PENETRATION TESTING FOR ANDROID APPLICATIONS WITH SANTOKU LINUX

A Project

Presented to the

Faculty of

California State Polytechnic University, Pomona

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

In

Computer Science

By

Ahlam Mohammad Almusallam

2018

SIGNATURE PAGE

PROJECT: PENETRATION TESTING FOR ANDROID APPLICATIONS WITH SANTOKU LINUX

AUTHOR: Ahlam Mohammad Almusallam

DATE SUBMITTED: Spring 2018

Computer Science Department

Dr. Mohammad Husain _____
Project Committee Chair
Department of Computer Science

Dr. Yu Sun _____
Department of Computer Science

ACKNOWLEDGEMENTS

This project wouldn't be possible without the help of the following people, who share my dream of achieving a master degree. First, Dr. Mohammad Husain has been a true inspirational person who introduced me to the world of Cyber Security. I am deeply thankful for the opportunity to work with him. Thank you for providing me all the knowledge and unlimited support that I need to complete this work.

The highest appreciation to my parents Mr. Mohammad and Mrs. Nawal, for their huge love, blessings, and all the moral support. Thank you for being an important part of whatever I pursue. My special and heartfelt gratitude to my sisters and my brother for giving me strength and encouragement to keep going when times are tough. The sincerest thanks to my grandfather, Ibrahim, for his precious lessons and wisdom that I will always carry throughout my life's journey. I sincerely thank my family members who always being there while achieving my goals. Also, I give deep thanks to my dear friend Miss. Nadiah Alahmari for motivating me from time to time.

I must thank all my professors whom I have taken classes with over the two years especially Dr. Young and Dr. Yu Sun. Thank you for the hard work and effort that you put into teaching us. Finally, I am so grateful to Saudi Arabia Culture Mission (SACM) scholarship for making possible for me to study here in USA.

May God bless all of you today and always.

ABSTRACT

Today, the majority of people have become completely dependent on mobile applications. These applications could be a gateway to sensitive data that attract more hackers. The Android penetration testing process helps to address security weaknesses or vulnerabilities in the Android platform. In this paper, we present different penetration tests (or pentests) for Android-based mobile applications in a very comprehensive manner. First, we explain how to set up a pentesting environment, including hardware/software requirements, USB debugging, and configuring a proxy. Next, we perform some of the most popular pentesting tools such as AFLogical, Dex2jar, JD-GUI, Apktool and Drozer by using Santoku Linux distribution. Finally, we conduct an Android repackaging attack on selected apps by using Santoku Linux distribution and then demonstrate the attack on our Android VM. This work attempts to give developers and security professionals a step-by-step guide for Android mobile application pentesting.

Key Words:
Penetration Testing, Android application, Reverse Engineering, Santoku, Mobile Security, Cyber Security.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

In the present day, people use their smartphones in almost every aspect of their life, social interactions, career, finance, learning, and even health. According to Statista [1], “all of the non-gaming app publishers in the Google Play Store had double-digit download figures without any signs of slowing down.” The rapidly increasing number of mobile application users has attracted more hackers. This not only increases the possibility and number of smartphones that hackers could target, but it also increases the possibility to hack any system or device that connects to the same network. Smartphone penetration testing is one of the most essential kinds of security pentesting that demonstrate what exploitable vulnerabilities exist within application and the level of damage that could occur if the application is hacked. In general, penetration testing or (ethical hacking) is the process of discovering security weaknesses and raising the security level of systems, networks, or applications as performed by a penetration tester or auditor. Also, it is important to point out that penetration testing (often shortened as pentesting) is commonly mistaken as vulnerability testing. In fact, vulnerability testing is to identify potential problems, where penetration testing is meant to analyze those problems as well as attack the system to determine vulnerabilities.

Among the mobile OSs, Android is one of the most popular and the fastest growing mobile operating system. However, during application development, sometimes developers make security mistakes or ignore the secure coding practices. So, Android apps often contain security weaknesses that need to be discovered before a successful hack. Application vulnerabilities, insecure password storage, information disclosure, manifest privileges, insecure file permissions, and DDoS attacks are all common vulnerabilities found among

pentesting applications. The more protection the application needs, the more often pentesting should be implemented to decrease the possibility of any attack.

There is a large amount of resources of mobile application pentesting tools that are available on the Internet. In this paper, we use Santoku Linux distribution [10] that is designed specifically for mobile forensics, mobile malware analysis, and mobile security testing. Basically, we cover the main steps that needed to conduct Android application pentesting such as setting up the environment, configuring proxy, and performing some security tools required for Android pentesting. Also, we conduct Android mobile application repackaging attack as a simulation of how an attacker could repack an Android application.

1.1. The Basics of Android Architecture

Android is an open source, Linux-based software stack divided into five main layers as shown in the architecture diagram below, Figure 1. Basically, it is designed in the form of a software stack architecture that contains four core layers: an applications layer, application framework layer, libraries layer, a runtime environment, and Linux kernel layer.

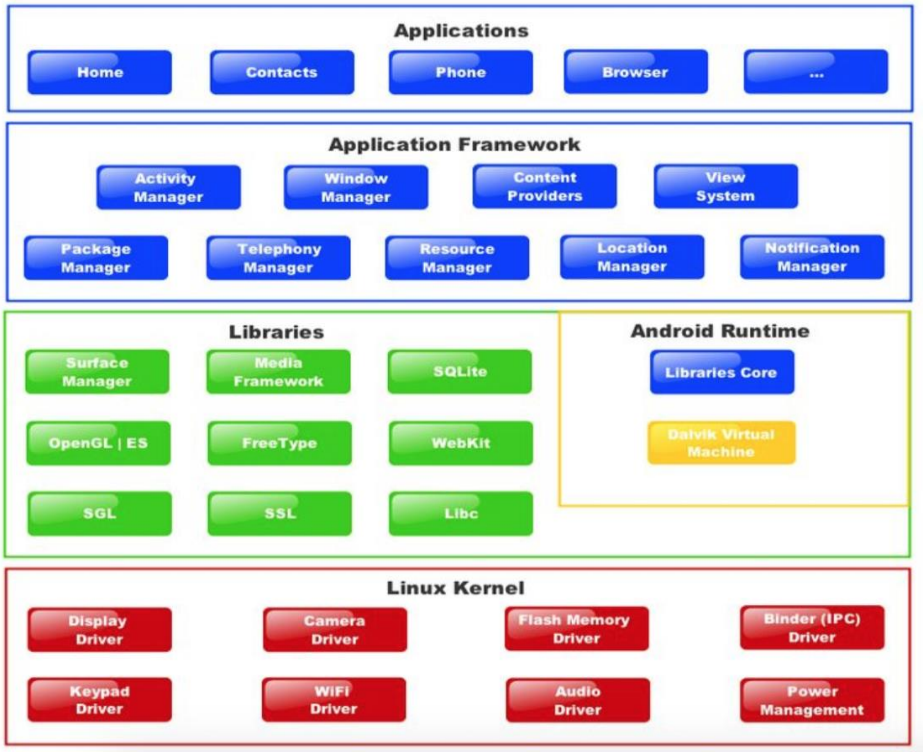

Figure 1 Five core layers of Android [43]

1. **Application layer**

The application layer is the top layer of the stack. It contains native applications and third-party applications that are installed by a user such as (WhatsApp and Snapchat).

2. **Application Framework layer**

The application Framework layer provides many upper level services to applications that manage and control the application layer. In this layer, the developers of the application are the only people who are allowed to control installed applications.

The Android framework includes four types of Android application components:

- **Activities:**

It controls all tasks through graphical user interface performed by a user.

- **Content:**

It provides access to the installed applications to share information with other applications.

- **Services:**

Provides access to non-code embedded services which allow the user to execute multiple application during one time such as listing to music while using the internet.

- **Notifications or Broadcast:**

It displays alerts and notifications that performed from the application to the user.

3. **Libraries**

This layer controls and accesses applications data. Android provides a lot of C/C++ libraries for different uses. Here some of the most useful libraries:

- System C Library: It allows developers to create applications.
- Media framework: It is useful to program video and audio files.
- android.content: It allows messaging between applications and application components.
- Android.webkit: It provides access to the internet within an application.

4. **Android Runtime**

This layer consists of Dalvik Virtual Machine (DVM) and a set of core java programming libraries. Before running any java applications, java files are converted into Dalvik format (dex) to be optimized for a minimum memory.

5. **Linux Kernel**

This layer is the most important layer because it controls core services such as hardware, memory management, power controls, security, and rest of the software stack.

2. BACKGROUND AND LITERATURE SURVEY

2.1. Android Background

Android OS has become one of the most popular OS in the world. Android OS is a Linux based platform developed by Google and Open Handset Alliance (OHA). Most Android applications are written in Java and compiled to Dalvik bytecode which is the official language of Android development. The first version of Android was released in September 2008 with no specific code name [11]. Figure 3 shows all versions of Android OS with the code names in alphabetical order since 2009's Android 1.5 Cupcake until the most recent version, Android P 9, released in May 2018. Today, Android OS is not only developed for computers and phones, but for all of things we interact with on our daily basis such as a refrigerator, an air conditioner, oven, and more.

Code name	Version number	Initial release date	API level	Security patches[1]
(No codename)[2]	1.0	September 23, 2008	1	Unsupported
(Internally known as "Petit Four")[2]	1.1	February 9, 2009	2	Unsupported
Cupcake	1.5	April 27, 2009	3	Unsupported
Donut[3]	1.6	September 15, 2009	4	Unsupported
Eclair[4]	2.0 – 2.1	October 26, 2009	5 – 7	Unsupported
Froyo[5]	2.2 – 2.2.3	May 20, 2010	8	Unsupported
Gingerbread[6]	2.3 – 2.3.7	December 6, 2010	9 – 10	Unsupported
Honeycomb[7]	3.0 – 3.2.6	February 22, 2011	11 – 13	Unsupported
Ice Cream Sandwich[8]	4.0 – 4.0.4	October 18, 2011	14 – 15	Unsupported
Jelly Bean[9]	4.1 – 4.3.1	July 9, 2012	16 – 18	Unsupported
KitKat[10]	4.4 – 4.4.4	October 31, 2013	19 – 20	Unsupported[11]
Lollipop[12]	5.0 – 5.1.1	November 12, 2014	21 – 22	Unsupported[13]
Marshmallow[14]	6.0 – 6.0.1	October 5, 2015	23	Supported
Nougat[15]	7.0 – 7.1.2	August 22, 2016	24 – 25	Supported
Oreo[16]	**8.0 – 8.1**	August 21, 2017	26 – 27	Supported
Android P[17]	9	May 8, 2018 (beta)	28	Presupported
Legend: Old version	Older version, still supported	**Latest version**	Latest preview version	

Figure 2 Android OS versions from 2008-2018 [11]

2.2. Literature Survey

Since the release of the first Android smartphone in 2008, there have been a number of research projects and documents analyzing different mobile pentesting. For example, Aditya Gupta [2] published the book *Learning Pentesting for Android Devices* that gives a step-by-step process to set up a penetration testing environment to conduct Android pentesting. It covers some of the methods and techniques that are used to reverse the Android applications as well as the interception of traffic in applications. The best part of the book is both the logical and physical acquisition of forensic data, as well as the tools that could ease the process of data extraction. In addition, it covers information of the SQLite [12] databases used by Android to store data. Furthermore, it covers various lesser-known techniques helpful in Android pentesting that include topics such as WebView [13] vulnerabilities and exploitation. Finally, the book gives an introduction about the ARM (Architecting a Smarter World) [14] platform on which most smartphones run today. This book helps security researchers and developers who choose to enter the mobile security field learn more about mobile penetration testing and to be more aware of security risks.

Another example of related work, Naresh Kumar's Master's thesis [3] reports about performing a security analysis of Android-based Smartphones. It focuses on WLAN network connectivity to perform the penetration tests by using three Android mobile versions. The report [3] describes port scanning and its output followed by a summary of the findings. Also, it describes different types of attacks that performed against the Android network stack. The works aims to explore and describe Android architecture, explore the security mechanisms present in the Android OS, and identify and evaluate security problems in the network stack of the Android OS.

3. PENETRATION TESTING

3.1. What is Penetration Testing?

A penetration test is the act of discovering security weaknesses or vulnerabilities in a system before they are discovered by an attacker. The test is performed by a legal and authorized real-world simulated attack from malicious insiders or outsiders. The purpose is to secure a target system, network, or web application of an organization against future attacks. Penetration testing is sometimes called pentesting, ethical hacking, legal hacking, and white hat hacking. Usually, a pentest has a certain time frame with a certain target to conduct the test. The time frame to perform a test could take days or weeks depending on the type or scope of that test. There are two common pentesting tools which are static analysis tools such as reverse engineering tools and dynamic analysis tools such as Wireshark tool [31]. A pentest usually results in the documentation of each vulnerability or potential issue discovered, which will be then presented to the owner of the target system. There are many types of the pentesting such as network penetration testing, application penetration testing, website penetration testing, physical penetration testing, cloud penetration testing, and social engineering.

3.2. Pentesting Strategies

1. **Targeted testing strategy:**

Targeted testing is conducted by the organization's security team and a pentester; both parties are working together.

2. **External testing strategy:**

An external penetration test is conducted by a professional pentester outside the network to test different resources available for anyone in the public. This test only provides the pentester limited information such as URLs or IP addresses. This strategy is an attempt to find out ways how an attacker could get unauthorized access to the system.

3. **Internal testing strategy:**

An internal pentest, on the other hand, is conducted by a professional pentester inside the network to test different resources available to anyone inside the security perimeter. This type to find out how much damage a disgruntled employee could cause.

4. **Blind testing strategy:**

A blind test strategy does not provide the pentester with any information about the organization except its name. It simulates real hacking, so the pentester is responsible for gathering information such as a website and domain name of the organization by using publicly available resources. During this test, organization employees and security team are informed about this pentest.

5. **Double blind testing strategy:**

This strategy is similar to the blind testing strategy. The only difference is that one or two people within the organization might know that a test is being conducted.

3.3. Penetration Tester

A penetration tester, also known as good guy, ethical hacker or pentester, is essentially taking on the legal role of an attacker. The difference between a pentester and a hacker is authorization. Authorization is the process of getting legal or authorized access to such a system. Basically, a pentester's job involves performing various test of products or systems

for exploitable security flaws or vulnerabilities for the purpose of making these systems more secure.

In order to conduct a pentest, the pentester should follow the below steps:

1. **Scope and goal:**

Before conducting a pentest, the pentester should have a document about the specific scope and goals of the target system, and it should be in detail and agreed upon. Mostly, that document includes information about a certain target system, time frame to perform a test, a certain type of computer asset, and how much detail is needed at the end of the pentest.

2. **Select pentesting tools:**

Most professional pentesters use specific pentesting tools that find frequent use. However, depending on the pentest, sometimes a pentester has to use recently published tools to conduct that test. Also, it is recommended for any pentester to be aware and be updated with any new pentesting security tools and technologies.

3. **Discovery: Learn about your pentest target:**

A pentester should have complete information about the system or the network that is being tested such as IP addresses, OS platforms, applications, patch levels, network ports, and users.

4. **Exploitation: Break into the target asset:**

After gathering information and selecting the needed tools, this is the most important step of the pentest which is: break-in into the target system to exploit vulnerabilities.

After conducting the test, a pentester should provide a report to the owner of how the target systems were breached and provide ways to prevent future attacks.

If an organization hires an unauthorized pentester, it may result within a major problem such as crashing the target system. This means any organization should use a professional pentester to prevent having problems during the test. In the worst case, an unprofessional pentester could crash the target along with all data that is involves in the test. A careful consideration should be given in order to select a pentester or pentest team. Past experience would help a lot in determining the appropriate pentest. For example, try to find a pentester who performed a test with similar scope and tools. Also, it's better to find a pentester with a long experience in pentesting.

3.4. Five Stages of Penetration Testing Methodology

The overall process of the pentesting can be divided into five stages which is called pentesting methodology:

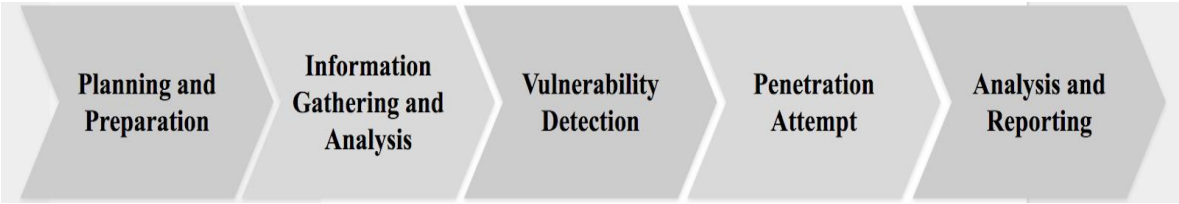

Figure 3 Five stages penetration testing methodology

1. **Planning and Preparation**

Before conducting the test, a pentester and a target system's organization should make and agree upon a detailed documentation. That document should include a clear objective for the pentest and the degree of exploitation. In case the organization does not have a clear objective, the pentest could find no vulnerabilities. Also, the deal should have all information about a certain target system such as IP addresses, operational requirements,

OS platforms, network ports, and all users of target entity. Moreover, both parties should have agreed about the time frame to perform the pentest. Sometimes a pentester performs a test that could stop the services of the target network. For example, performing Denial of Service attack (DOS) would cause flooding the target with traffic that prevent users from accessing information or services. So, a pentester should have a specific day and time to perform any pentest that could make the system or network resource unavailable to the target users. In addition, the organization should inform their pentester whether there is a security team that would detect or prevent testing. The pentester will use this documentation during the pentest to make sure he/she is working within the scope. Finally, both parties should have a clear view of expected findings that should be obtained from the pentest.

2. Information Gathering and Analysis

When a pentester and an organization get acquainted with goals and the scope of the pentest, the pentester should start the pentesting with an information gathering phase. Mostly, the pentester locates publicly available information about the target system in order to use information found to exploit vulnerabilities into that system. Pentesters use tools to gather information such as Wirsharks [31] and Nmap[27]. By using pentesting tools, a pentester is expected to get information about the target system or network, including a list of all systems and IP addresses that the target entity is using along with information about the software, fingerprint of OS, running services, open ports, and the applications in the target network. There are two types of information gathering process: passive and active information gathering. Passive information gathering means to only obtain information about a targeted system without actively establishing contact with the target. In contrast,

active information gathering means to interact directly with a system in order to actively exploit vulnerabilities or determine weaknesses in the target system.

3. Vulnerability Detection

After gathering information, a pentest discovers and identifies any possible vulnerabilities that might exist in the target system. This phase is called vulnerability detection or vulnerability scanning. In this step, a pentest will produce a list that contains almost all known vulnerabilities in the target system. There are a lot of free tools that help to provide a detailed picture of the flaws and vulnerabilities such as Nessus [37]. Note that there are some vulnerabilities that are unknown to the public, so it could not be found by using vulnerability scanning phase.

4. Penetration Attempt

After finding the vulnerabilities, the pentester now starts to exploit the identified vulnerabilities and breach the system. So, a pentester needs to have a list of all presence vulnerabilities that were found in the previous phase to conduct the test. The pentest includes password cracking, network exploitation, social engineering and even physical security testing.

5. Analysis and Reporting

This phase is the main point of the overall test. A pentester now should have a specific information on what findings were compromised and all exploited vulnerabilities. After that, the pentester should clean up the systems and delete traces of information, such as user accounts, that were created during the pentesting.

4. ANDROID APPLICATIONS PENETRATION TESTING

4.1. Introduction to Android Applications Penetration Testing

Android mobile applications have been a gateway for hackers to extract sensitive information. Mobile applications pentesting is a critical step in the development of any secure application and it has been widely adopted by lots of companies around the world. The goals of mobile pentesting are to identify, fix, and prevent security flaws and vulnerabilities in any kind of software application (whether web services or APIs). Mobile app pentesting requires the use of different security tools for evaluating mobile application and infrastructure vulnerabilities. The results of any pentest will be documented as a report, and the vulnerabilities found could then be resolved.

4.2. Ways to Analyze Android Traffic: Passive and Active

There are two ways to capture Android traffic, passive analysis and active analysis:

1. **Passive analysis:**

In this scenario, a pentester has no active interception with the applications. He is only allowed to capture network packets and later view the data by using a network analyzer tool (such as Wireshark [31]) to finds vulnerabilities or weak security issues.

2. **Active analysis:**

In contracts, within an active analysis, a pentester has active interception with the applications. He is allowed to capture and analyze all network packets on the fly and view a description of all captured data. In this scenario, a pentester needs to setup a proxy to have full access for all the requests that have been done within that particular proxy.

5. SELECTING THE TESTING ENVIRONMENT

The most secure form of pentesting is to practice within a virtual environment which means running a virtual machine within a real machine. For example, running Kali Linux image [16] within a computer that runs on Windows operating system [17] by using a virtualization system such as VMware [18] or VirtualBox [19]. In this scenario, Kali Linux called "gust" and Windows called "host". Why using virtual environment while pentesting is more secure? Because if the guest OS gets hacked, the host OS will be save.

Pentesting and security auditing requires special tools to implement the test. Kali Linux [16], Pentoo[20], Backbox[21], Parrot Security OS[22], and NodeZero Linux[23] are a group of the best Linux pentesting distributions. Each of them has a very unique set of tools. So, it was not easy to figure out which one we should use for this research. Also, it could take a lot time to search for appropriate tools within different areas such as mobile security, malware analysis, and forensics. After a lot of searching, we choose Santoku Linux distribution [10] because it built especially for mobile pentesting and forensic investigation side. Also, Santoku [10] already comes with pre-installed tools that we would use for our pentesting, such as AFLogical [24], Apktool [39], Dex2jar [30], and Drozer [38] without the need of installing each tool individually.

5.1. Software and Hardware Requirements

After digging in the Internet and having a better idea of pentesting lab requirements, we choose the below:

Software requirements:

- VirtualBox 5.1.30
- Santoku 0.5 Linux VM
- Android x86 ISO
- Android Virtual Device AVD by using Android SDK Manager

Hardware requirements:

- MacBook Pro OS version 10.11.6 (Minimum 16 MB ram)
- Samsung Galaxy S4 Android version 5.0.1

6. SETTING UP THE TESTING ENVIRONMENT

6.1. Running Santoku in VirtualBox

Download Santoku image and VirtualBox from the official websites. Then, launch Santoku VM within VirtualBox.

Since each vulnerability analysis area requires different tools to examine the security configuration of the application, the tools of Santoku is categorized into four areas: device Forensics tools, penetration testing tools, reverse engineering tools, and wireless analyzers tools Figure 4.

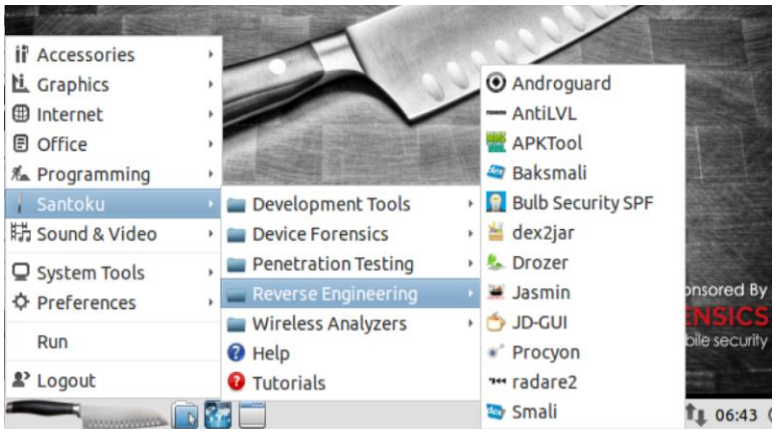

Figure 4 Santoku Linux distribution desktop

6.2. Steps to Set Up Android Device

Device model: Samsung Galaxy S4 Android version 5.0.1

1. Unlock developer options in Android:

First, we need to unlock developer options to enable USB debugging for Android Debug Bridge (adb) development, otherwise, we won't be able to access our app files during the pentest. Adb is a command line that comes with the Android SDK to communicate between PC and Android device. “USB Debugging is required by adb, which is used for rooting, backing up, installing a custom ROM, taking screenshots from computer and more [34].”

Developer options are disabled by default, to unlock it we need to follow the procedures below:

Settings > about device > build number

Tap on "build number" 7 times and after the 7th tap the developer options will be unlocked and we will see the message "You are now a developer!". Now, we can access all the advanced settings such as USB debugging Figure 5.

Figure 5 Unlock developer options

2. Enable USB debugging

Now, with USB debugging enabled Figure 6, we could connect our device to our PC, so we could run adb commands that we will need in almost every attack.

Settings > Developer Options > USB Debugging.

After that, we installed SuperSU[35], KingRoot[15], and Terminal Emulator[36] tools in our phone to root the device and it successfully rooted Figure 7.

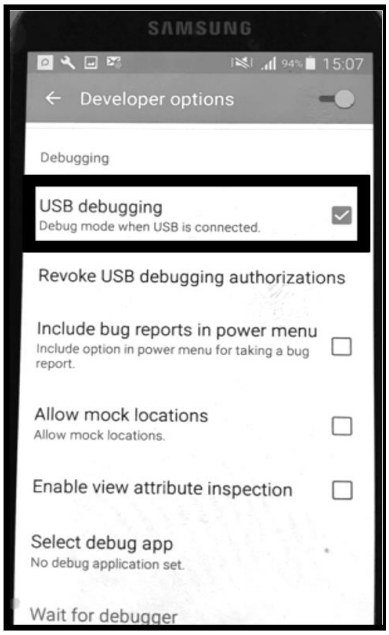

Figure 6 USB debugging

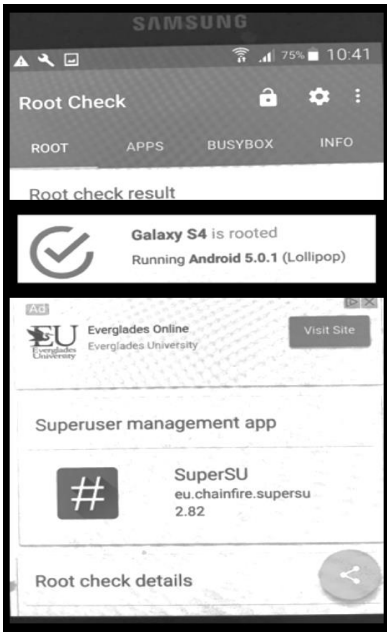

Figure 7 Rooted our device

3. Setting up a proxy

We need to set device's proxy to be as VM's proxy. This could be done by the below steps:

- Settings> WIFI
- Long tap on connected network's name
- Modify network config
- Show advanced options
- Set proxy settings > Set Proxy to ″Localhost″ and Port "8080"

6.3. Steps to Set Up Android Emulator

There are a lot of ways to build Android Virtual Device such as Genymotion [33], Android Studio [44] and Android SDK Manager [45]. In this paper, we use Android SDK Manager to build our emulator.

1. **Android SDK Manager:**

Deploy an emulator in Santoku [10] is straightforward. When launching an Android emulator, all we need to do is open the Android Virtual Device (AVD) tool either from SDK folder or straight from Santoku tool Figure 8. Then, we can set up a new device and custom features.

Figure 8 Open SDK folder straight from Santoku Linux

2. Connect the Device

After launching the emulator Figure 9, we could connect it by the below command.

$adb devices

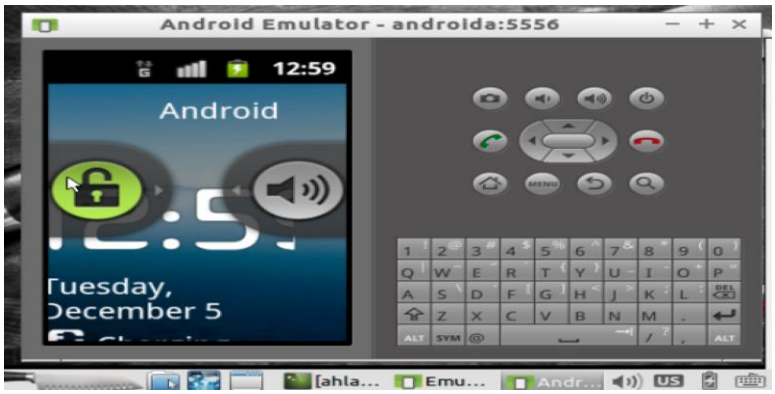

Figure 9 Android emulator

In case you have any errors, the commands below help to restart the adb:

- adb kill-server (to terminate adb)
- adb start-server (checks if the adb is running, and if not, it will restart it)

7. IMPLEMENTATION AND TOOLS

7.1. Device Forensics Tools

7.1.1. AFLogical Tool

The AFLogical OSE tool [24] is an Android forensics tool developed by viaForensics. It allows an examiner to extract CallLog Calls, Contacts Phones, MMS, and SMS messages from Android devices.

By using Android emulator:

Since we built our emulator successfully, we first navigate to:

Santoku > Device Forensics > AFLogical OSE

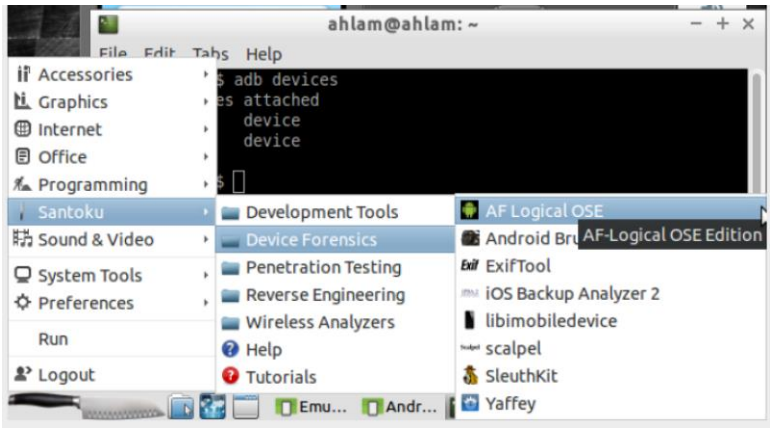

Figure 10 AFLogical OSE is preinstalled in Santoku

After that, we run the below command to extract CallLog Calls, Contacts Phones, MMS, and SMS messages.

$AFLogical OSE

A window will pop up with the information that we want to capture. So, we select what all the relative options and then press the "Capture" icon. Now, all the information will be saved in (Aflogical-data/forensics) file. In Figure 12, we could see the contacts and the messages that were captured.

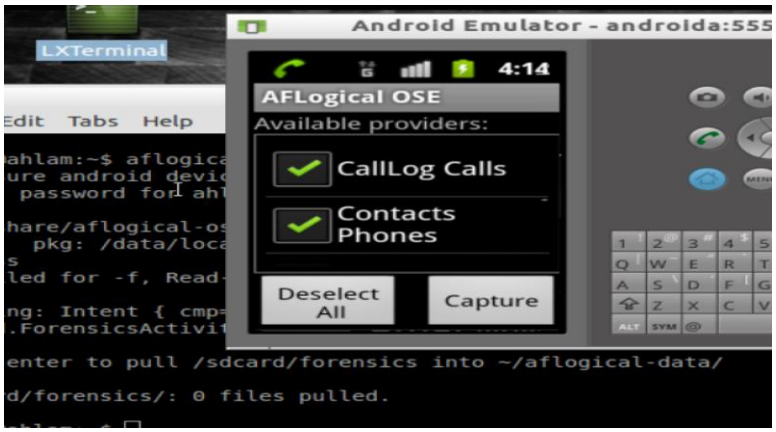

Figure 11 Choose information that weant to capture

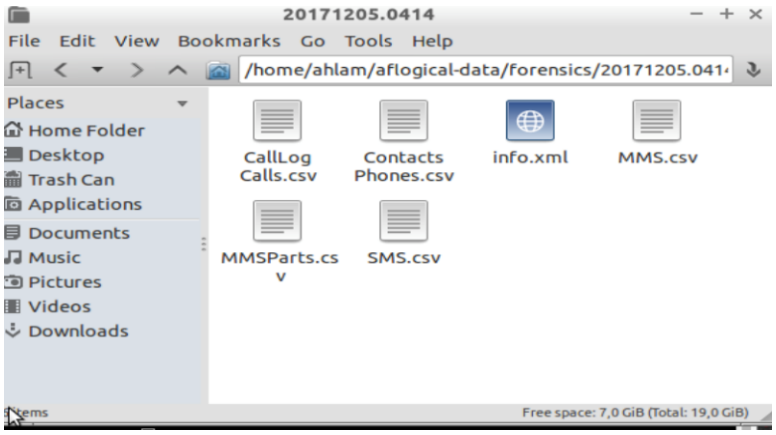

Figure 12 Information saved in (Aflogical-data/forensics) file

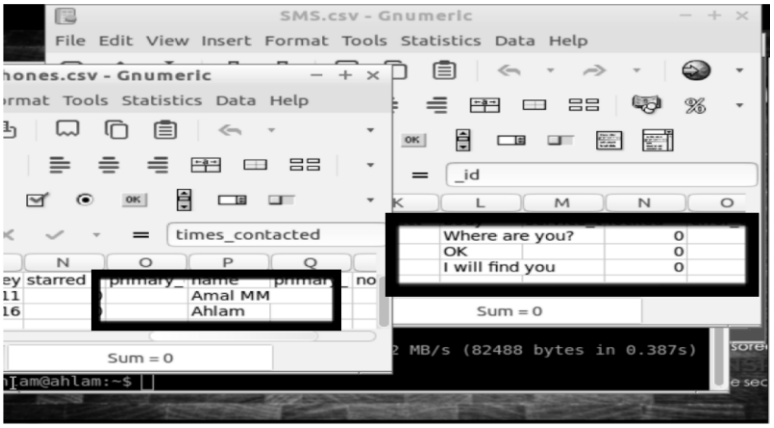

Figure 13 Some captured messages

By using Android device:

Enable USB debugging on Android device:

Settings > Applications > Development> check ‘USB debugging’

In VirtualBox:

We need to connect the device with Santoku VM

Devices > USB Devices > checkmark next to Android device

After that, in Android device, it will show a screen that has 2 options of USB PC connection: MTP and PTP. We must choose PTP because Mac laptop does not support MTP. After that we download AFLogical OSE tool in Android by using Terminal Emulator tool:

$ adb install AFLogical-OSE_1.5.2.apk

In Santoku VM:

Santoku > Device Forensics >AFLogical OSE then run the below command:

$ sudo adb devices

This command shows all connected devices, so we should see our Android device as:

List of devices attached

4d009941f57c24083 device

Then we run AFLogical OSE command:

$aflogical -ose

Figure 14 Pull all the data from Android

Now we ready to pull all the data from the device. It will show a screen that has options of what we want to extract such as Contacts Phones, MMS, and SMS messages. Then, all we need is choosing needed options then tap "Capture" icon. Figure 15 shows all data that we extracted and located in the file (Aflogical-data/forensics).

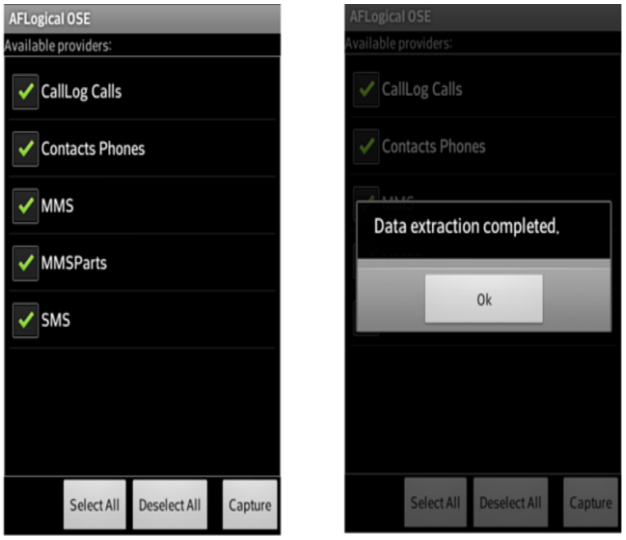

Figure 15 Data extraction completed

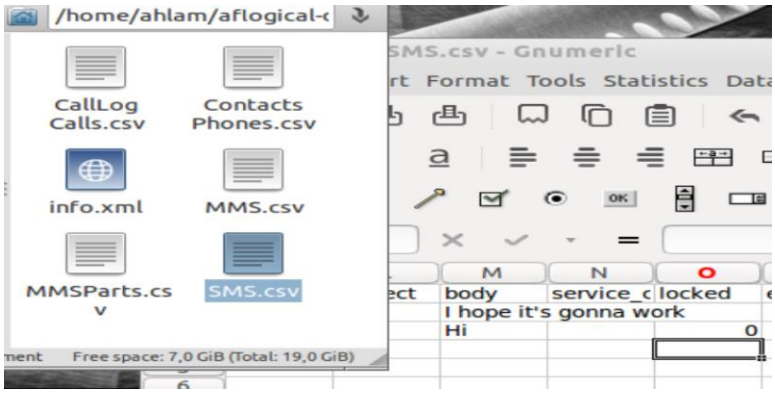

Figure 16 Captured messages

7.2. Reverse Engineering Tools

In cyber security domains, reverse engineering tools help ethical hackers to identify and understand the ways that an attacker uses to breach a system. Reverse engineering tools enable programmers or developers to incorporate new features into the source code. There are different purposes such as disassembler tools, debugger tools, and hex editor tools. Here, we implemented some reverse engineering tools to examine the functionality of some of Android mobile applications.

In order to implement the reverse engineering tools, we need first to install apk files. The apk (Application Package Kit) file is a zipped package that contains the compiled programming code (dex file), resources, certificates, manifest, and additional files. Any apk file can be unpacked and decompiled for analysis by using specific tools. We have to make sure that we install the apk files from trusted sources. After searching in the internet, we found two open sources that have most of the Android apk files: apkpure [40] and APKMirror [41]. In Figure 17, we downloaded six apk files we want to decode to perform our implementation.

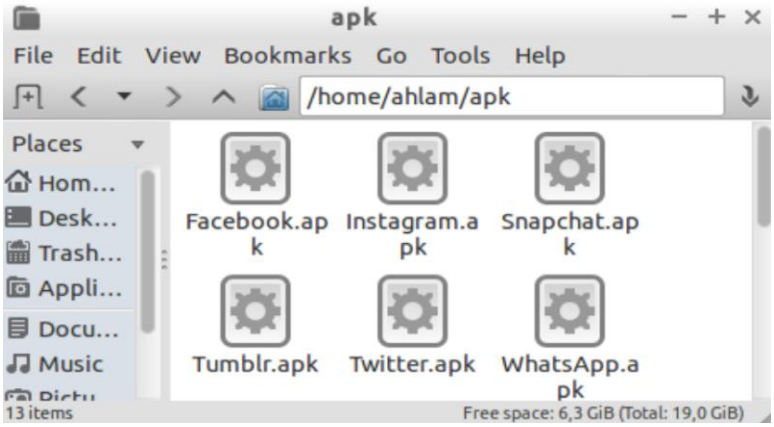

Figure 17 Apk files

7.2.1. Dex2jar Tool:

Dex2jar tool [30] is used to convert a program's code from the dex file (Dalvik Executable) to a jar/class file. After that, we could use any Java decompiler to view the source code of any Android application. First, after we install an apk file (such as Facebook.apk file), we run the following command:

```
$d2j-dex2jar Facebook.apk
```

Figure 18 Dex2jar commands

After that, a new file called **Facebook-dex2jar.jar** will be generated in the same location/folder. Figure 19 shows all the. dex apk files converted into .jar files.

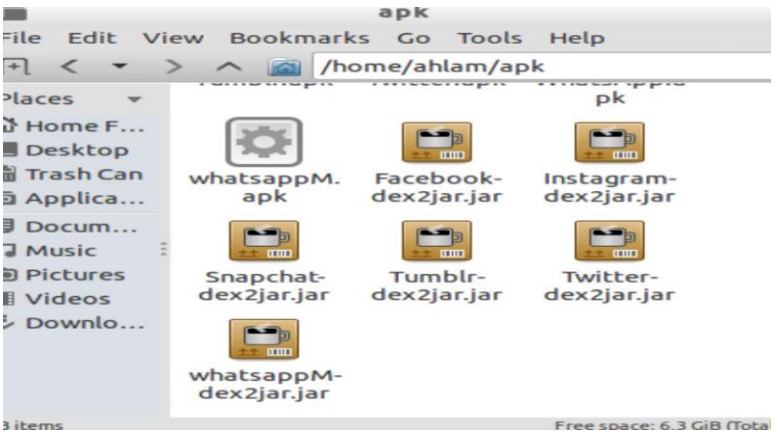

Figure 19 A new file called Facebook-dex2jar.jar will be generated in the same folder

Until this stage, we will not be able to access the java files as shown in Figure 20 below. So, we will use JD-GUI tool [46], a Java decompiler, that helps to access the source code of any jar files.

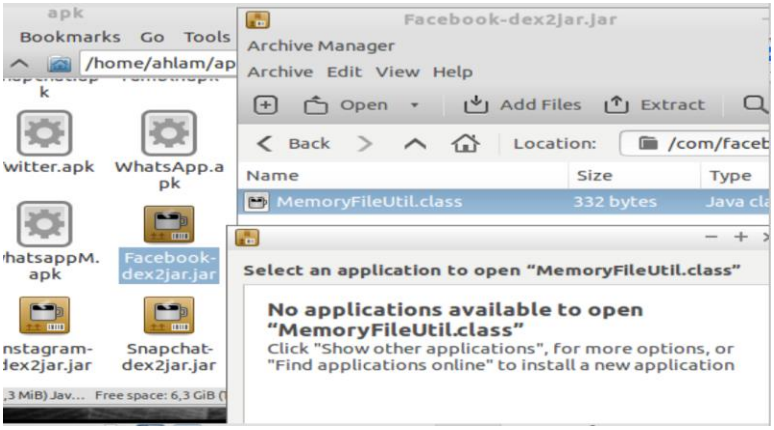

Figure 20 Unreadable file

7.2.2. JD-GUI Tool:

JD-GUI tool [46] is a Java decompiler used to decompile jar files into Java files. By using JD-GUI tool, we will be able to display and access all Java files of any apk. This tool is preinstalled in the Santoku Linux, so we only need to open the Facebook-dex2jar.jar file within JD-GUI tool. We first navigate to:

Santoku > Reverse Engineering> JD-GUI

The below window will pop up, then we could choose any jar file to decompile it.

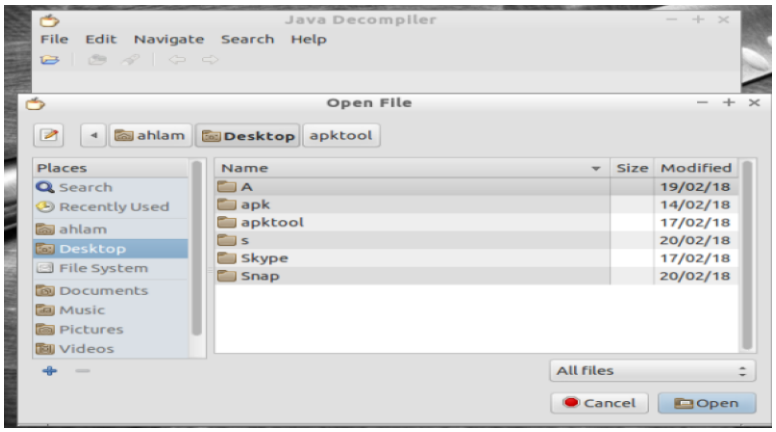

Figure 21 JD-GUI tool

Figure 22 shows that we successfully able to view the original code of Facebook application.

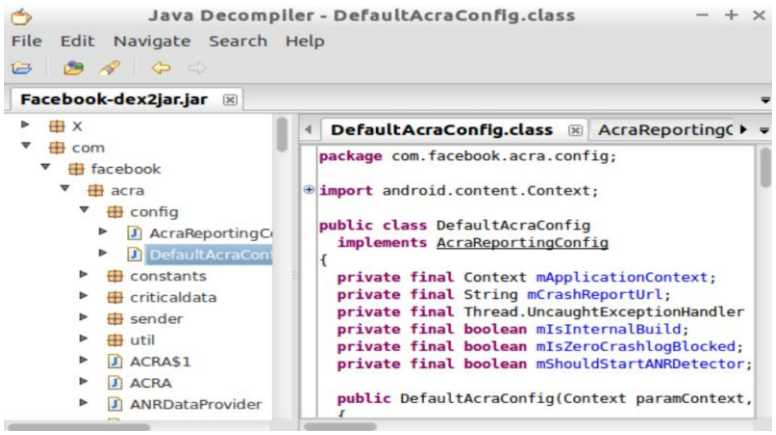

Figure 22 View the original code

At this point, we could view and modify all the applications that we installed and it is completely readable.

7.2.3. Apktool

According to the official website [39], Apktool is "A tool for reverse engineering 3rd party, closed, binary Android apps. It can decode resources to nearly original form and rebuild them after making some modifications; it makes possible to debug smali code step by step. Also, it makes working with an app easier because of project-like file structure and automation of some repetitive tasks like building apk, etc." We will use apktool in the next chapter: Android repackaging attack with Santoku VM.

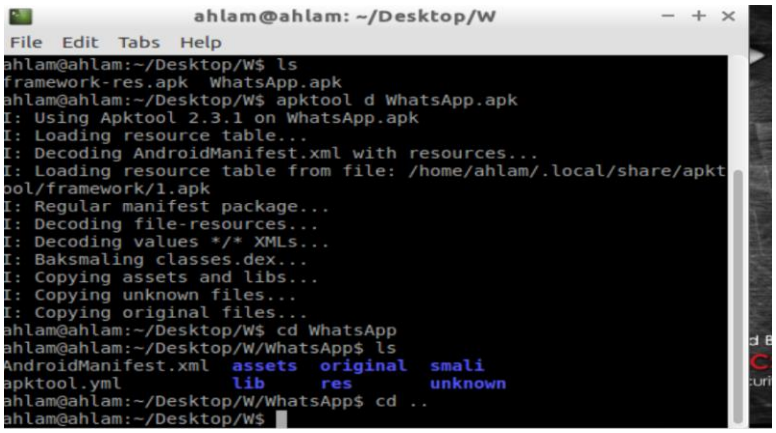

Figure 23 Apktool

7.3. Android Security Assessment Tools

7.3.1. Drozer

Drozer is a framework for Android security assessments developed by MWR Labs [38]. It helps to find security vulnerabilities in applications and devices. Basically, a pentester needs to run the commands on the workstation console and then Drozer sends them to the agent installed on the target device to execute the relevant task. Drozer has different 'modules' to perform different tasks such as retrieving package information.

Getting started with lab setup

- Santoku VM running within VirtualBox
- Emulator or Android device running 2.1 or higher
- Drozer tool is in Santoku by default, but we need to make sure that it is updated
- Install Drozer Agent apk from the official website
- Install any Android application in our emulator to execute the tool commands

Connecting the emulator and installing the Agent

First step we need to connect Santoku VM with our Android emulator. After that, we download Drozer Agent apk from the official website then install it in our emulator by using the below command.

$adb install drozer-agent.apk

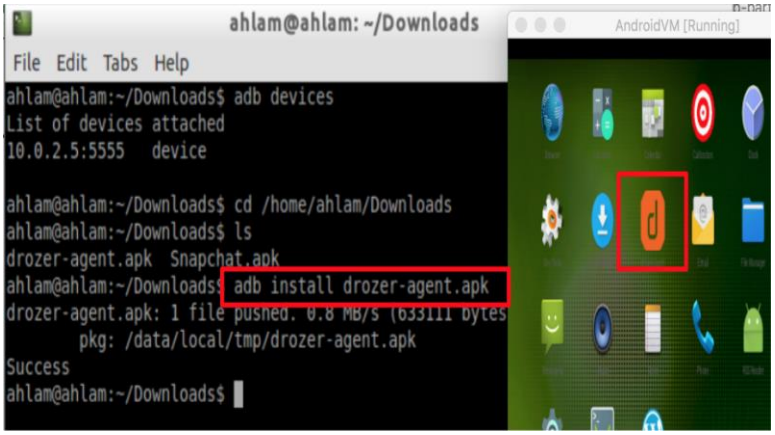

Figure 24 Install Drozer Agent

Starting a Session

Here, in order to connect Drozer console in Santoku to Drozer agent on the Android, we need to set up a port forward by using adb. By default, drozer uses port 31415.

$adb forward tcp:31415 tcp:31415

Now, we should launch the Agent by turning on the embedded server that listens on TCP port 31415. After that, we start Drozer console within Santoku terminal by using the command below:

$drozer console connect

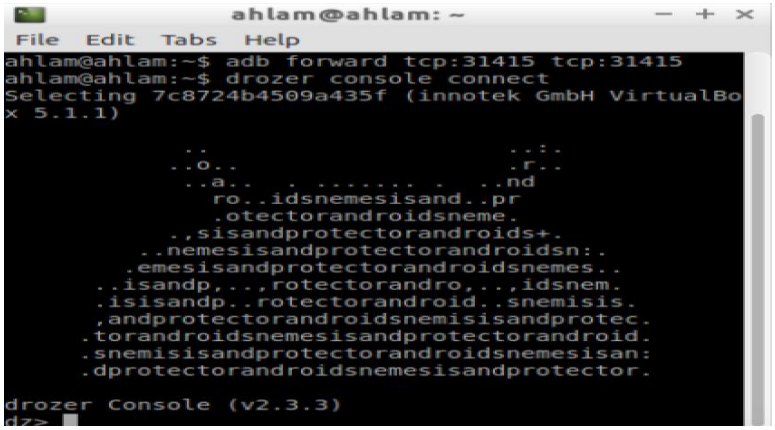

Figure 25 Start the console

At this point, we have a fully operating attack framework, and we can use many of the reconnaissance, scanning, and attack commands that drozer provides.

7.3.2. Performing Android Security Assessments

Listing to all the modules

Drozer tool could give us a list of all Drozer modules that can be executed in the current session by using the below command, Figure 26. And notice that any command began with '$' is running within the terminal of the target device, on the other hand, the command began with 'dz>' is running within Drozer console.

dz> list

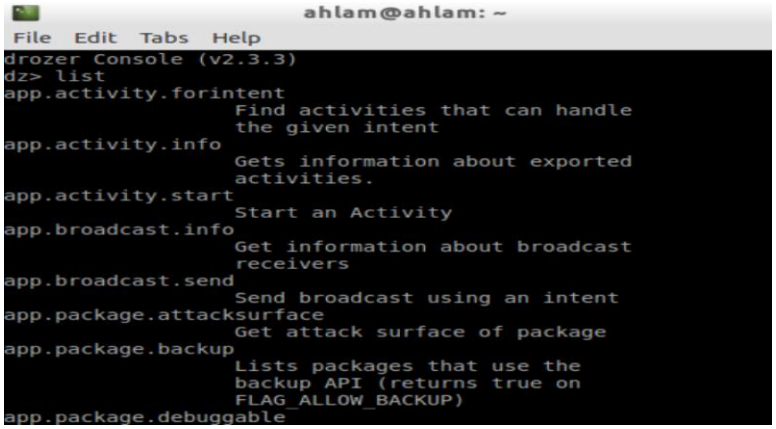

Figure 26 Listing to all the modules

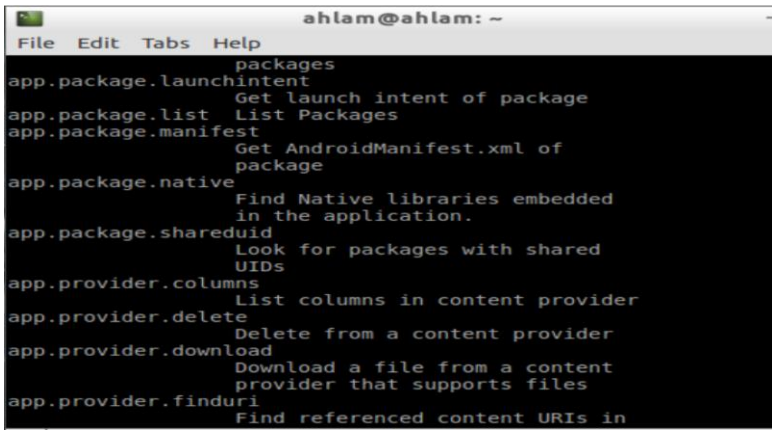

Figure 27 More modules

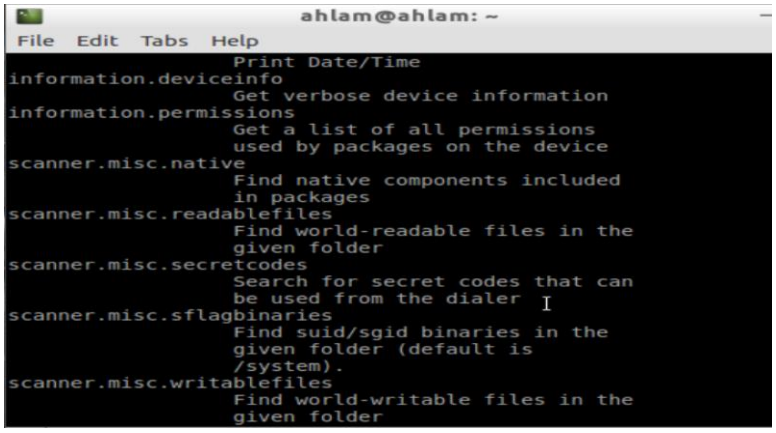

Figure 28 The output is truncated

Retrieving Package Information

Apps installed on an Android device are identified by their 'package name'. We can use the `app.package.list` command to get a list of all the packages installed on the target device, Figure 28.

dz> run app.package.list

Figure 29 A command to show a list of all packages

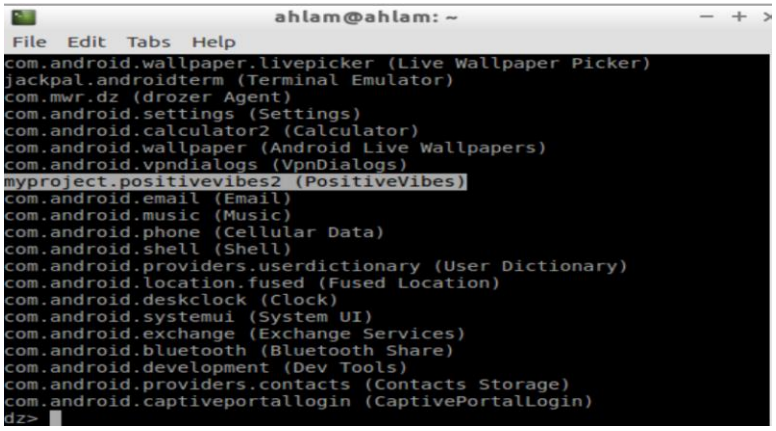

Figure 30 To find a specific package name

To find the package name of any app, we can use the flag "-f" with `app.package.list` followed by the name of target application, Figure 30. Here, we use our application, PositiveVibes,

dz> run app.package.list –f PositiveVibes

After executing the above command, we will have the package name. Now, we can ask drozer to provide some basic information about that package by running the below command:

dz> run app.package.info –a myproject.positivevibes2

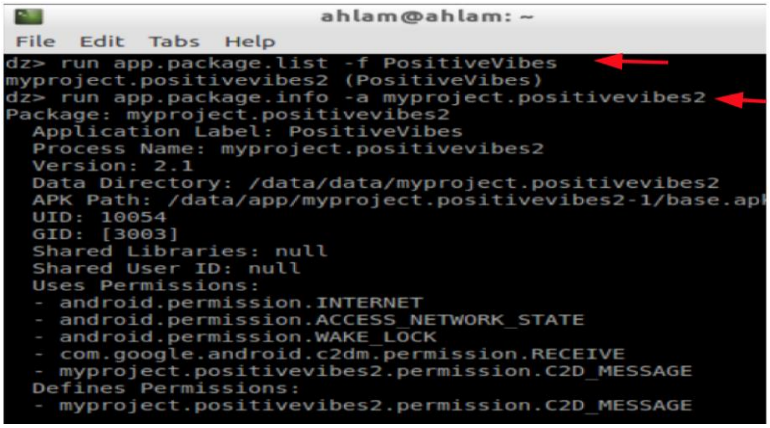

Figure 31 Shows details about PositiveVibes app

We can see a lot of information about the app. It shows details about PositiveVibes app, including the version, where the app keeps its data on the device, where it is installed and details about the app permissions.

Identify the Attack Surface

Drozer tool also helps to identify the attack surface of our target application. It gives a lot of details such as the number of exported applications components, Figure 31.

dz> run app.package.attacksurface myproject.positivevibes2

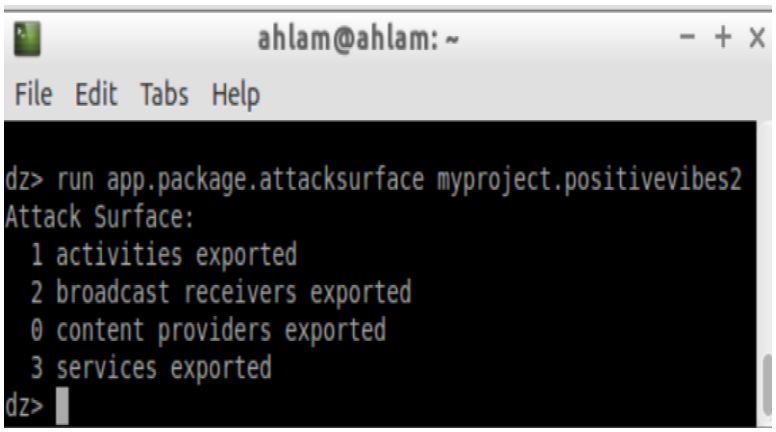

Figure 32 Identify the attack surface

This command gives account of number of activities exported, broadcast receivers, content providers and services that exported by the target app.

Launching Activities

Application activities are the components of the target device application that facilitate user interaction. All we need is to find any activity using “app.activity.info” module.

In Figure 33, we able to access Manifest file to the target application by using Santoku terminal.

dz> run app.package.mainfest myproject.positivevibes2

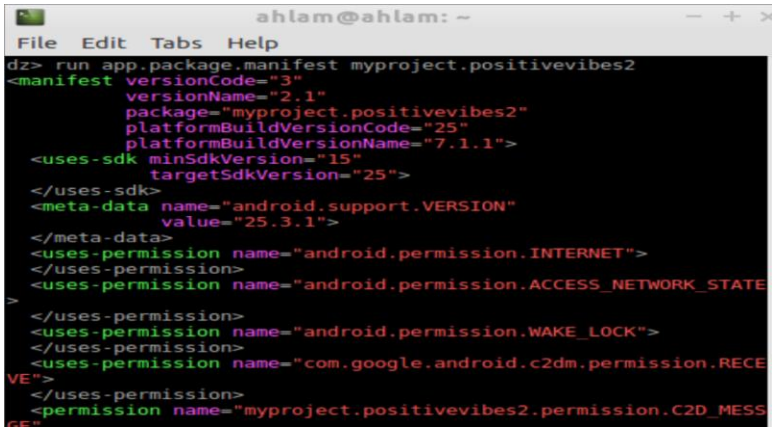

Figure 33 Shows the manifest file

Reading from Content Providers

Drozer allow to read from the content providers without permission. In Figure 34 shows core information about the target email by using the below command:

dz> run app.provider.info -a com.android.email

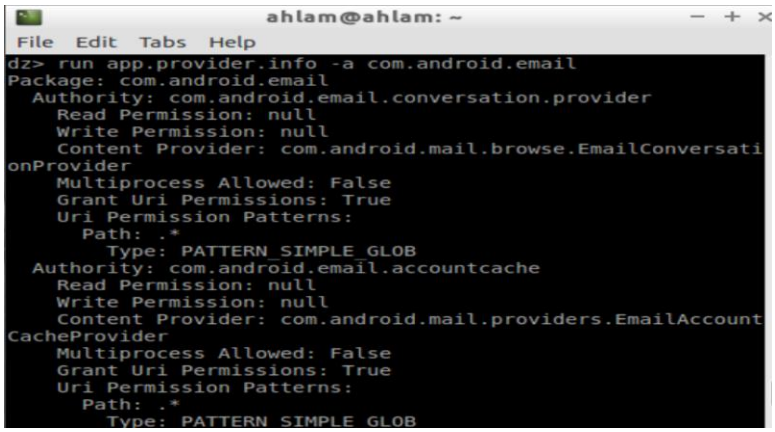

Figure 34 Reading from content provider

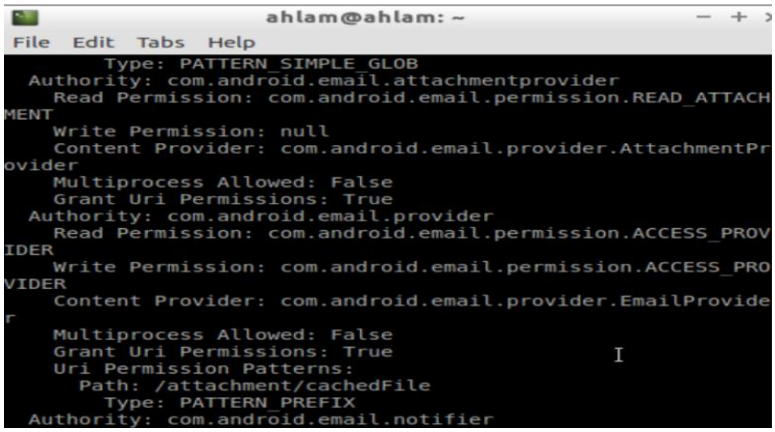

Figure 35 Reading from content provider

Database-backed Content Providers (Data Leakage)

Drozer use scanner module to gather specific information. In this module we need to guess the path of the needed component to get a list of URLs, Figure 36.

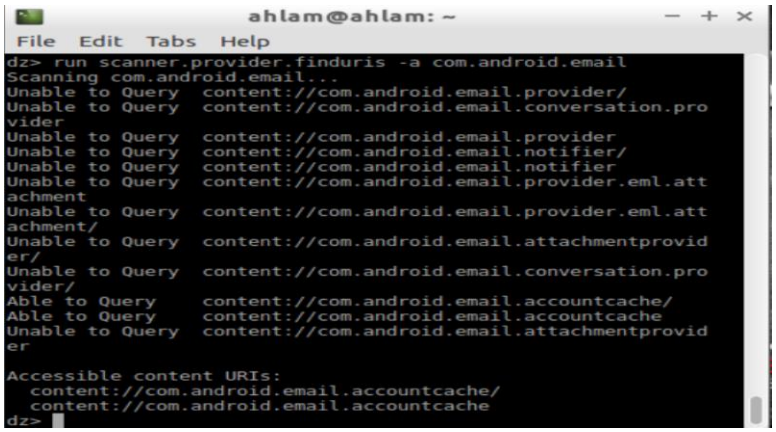

Figure 36 Database-backed content providers (data leakage)

Content Provider Vulnerabilities

Drozer provides modules to test if the content of the target device has vulnerabilities or not, Figure 37.

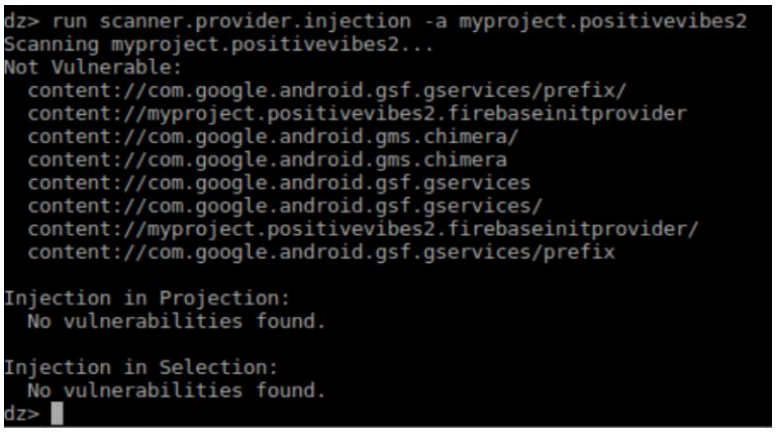

Figure 37 Content provider vulnerabilities

8. ANDROID REPACKAGING ATTACK BY USING SANTOKU LINUX

We performed Android repackaging attack as a simulation of how an attacker could hack Android applications. The attacker installs apk file from any apk website then modifies the apk wither by reverse engineer the app, inject a malicious file or only change some lines from the Jave code. After that, the attacker rebuilds the modified apk, signs it and finally publishes it to the public. Repackaging attack victims mostly cannot notice that they have been attacked because both apks, the original and the modified, are almost the same.

Android repackaging attack steps:

Pull framework file

Basically, framework-res.apk file has the code and resources of the Graphical User Interface-GUI of each phone. Apk file includes manifest file which index all the resources (images, sounds etc) and other important files.

In our case, framework-res.apk file helps to perform Apktool for decoding and building apk files. We could find Framework-res.apk file in *System/Framework/Framework-res.apk.* To pull a framework-res.apk, we need to connect our Android emulator with Santoku VM, then pull a framework file via the below command. After that, we need to install it via apktool. The file will be in *home/../.local/share/apktool/framework/1.apk.*

$ adb connect 10.0.2.5

$adb pull /system/framework/framework-res.apk

$ apktool if framework-res.apk

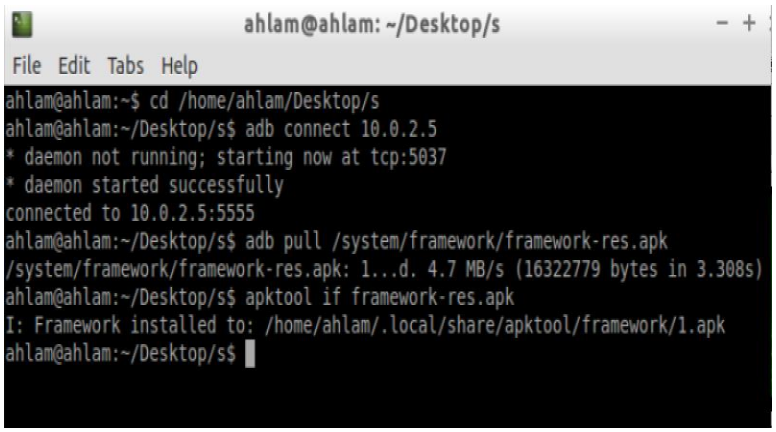

Figure 38 Connect our Android emulator with Santoku

Install and decode apk file

We could install apk files from a trusted third parity website such as Apkpure. Here, we installed (skype.apk) then decode it by using Apktool below command:

$ apktool d -r skype.apk

Since we do not use resources file, we could use *-r* within the command line.

"***-r,*** this will prevent the decompile of resources. This keeps the resources.arsc intact without any decode. If only editing Java (smali) then this is the recommended action for faster decompile & rebuild" [39].

After decoding the apk file, we will get a new folder with same name (skype folder). This folder has the following files:

AndroidManifest.xml, assets, original, resources.arsc, smali, apktool.yml, unknown, lib, and res.

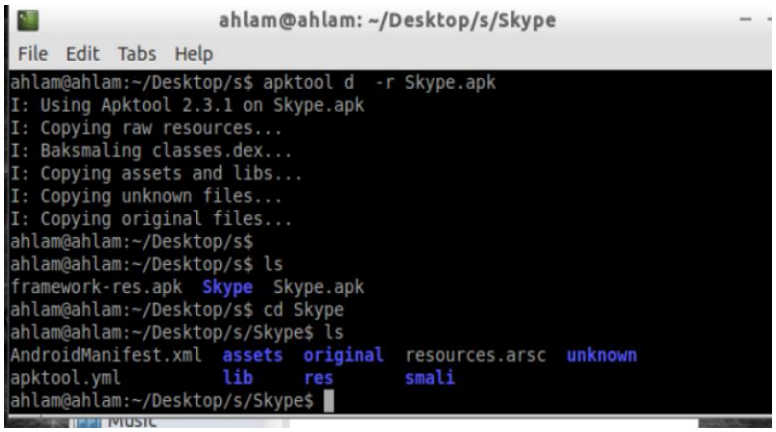

Figure 39 Decode the file

Inject Malicious Code

We could either modify some existing smali file by adding the malicious code or create a new file with the malicious code then place it in the *smali/com* folder. Here, we choose the second option.

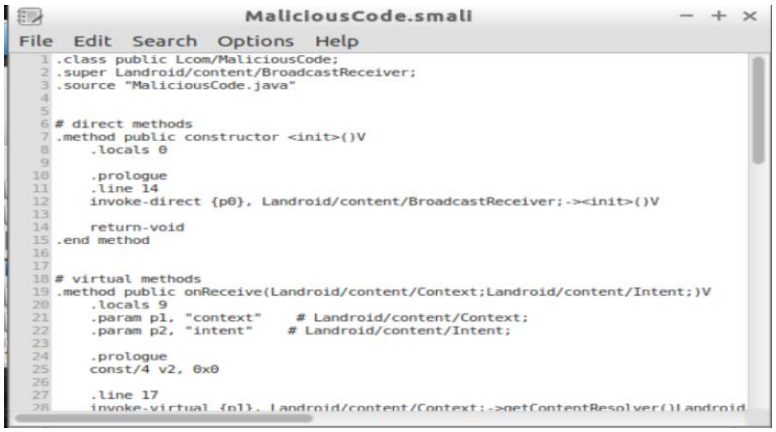

Figure 40 Malicious code [42]

Now, we need to add permissions which is to read/write contacts and receive a boot from the AndroidManifest.xml file. These permissions allow to tell the system when to invoke our broadcast receiver.

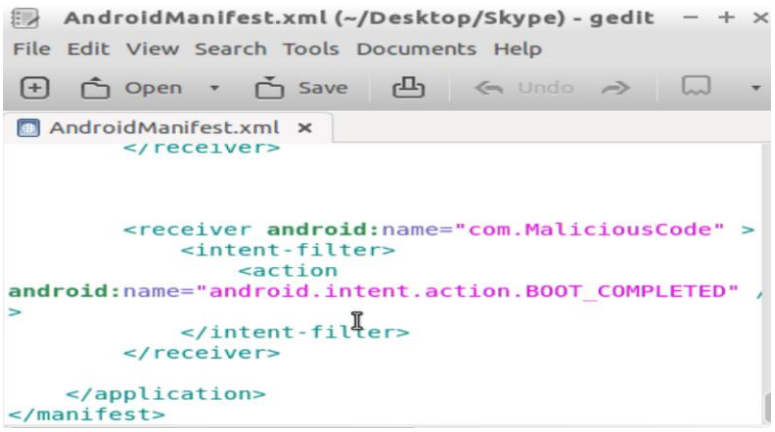

Figure 41 Adding permissions [42]

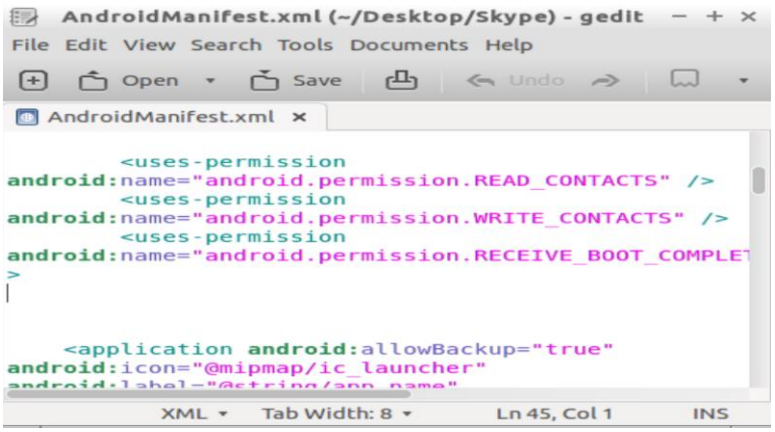

Figure 42 Adding permissions [42]

Repackaging the new apk

Here, we reassemble all the files then create a new apk file that has our malicious code. After this step, we will get two new files: build and dist files. We could find the modded app in the dist file located inside the original folder.

$ apktool b -f Skype

"-f, Overwrites existing files during build, reassembling the resources.arsc file and dex file(s)" [39].

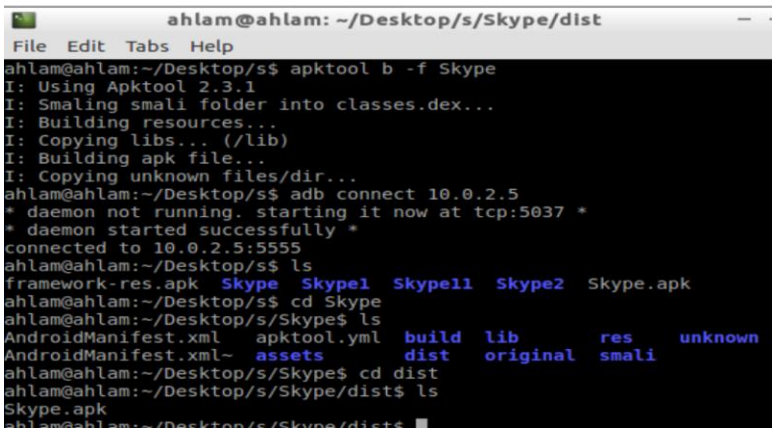

Figure 43 Rebuild the apk

Sign the apk file:

Here, after rebuilding the apk file, we need to sign our apk by using Keytool and Jarsigner:

Generate a public and private key pair (RSA key) by using the Keytool command:

$ keytool -alias am -genkey -v -keystore my-release-key.keystore-keyalg RSA -keysize 2048 -validity 10000

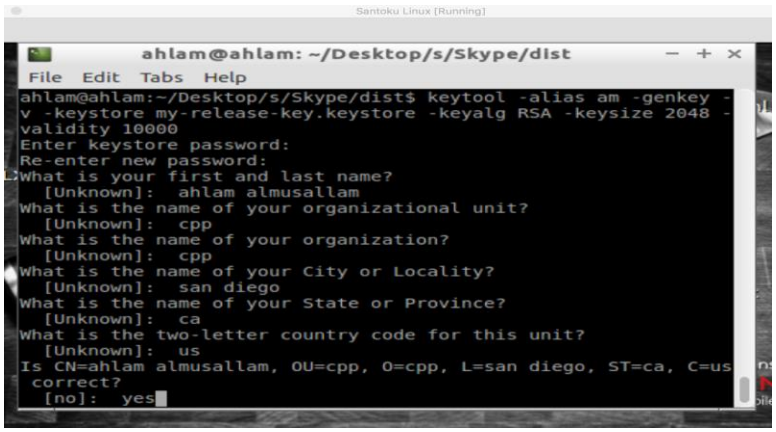

Figure 44 Create the key

Keytool options:

-genkey: Asks Keytool to generate a key

-v: To enables verbose

-keystore: Find a location to store the key

-alias: An alias for the key pair

-keyalg: To generate an encryption algorithm, RSA or DSA.

-keysize: The length of the key, we choose 2048

-validity: The valid number of days for the key, we choose 10,000 days

Jarsigner to sign the apk file by using the key generated in the previous step:

$ jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1-keystore my-release-key.keystore Skype.apk am

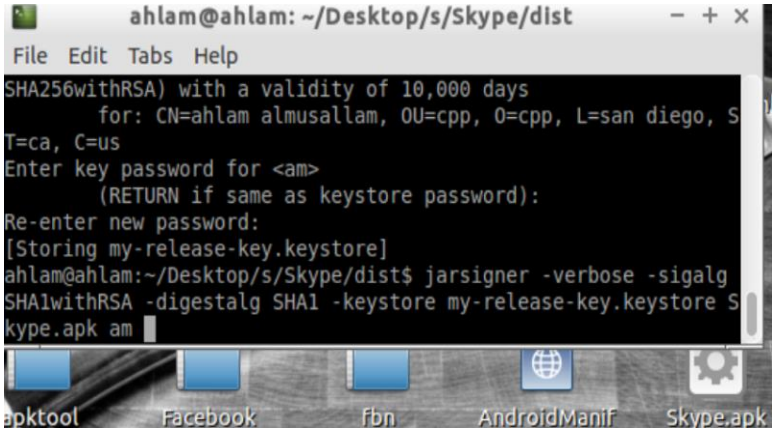

Figure 45 Sign the apk

To demonstrate the attack, we need to create contact information in the Android emulator:

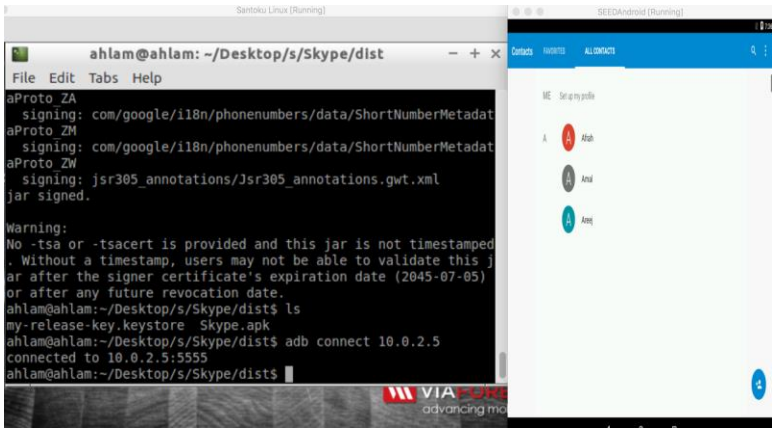

Figure 46 Create info contacts

Then, we install the infected apk in Android emulator by using the below command:

$ adb install Skype.apk

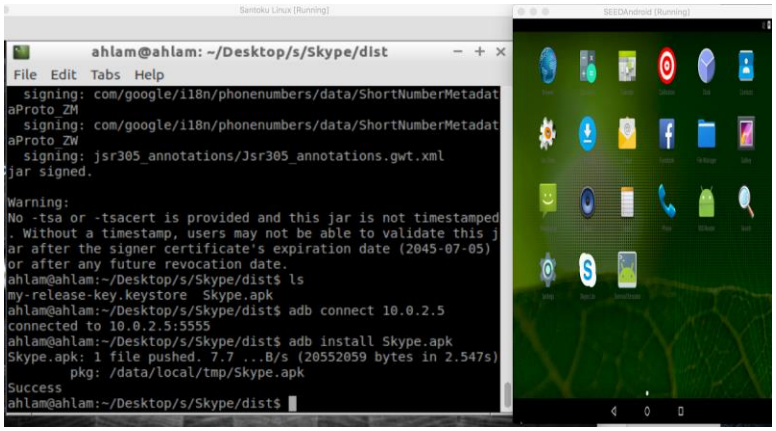

Figure 47 Install Skype in Android

After installing the app, we restart Android emulator; and as a result, all the contact information that we just created will be deleted.

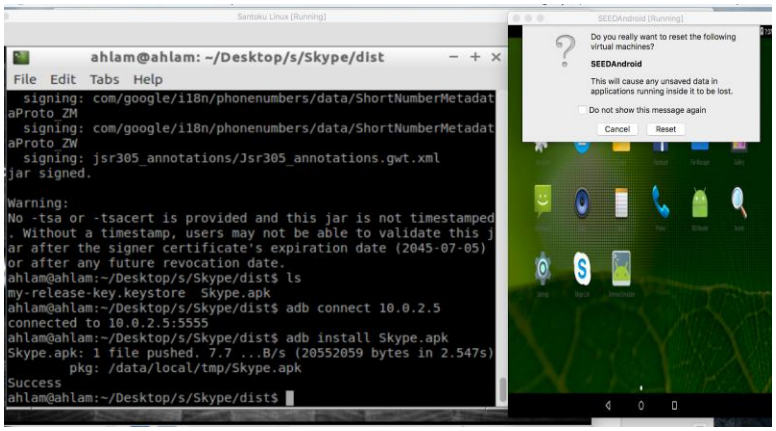

Figure 48 Restart Android

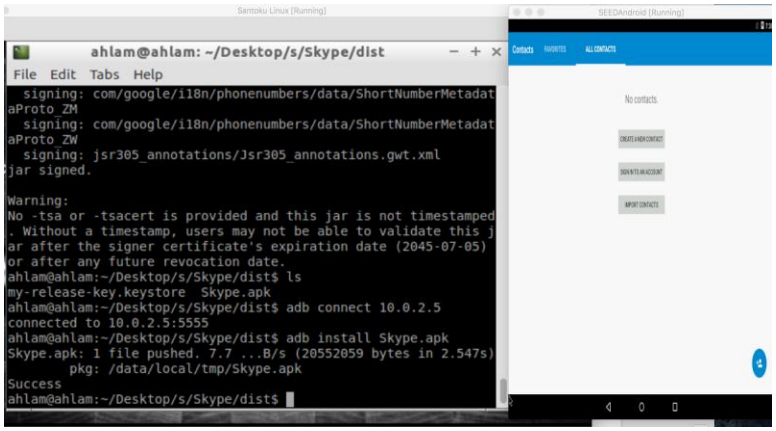

Figure 49 All the contacts are deleted

We have performed the same with the **WhatsApp** application:

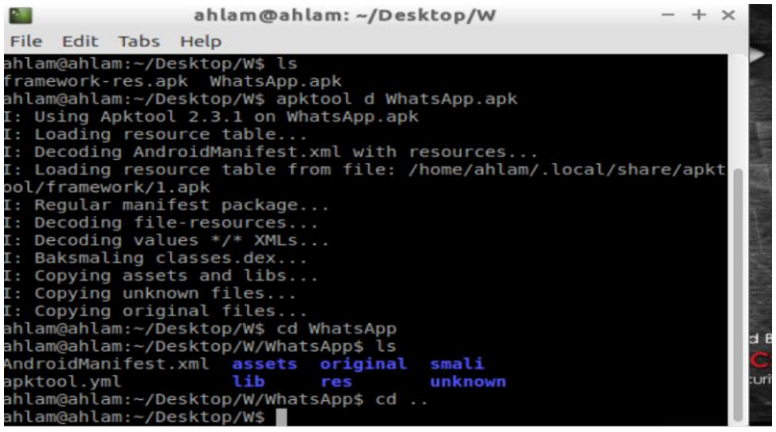

Figure 50 Decode WhattsApp apk

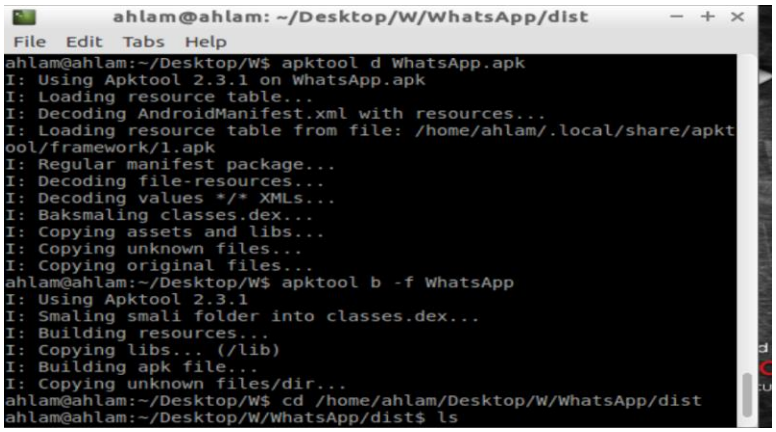

Figure 51 Rebuild apk

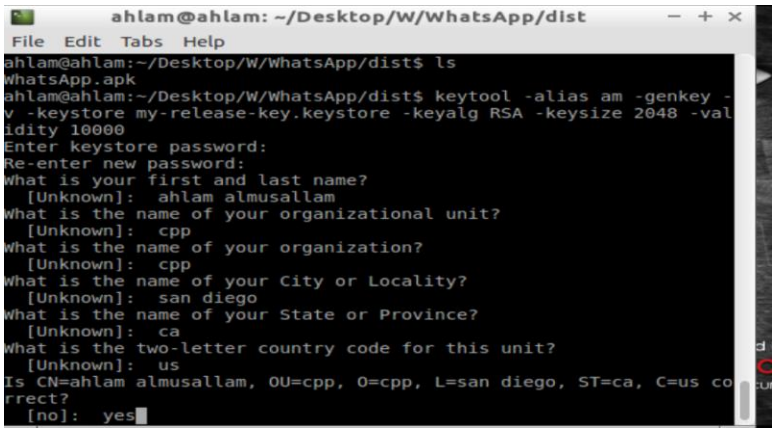

Figure 52 Create key

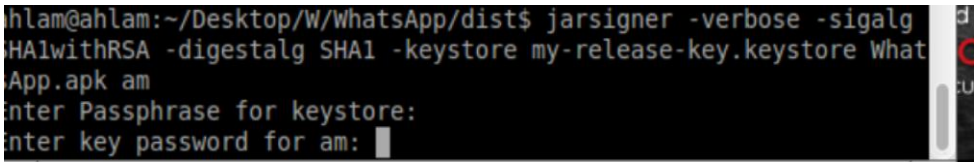

Figure 53 Sign it

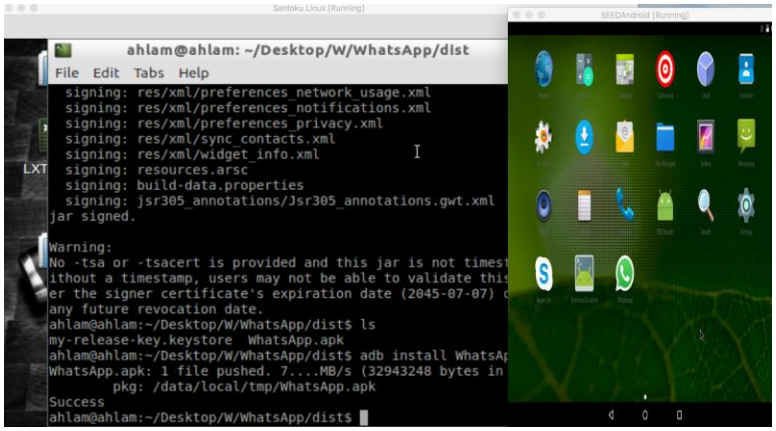

Figure 54 Install WhatsApp

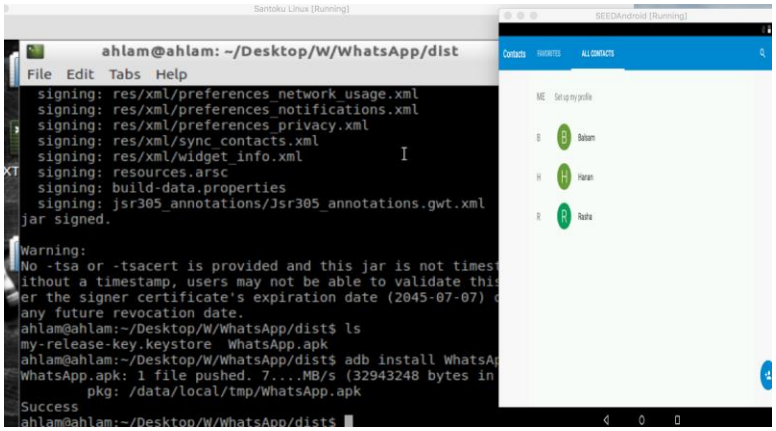

Figure 55 Create contacts

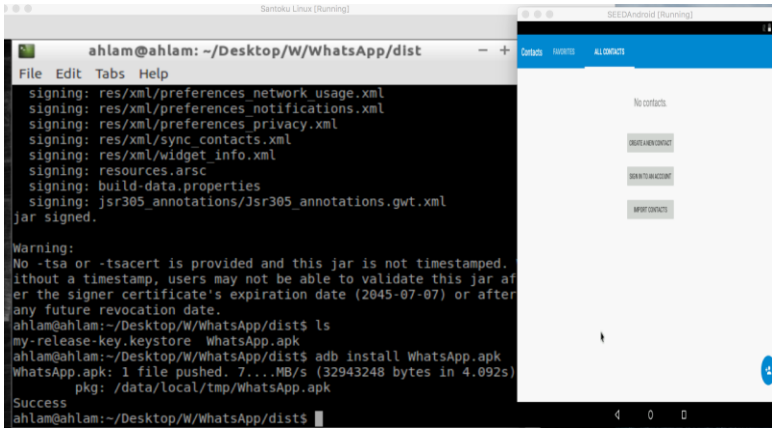

Figure 56 All the contact deleted

Also, we have performed the same with the **Angry Birds** game:

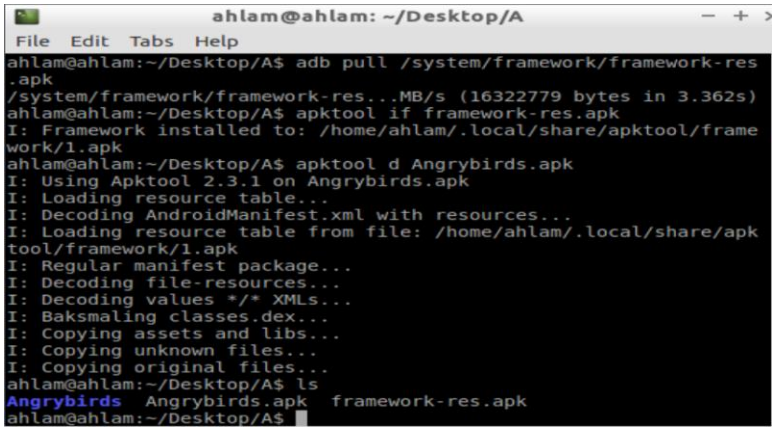

Figure 57 Decoding apk

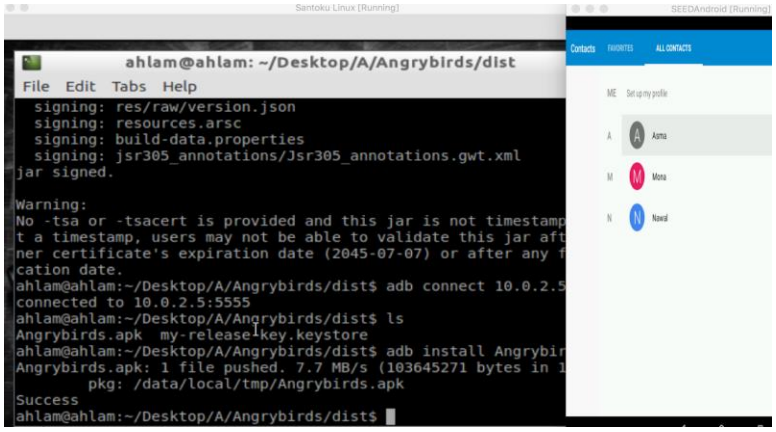

Figure 58 Create contacts

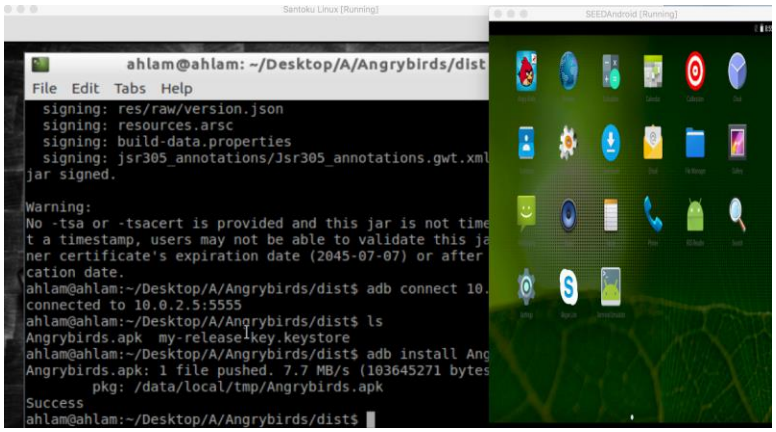

Figure 59 Installing AngryBird

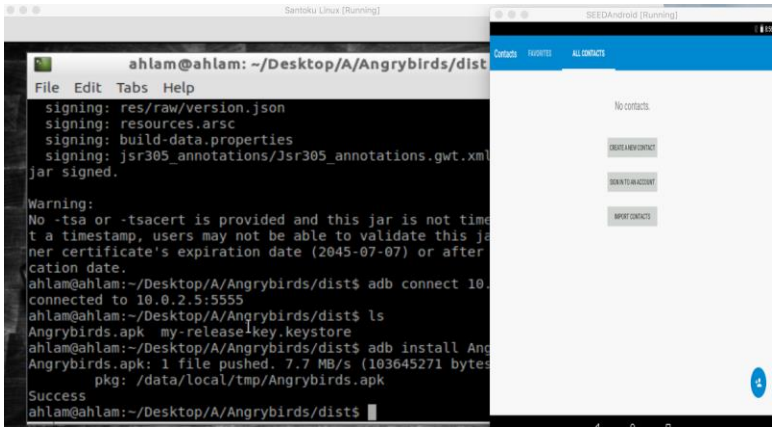

Figure 60 All the contacts are deleted

9. CONCLUSION AND FUTURE WORK

Although developers care about security, not all of them have enough knowledge to diagnose or address security issues. It may be impossible to create a fully secure environment, but there are some ways that help to identify threats and prevent data disclosure. Mobile penetration testing helps to address security weaknesses or vulnerabilities that exist within the mobile infrastructure. The success of any pentest depends on a proper selection of security tools. In this paper, AFLogical, Dex2jar, JD-GUI, Apktool and Drozer tools were introduced and then examined. The selection of these tools was based on its versatility, usability and effectiveness. This guide also discussed detailed procedures for Android repackaging attack by using Santoku Linux distro. In the future, more effort should be spent in mobile application security from researchers and developers. This paper can be extended in different directions: first, Android mobile forensics to extract data from the Android platform by using forensic tools that recovering the necessary digital evidences. Second, the focus of this paper is only on the Android platform, so efforts can be made to other platforms such as iOS or BlackBerry. Finally, in case of repackaging attack, hackers publish their vulnerable mobile applications by using the third-party app sites, so a study needed to discover vulnerabilities found in the app before installation.

10. REFERENCES

[1] Statista, “Annual number of mobile app downloads worldwide 2022 | Statistic,” 2018, *Statista*. [Online]. Available: https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobileapp-store-downloads/. [Accessed: 07-Jun-2018].

[2] A. Gupta, and E. Shapira, *Learning pentesting for Android devices: a practical guide to learning penetration testing for Android devices and applications*. Birmingham, UK: Packt Publishing, 2014.

[3] N. Kumar, and M. Haq, *Penetration testing for Android Smartphones*, master's thesis, Chalmers University of Technology, Goteborg: SW, 2011.

[4] Guarda, Teresa, et al. “Penetration Testing on Virtual Environments.” *Proceedings of the 4th International Conference on Information and Network Security - ICINS 16*, 2016, doi:10.1145/3026724.3026728.

[5] Denis, Matthew, et al. “Penetration testing: Concepts, attack methods, and defense strategies.” *2016 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, 2016, doi:10.1109/lisat.2016.7494156.

[6] Koopari Roopkumar and Bharath Kumar, "Ethical Hacking Using Penetration Testing" (2014). *LSU Master's Theses.* 3238.
http://digitalcommons.lsu.edu/gradschool_theses/3238

[7] Dawson, Joel, and J. Todd Mcdonald. “Improving Penetration Testing Methodologies for Security-Based Risk Assessment.” *2016 Cybersecurity Symposium (CYBERSEC)*, 2016, doi:10.1109/cybersec.2016.016.

[8] Peake, Chris (2003, July 16). Red Teaming: The Art of Ethical Hacking. Retrieved from http://www.sans.org/reading_room/whitepapers/auditing/red-teaming-art ethicalhacking_1272

[9] K. Makan, *Android security cookbook*. Birmingham, Mumbai: Shroff Publishers & Distr, 2014.

[10] Santoku-Linux. Retrieved from https://santoku-linux.com/.

[11] “Android version history,” *Wikipedia*, 05-Jun-2018. [Online]. Available: https://en.wikipedia.org/wiki/Android_version_history. [Accessed: 07-Jun-2018].

[12] SQLite. Retrieved from https://www.sqlite.org/

[13] WebView. Retrieved from https://developer.android.com/reference/android/webkit/WebView.html

[14] ARM. Retrieved from https://www.arm.com/

[15] Kingo Root. Retrieved from https://www.kingoapp.com/root-tutorials/how-to-enable-usb-debugging-mode-on-android
.htm

[16] Kali Linux. Retrieved from https://www.kali.org/

[17] Windows operating system. Retrieved from https://www.microsoft.com/en-us/windows/

[18] VMware. Retrieved from https://www.vmware.com/

[19] VirtualBox. Retrieved from https://www.virtualbox.org/wiki/Downloads

[20] Pentoo. Retrieved from http://www.pentoo.ch/

[21] BackBox. Retrieved from https://backbox.org/

[22] Parrot Security OS. Retrieved from https://www.parrotsec.org/

[23] NodeZero. Retrieved from https://sourceforge.net/projects/nodezero/

[24] AFLogical. Retrieved from http://aflogical-ose.apk.black/

[25] Burp Suite. Retrieved from https://portswigger.net/burp

[26] Ettercap. Retrieved from https://ettercap.github.io/ettercap/

[27] Nmap. Retrieved from https://nmap.org/

[28] Jasmine. Retrieved from https://jasmine.github.io/

[29] Smali. Retrieved from https://github.com/JesusFreke/smali/wiki/SmaliBaksmali2.2

[30] Dex2jar. Retrieved from https://sourceforge.net/projects/dex2jar/

[31] Wireshark. Retrieved from https://www.wireshark.org/

[32] Tcpdump. Retrieved from http://www.tcpdump.org/

[33] Genymotion. Retrieved from https://www.genymotion.com/

[34] MakeUseOf. Retrieved from http://www.makeuseof.com/tag/what-is-usb-debugging-mode-on-android-makeuseof-explains/

[35] SuperSU. Retrieved from

https://play.google.com/store/apps/details?id=eu.chainfire.supersu&hl=en

[36] Terminal Emulator. Retrieved from
https://play.google.com/store/apps/details?id=jackpal.androidterm&hl=en

[37] Nessus. Retrieved from

https://www.tenable.com/products/nessus/nessus-professional

[38] "Drozer," *MWR Labs*. [Online]. Available:
https://labs.mwrinfosecurity.com/tools/drozer/. [Accessed: 07-Jun-2018].

[39] Apktool. Retrieved from https://ibotpeaches.github.io/Apktool/

[40] Apkpure. Retrieved from https://apkpure.com/

[41] APKMirror. Retrieved from https://www.apkmirror.com/

[42] "Android Repackaging Attack Lab," *SEED Project*. [Online]. Available:
http://www.cis.syr.edu/~wedu/seed/Labs_16.04/Mobile/Android_Repackaging/.
[Accessed: 07-Jun-2018].

[43] Android App Development Tutorial. (n.d.). Retrieved from http://android-app-tutorial.blogspot.com/2012/08/architecture-system-application-stack.html#.WxenJPZFzRM

[44] Android Studio. Retrieved from https://developer.android.com/studio/install

[45] Android SDK Manager. Retrieved from
https://stuff.mit.edu/afs/sipb/project/android/docs/tools/help/sdk-manager.html

[46] JD-GUI tool http://jd.benow.ca/