitive information as the code for accessing such is already implemented in Android using permissions and intents, there were a very limited number of truly effective countermeasures. One of such could be the installation of advanced anti-virus software such as Kaspersky, also the Avira AV was proven to be a reliable anti-virus which at the time of this investigation had the highest detection rate of the analyzed samples, although this does not guarantee the safety of the victim if encountered malware such as Hydra.

Another suggestion which can be made is to opt for the up-to-date device from Samsung as it seemed like that was one of the only companies which offer Android devices with proprietary malware defense mechanisms such as file encryption and automatic file deletion upon unauthorized access.

Lastly, it could be suggested for the average user to opt for a more fundamentally secure Operating System such as iOS which offer less functionality and customization, however, on the other hand the security standards are much higher than in Android due to the lack of ability to sideload software.

## 2.6 CONCLUSIONS AND ASSOCIATED CHALLENGES

---

To conclude on the topic of the Android Malware Analysis, it was suggested that the topic is not widely known, and the community interested or knowledgeable in that sphere is rather low. This is also portrayed in the availability of available tools and known techniques. The basic challenge which was associated with the analysis of the Android malware was the fact that there was no standardized methodology for the matter. Some could suggest that the standard malware analysis methodology could be used with the Android based samples, however, by the end of the investigation it was clear that the principles of operation of android software is too different for the standard methodology to be applicable.

The same ideas could be said about the availability of tools for malware analysis developed for Android. Due to the lack of awareness and interest within the community there was limited-to-none available opensource tools, in fact, there was only one Sandbox located which would meet the modern requirements of analysis, which was Joe Sandbox Cloud, what will happen if this tool goes fully paid due to its dominance?

## 2.7 FUTURE WORK

---

The future work past this investigation will involve a concise set of goals:

1. Research deeper into the networking principles behind Hydra malware.
2. Fetch and analyze different Android malware samples until certain correlation and similarities are found.
3. Open a repository and community platform for spreading awareness about android malware and the ways of analysis and countermeasures.

## REFERENCES

- Distribution of malware by OS 2020 | Statista. Available at: <https://www.statista.com/statistics/680943/malware-os-distribution/> (Accessed: 31 May 2022).
- Gdatasoftware.com. Available at: <https://www.gdatasoftware.com/news/g-data-mobile-malware-report-2019-new-high-for-malicious-android-apps> (Accessed: 31 May 2022).
- ANY.RUN vs. Joe Sandbox: Malware analysis tools comparison. Available at: <https://www.techrepublic.com/article/anyrun-vs-joe-sandbox/> (Accessed: 28 May 2022).
- Cloudi (2017) *How to install CuckooDroid? – All things in moderation*, Hydrasky.com. Available at: <https://hydrasky.com/malware-analysis/how-to-install-cuckoodroid/> (Accessed: 29 May 2022).
- Stephenson, P. and Stephenson, P. (2017) First Look: Joe Security Joe Sandbox Cloud, Scmagazine.com. Available at: <https://www.scmagazine.com/product-test/first-looks/first-look-joe-security-joe-sandbox-cloud> (Accessed: 30 May 2022).
- (2022) Reddit.com. Available at: [https://www.reddit.com/r/Malware/comments/s27592/joes\\_sandbox\\_pricing/](https://www.reddit.com/r/Malware/comments/s27592/joes_sandbox_pricing/) (Accessed: 30 May 2022).
- More, R. (2019) What is Ghidra and Why is it Important?, Help Desk Geek. Available at: <https://helpdeskgeek.com/free-tools-review/what-is-ghidra-and-why-is-it-important/> (Accessed: 30 May 2022).
- Meskauskas, T. (2021) FluBot Malware (Android), Pcrisk.com. Available at: <https://www.pcrisk.com/removal-guides/20475-flubot-malware-android> (Accessed: 31 May 2022).
- Where can I get Android Malware Samples?, researchgate.net. Available at: <https://www.researchgate.net/post/Where-can-I-get-Android-Malware-Samples> (Accessed: 31 May 2022)
- Manifest.permission | Android Developers (2022). Available at: <https://developer.android.com/reference/android/Manifest.permission> (Accessed: 31 May 2022).
- Android Malware Analysis : Dissecting Hydra Dropper – Pentest Blog (2019). Available at: <https://pentest.blog/android-malware-analysis-dissecting-hydra-dropper/> (Accessed: 31 May 2022).
- Bucur, I. and Labs, A. (2022) Avira Labs Research Reveals Hydra Banking Trojan 2.0 targeting a wider network of German and Austrian banks, Avira Blog. Available at: <https://www.avira.com/en/blog/avira-labs-research-reveals-hydra-banking-trojan-2-0> (Accessed: 31 May 2022).
- Malwareanalysis.co. Available at: [https://malwareanalysis.co/wp-content/uploads/2019/12/Android\\_Malware\\_and\\_Analysis.pdf](https://malwareanalysis.co/wp-content/uploads/2019/12/Android_Malware_and_Analysis.pdf) (Accessed: 31 May 2022).
- Doffman, Z. (2022) Warning As Devious New Android Malware Hides In Fake Adobe Flash Player Installations (Updated), Forbes. Available at:

<https://www.forbes.com/sites/zakdoffman/2019/08/16/dangerous-new-android-trojan-hides-from-malware-researchers-and-taunts-them-on-twitter/?sh=3d646c4a6d9c> (Accessed: 31 May 2022).

University, I. (2022) Use the VMWare Remote Console to boot a virtual machine from a locally stored ISO image, Kb.iu.edu. Available at:

<https://kb.iu.edu/d/azlh#:~:text=Launch%20the%20VMware%20Remote%20Console,enter%20the%20B IOS%20setup%20screen.> (Accessed: 31 May 2022).

Malware Analysis Explained | Steps & Examples | CrowdStrike (2022). Available at:

<https://www.crowdstrike.com/cybersecurity-101/malware/malware-analysis/> (Accessed: 31 May 2022).

Free Malware Sample Sources for Researchers (2022). Available at: <https://zeltser.com/malware-sample-sources/> (Accessed: 31 May 2022).

(2022) Exploit-db.com. Available at: <https://www.exploit-db.com/docs/english/33093-introduction-to-android-malware-analysis.pdf> (Accessed: 31 May 2022).

Cuckoo Sandbox - Automated Malware Analysis (2019). Available at:

<https://cuckoosandbox.org/download> (Accessed: 31 May 2022).



# APPENDICES

## APPENDIX A: FLUBOT MITRE ATT&CK MATRIX

Table 1: FluBot Mitre Att&ck Matrix

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Exfiltration	Command and Control	Network Effects	Remote Service Effects	Impact
Valid Accounts	Windows Management Instrumentation	Path Interception	Path Interception	Application Discovery <b>1</b>	OS Credential Dumping	System Network Connections Discovery <b>1</b>	Remote Services	Access Contact List <b>1</b>	Exfiltration Over Other Network Medium	Encrypted Channel <b>1</b>	Exploit SS7 to Redirect Phone Calls/SMS <b>2</b>	Remotely Track Device Without Authorization	Delete Device Data <b>1</b>
Default Accounts	Scheduled Task/Job	Boot or Logon Initialization Scripts	Boot or Logon Initialization Scripts	Obfuscated Files or Information <b>1 1</b>	LSASS Memory	System Network Configuration Discovery <b>1</b>	Remote Desktop Protocol	Location Tracking <b>1</b>	Exfiltration Over Bluetooth	Non-Standard Port <b>1</b>	Eavesdrop on Insecure Network Communication <b>1</b>	Remotely Wipe Data Without Authorization	Carrier Billing Fraud <b>1</b>
Domain Accounts	At (Linux)	Logon Script (Windows)	Logon Script (Windows)	Evade Analysis Environment <b>1</b>	Security Account Manager	Location Tracking <b>1</b>	SMB/Windows Admin Shares	Network Information Discovery <b>1</b>	Automated Exfiltration	Remote Access Software <b>1</b>	Exploit SS7 to Track Device Location	Obtain Device Cloud Backups	Delete Device Data
Local Accounts	At (Windows)	Logon Script (Mac)	Logon Script (Mac)	Binary Padding	NTDS	Application Discovery <b>1</b>	Distributed Component Object Model	Input Capture	Scheduled Transfer	Application Layer Protocol <b>1</b>	SIM Card Swap		Carrier Billing Fraud
Cloud Accounts	Cron	Network Logon Script	Network Logon Script	Software Packing	LSA Secrets	System Information Discovery <b>1</b>	SSH	Keylogging	Data Transfer Size Limits	Fallback Channels	Manipulate Device Communication		Manipulate App Store Rankings or Ratings
Replication Through Removable Media	Launchd	Rc.common	Rc.common	Steganography	Cached Domain Credentials	Evade Analysis Environment <b>1</b>	VNC	GUI Input Capture	Exfiltration Over C2 Channel	Multiband Communication	Jamming or Denial of Service		Abuse Accessibility Features
External Remote Services	Scheduled Task	Startup Items	Startup Items	Compile After Delivery	DCSync	Process Discovery <b>1</b>	Windows Remote Management	Web Portal Capture	Exfiltration Over Alternative Protocol	Commonly Used Port	Rogue Wi-Fi Access Points		Data Encrypted for Impact

## APPENDIX B: HYDRA MITRE ATT&CK MATRIX

Table 2: Hydra Mitre Att&ck Matrix

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Exfiltration	Command and Control	Network Effects	Remote Service Effects	Impact
Valid Accounts	Windows Management Instrumentation	Path Interception	Path Interception	2 Application Discovery	2 Capture SMS Messages	1 System Network Connections Discovery	Remote Services	1 Access Contact List	Exfiltration Over Other Network Medium	1 Encrypted Channel	2 Exploit SS7 to Redirect Phone Calls/SMS	Remotely Track Device Without Authorization	2 Generate Fraudulent Advertising Revenue
Default Accounts	Scheduled Task/Job	Boot or Logon Initialization Scripts	Boot or Logon Initialization Scripts	Rootkit	1 Access Stored Application Data	1 Security Software Discovery	Remote Desktop Protocol	1 1 Location Tracking	Exfiltration Over Bluetooth	1 Non-Standard Port	Exploit SS7 to Redirect Phone Calls/SMS	Remotely Wipe Data Without Authorization	1 1 Delete Device Data
Domain Accounts	At (Linux)	Logon Script (Windows)	Logon Script (Windows)	Obfuscated Files or Information	Security Account Manager	1 1 Location Tracking	SMB/Windows Admin Shares	1 Capture Audio	Automated Exfiltration	1 Multi-hop Proxy	Exploit SS7 to Track Device Location	Obtain Device Cloud Backups	1 Carrier Billing Fraud
Local Accounts	At (Windows)	Logon Script (Mac)	Logon Script (Mac)	Binary Padding	NTDS	2 Application Discovery	Distributed Component Object Model	1 Network Information Discovery	Scheduled Transfer	1 Application Layer Protocol	SIM Card Swap		Carrier Billing Fraud
Cloud Accounts	Cron	Network Logon Script	Network Logon Script	Software Packing	LSA Secrets	1 System Information Discovery	SSH	1 Screen Capture	Data Transfer Size Limits	2 Proxy	Manipulate Device Communication		Manipulate App Store Rankings or Ratings
Replication Through Removable Media	Launchd	Rc.common	Rc.common	Steganography	Cached Domain Credentials	1 Process Discovery	VNC	2 Capture SMS Messages	Exfiltration Over C2 Channel	Multiband Communication	Jamming or Denial of Service		Abuse Accessibility Features
External Remote Services	Scheduled Task	Startup Items	Startup Items	Compile After Delivery	DCSync	Network Sniffing	Windows Remote Management	1 Access Stored Application	Exfiltration Over Alternative Protocol	Commonly Used Port	Rogue Wi-Fi Access Points		Data Encrypted for Impact