**Index**

Contents

[1. UNZIP, WINRAR 2](#_Toc470272393)

[2. Apk Tool 2](#_Toc470272394)

[3. Mobilizer 5](#_Toc470272395)

[4. Dex2jar and JDGUI 8](#_Toc470272396)

[5. Drozer 11](#_Toc470272397)

[6.SQLite Database Browser 19](#_Toc470272398)

[7. CatLog App 21](#_Toc470272399)

[8. Class Name DE obfuscator 21](#_Toc470272400)

[9. Xposed Framework – 23](#_Toc470272401)

[10. Mobile-Security-Framework-MobSF 26](#_Toc470272402)

[11. Configuring an Android Device to Work With Burp 27](#_Toc470272403)

[12. Installing Burp's CA Certificate in an Android Device 31](#_Toc470272404)

**Tools Used for Android Mobile Testing**

This Document contains step by step guidance. On How to perform static analysis by using tools:

# 1. UNZIP, WINRAR

In order to sees the files inside the given apk we need to

1. Change its extension to .zip or .rar
2. Then extract it

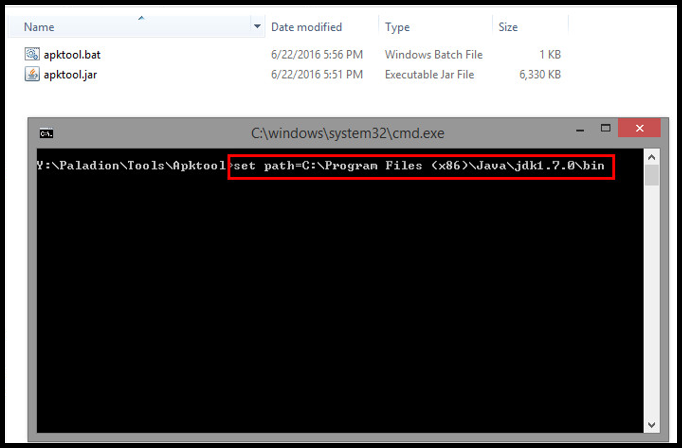
Note: We have not yet decompiled it. This step is important for checking the classes.dex file inside the extracted apk

# 2. Apk Tool

**Step1:**

Set java path(if not set)using following command:

**Set path=<PATH TO JAVA>**

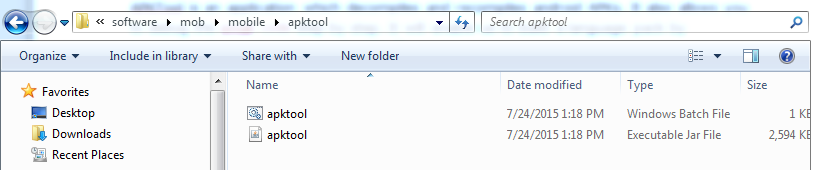
****

**Step2:**

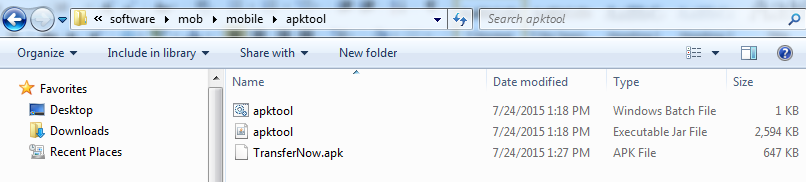
[Use APKTool to Decompile an APK](http://www.miui-au.com/add-ons/apktool/)

APKTool is an application which decompiles android APKs. It also allows you to debug the [smali code](http://code.google.com/p/android-apktool/)step by step. It will enable us to build a language pack by translating the .xml strings inside APK files.

1. Download the prepackaged APKTool zip.
2. Extract them to a directory.

****

Copy the apk file to the apktool folder.

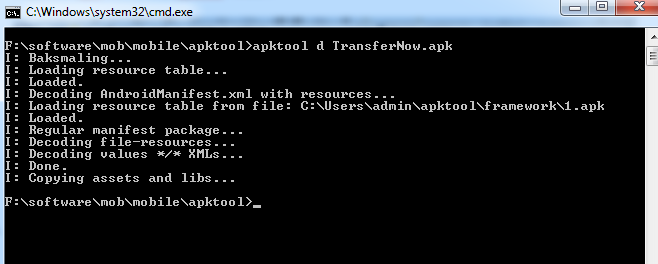
****

**Step3:**

Open command prompt and change the working directory to the apktool installed folder.

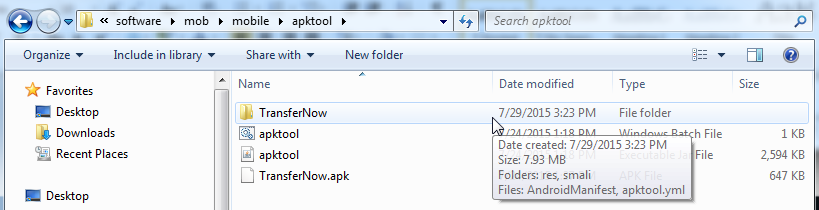
Eg. “F:\software\mob\mobile\apktool”

Run command (for example consider TransferNow.apk) “apktool d TransferNow.apk”

****

**Step4:**

A new folder will be created by the same name of the apk file (here example TransferNow.apk)

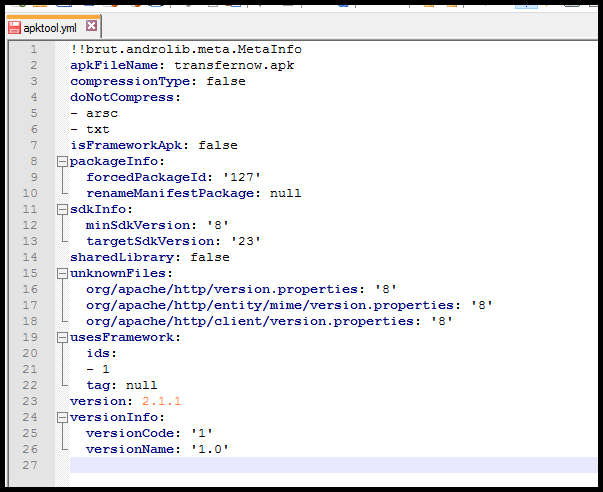
****

**Step5:**

This apk folder contains AndroidManifest.xml file, apktool.yml file and smali folder. This is decompiled form of apk.

**Step 6:**

Open apktool.yml with notepad.



Identify "minSdkVersion" on apktool.yml, the value be set over than 17a

Owasp Catorgy - M5 (Poor Authorization and Authentication)

**Step 7:**

Open AndroidManifest.xml with notepad.



Here we check for :-

* 1. Permissions
  2. Android exported components [Owasp Category – M8. Security Decisions Via untrusted Inputs]
  3. "android:allowBackup" attribute which should be set to "false" [Owasp Category – M2. Insecure Data Storage]
  4. "android:debuggable" attribute which should be set to "false" [Owasp Category – M10. Lack of Binary Protections]

# 3. Mobilizer

Mobilizer tool is used to find the sensitive data in the apk file like password, emailids etc.

**Step1:**

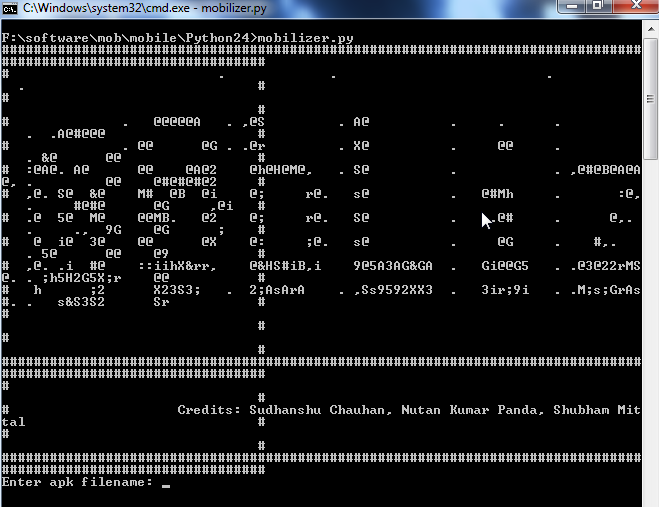
Open command prompt and set the path to the python installed folder and copy the apk file inside the python folder.

Eg: “**F:\software\mob\mobile\Python24**”



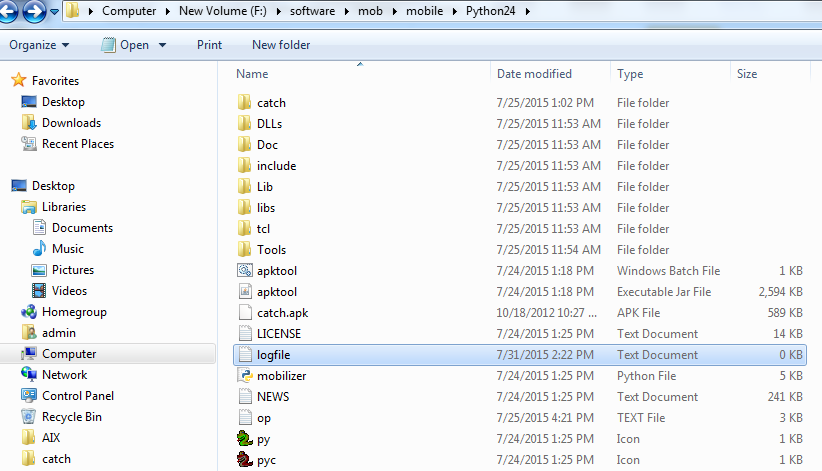
**Step2:**

Run mobilizer tool by running command “ **mobilizer.py** ” in command prompt.



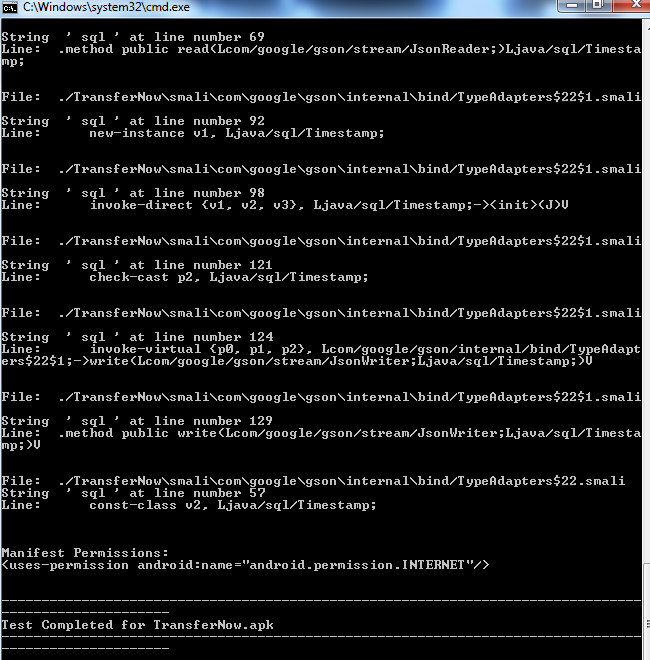
**Step3:**

A log file will be created in the python folder.



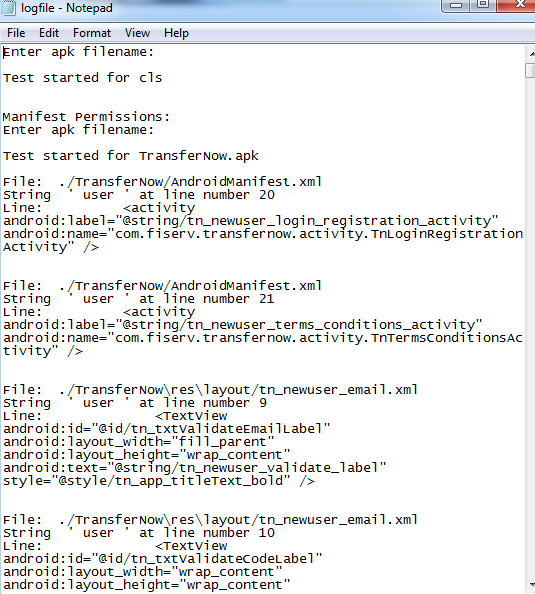
**Step4:**

Enter the apk file name Eg: “**TransferNow.apk**”



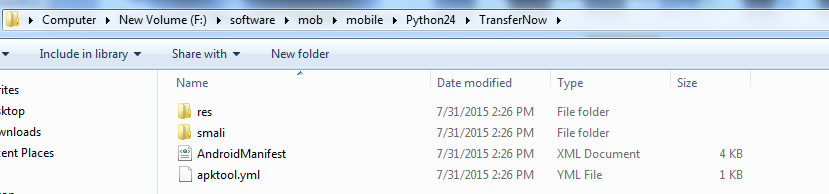
**Step5:**

All data will append to log file. Now, check the sensitive data inside the log file.



**Step6:**

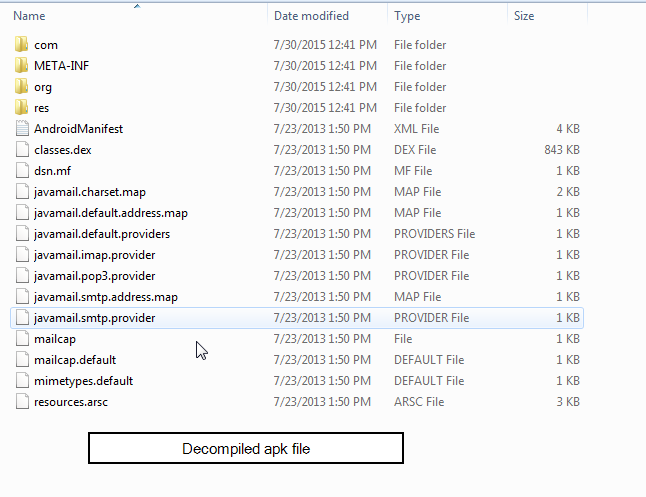
A folder of the same name as that of apk will be created containing AndroidManifest.xml file.



# 4. Dex2jar and JDGUI

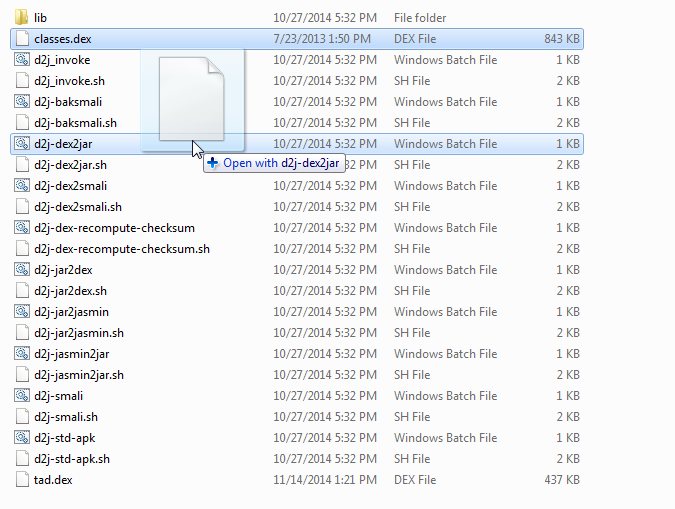
**Step 1:**

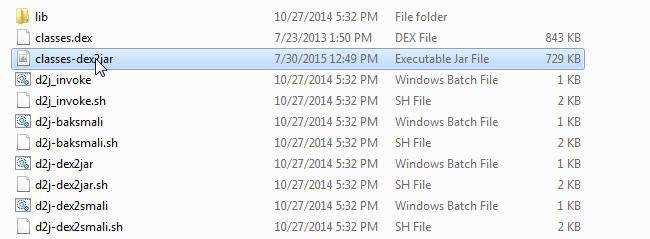
Covert the .apk file into .zip file then Extract the zipped file and under extracted folder we find classes.dex file along with other files



**Step 2:**

The dex files are Dalvik executable files format, and are not human readable. So, we need to convert it back to some human eye friendly language. We will use a tool called dex2jar which can be used to convert .dex to JAR files

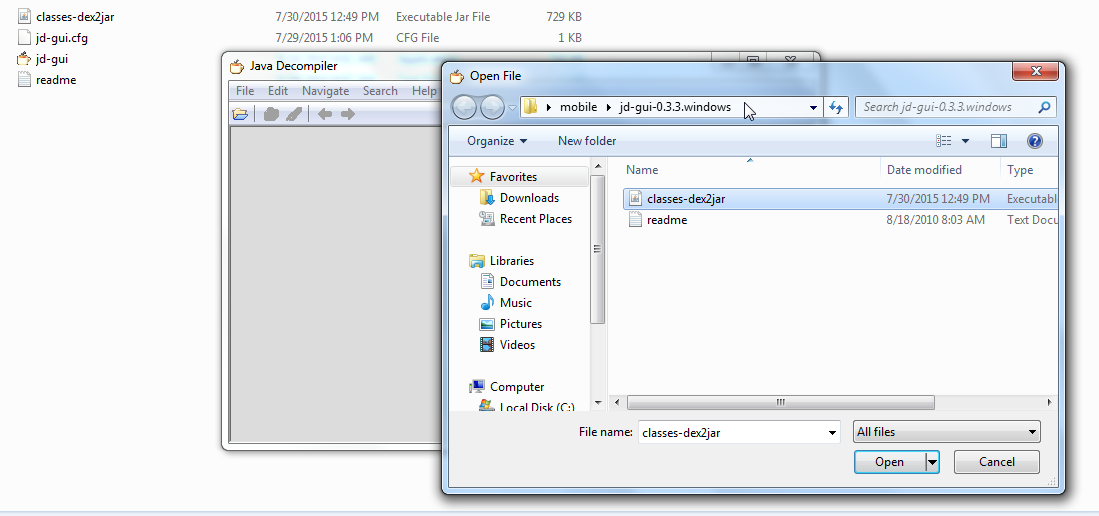
For that we just DRAG the classes.dex file and DROP on **d2j-dex2jar.bat file**. It creates a jar file for the corresponding app as shown below.  We get a executable jar file as shown.

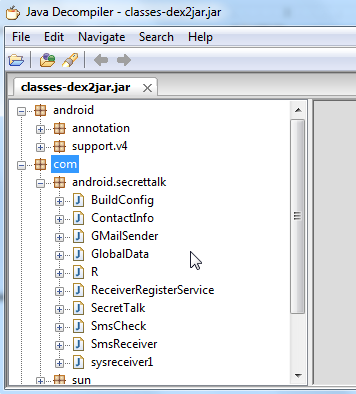


**Step 3:**

Now inorder to open the classes.dex2jar file we need a tool called **jdgui.**

For that we just open the tool and add the classes.dex2jar file in it.

****



Here we can check for –

* + 1. Source obfuscation of Android code. [Owasp Category – M10. Lack of Binary Protections]
    2. printStackTrace in Application Source Code.
    3. Hard-coded credentials on source code. [Owasp Category – M2. Insecure Data Storage]
    4. Insecure/deprecated cryptographic algorithms (RC4, MD5, SHA1) on source code. [Owasp Category – M6. Broken Cryptography]
    5. Identify hardcoded key in application or Keys may be intercepted via Binary attacks. [Owasp Category – M6. Broken Cryptography]
    6. Identify use of custom encryption protocol. [Owasp Category – M6. Broken Cryptography]
    7. checkServerTrusted with nobody [Owasp Category M3. Insufficient Transport Layer Protection]
    8. Value of AllowAllHostnameVerifier [Owasp Category M3. Insufficient Transport Layer Protection]
    9. Identify misconfiguration on "android.webkit.WebSettings". [Owasp Category M7. Client Side injection]

# 5. Drozer

Drozer is the leading security assessment framework for the Android platform.

Drozer allows us to assume the role of an Android app and interact with other apps. It can do anything that an installed application can do, such as make use of Android’s Inter-Process Communication (IPC) mechanism and interact with the underlying operating system.

Drozer also helps us to remotely exploit Android devices, by building malicious files or web pages that exploit known vulnerabilities. The payload that is used in these exploits is a rogue drozer agent that is essentially a remote administration tool. Depending on the permissions granted to the vulnerable app, drozer can install a full agent, inject a limited agent into the process using a novel technique or spawn a reverse shell.

**Step1:**

Download drozer files from this website:

1. Drozer (Windows Installer)
2. Drozer (Agent .apk only)
3. Drozer Users' Guide

<https://www.mwrinfosecurity.com/products/drozer/community-edition/>

**Step2:**

Install the Drozer in windows. And you will find a separate folder name “Drozer” in your C drive.

**Step3:**

Start the emulator and take **adb** access and Install “drozer-agent-2.3.4.apk”

After installing open the **“drozer agent”** and you will find embedded server written and **next** it will be **“ON”** button. Click on it.

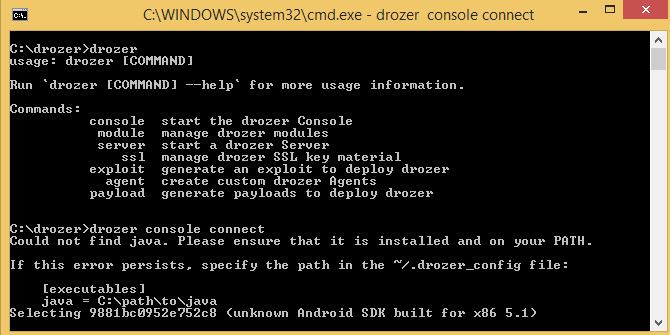
**Step4:**

As we have adb command prompt open. We have to do port forwarding before using drozer. As we know drozer by default uses port 31415. We will type command:

# adb forward tcp:31415 tcp:31415

**Step5:**

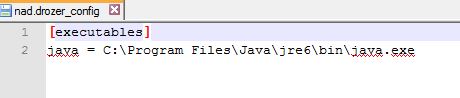
Now for using drozer, first we will go to drozer folder installed in C drive. And open command prompt, and we will type **drozer console connect command**:



**Step6:**

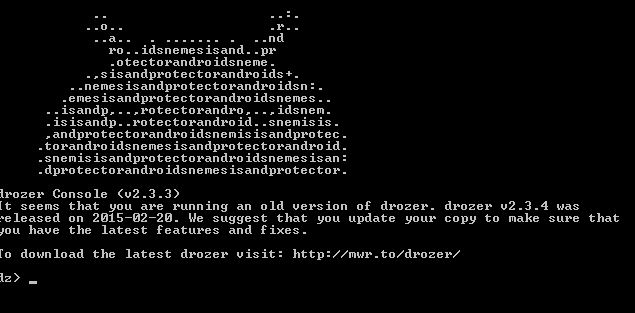
As we saw, an error occurred. So now we have to give our java installation path to drozer.

We will have to make a file **Anytext.drozer\_config**



**Step7:**

Now again we will type **drozer console connect** and will try to run drozer.



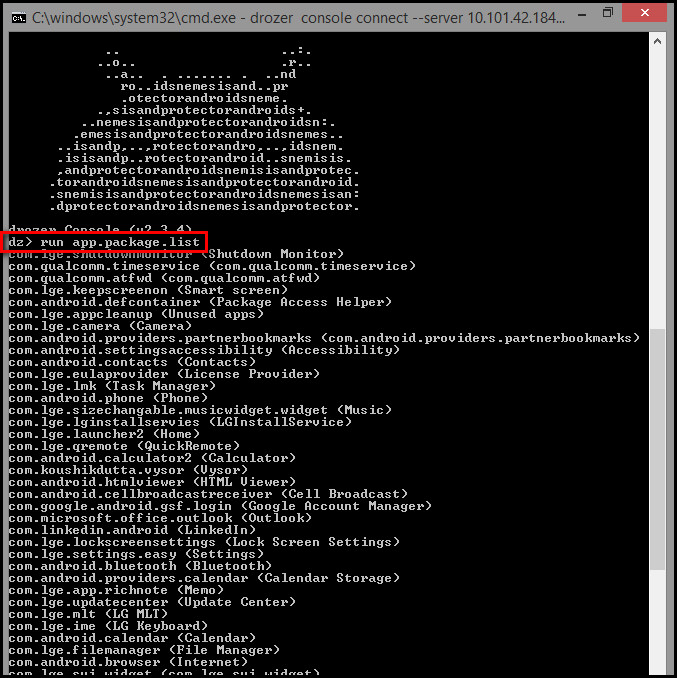
Now drozer has started and we can use for testing.

**Some basic drozer commands**

**List -**– shows the list of all Drozer modules that can be executed in the current session.



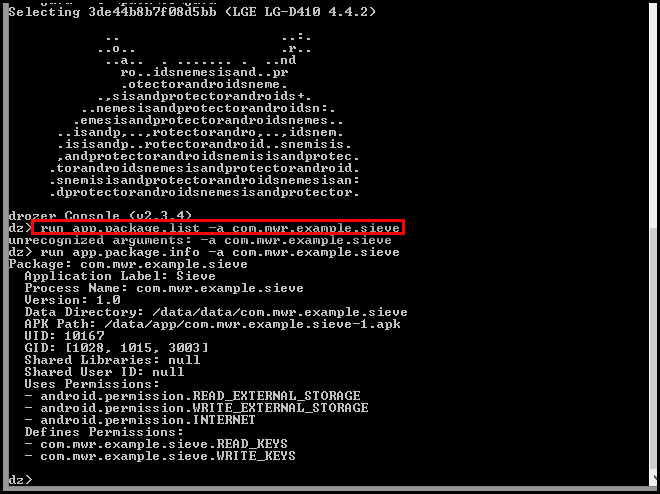
**run app.package.list -** To list out all the packages installed on the emulator



**run app.package.list –f (Keyword) –** to search specific app package

****

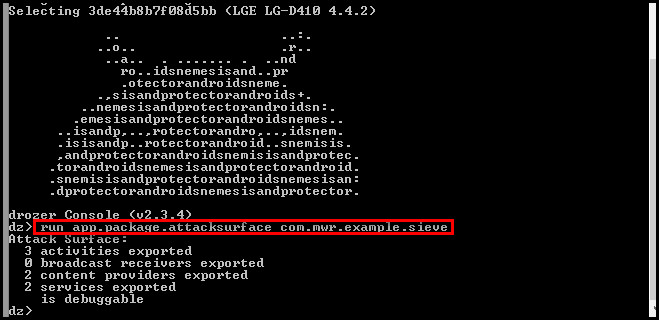
**run app.package.info –a (package identifier) -** To see some basic information about the package



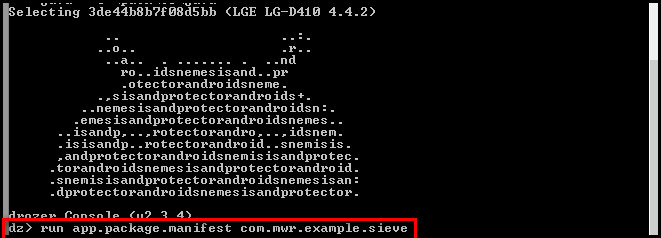
**run app.package.info –help -** use –help switch with any of module given above to get to know more about the functionality of that particular module

****

**run app.package.attacksurface (Package identifier) –** To identify the attack surface of our target application

****

**run app.package.manifest (Package Identifier)** – To view the manifest.xml file of particular application



[Owasp Categoy - M5. Poor Authorization and Authentication]

# 6.SQLite Database Browser

SQLite Database browser is a light GUI editor for SQLite databases, built on top of Qt. The main goal of the project is to allow non-technical users to create, modify and edit SQLite databases using a set of wizards and a spread sheet-like interface.

**Step 1:**

Open File Explorer which support Root (Esfile Explorer/Root Explorer). Navigate to root folder of the system.

Then navigate to **Data/Data.**

**Step 2:**

Look for the package name of the application and open the folder.

**Step 3:**

Here we check for sensitive data in following folder:

1.Database

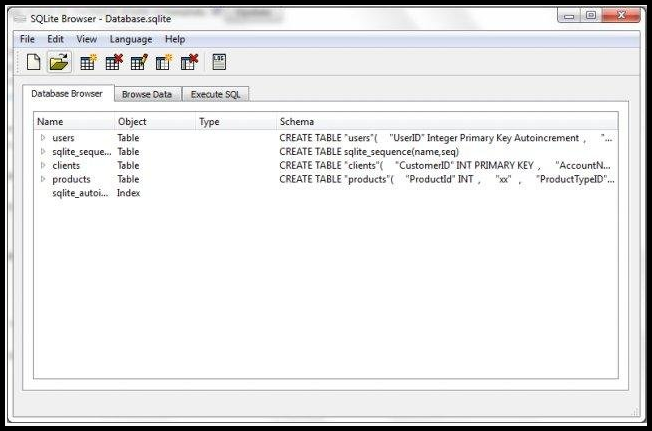
2. Cache

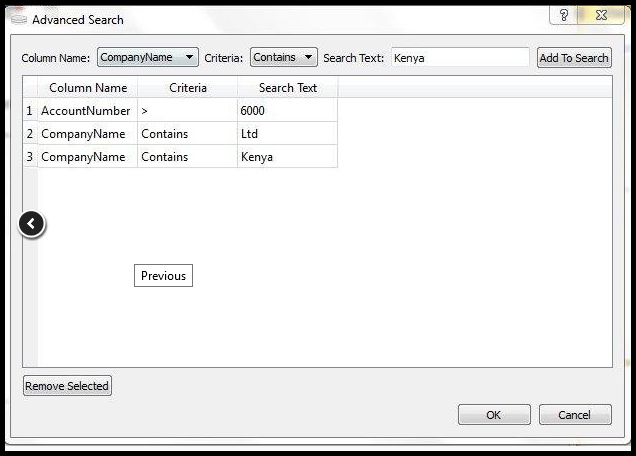
3. Shared\_prefs

**Step 4:**

For checking the content of Database, open the Database Folder and copy the content to the system.

Open the files present in the folder(Usual extensions are either .sqlite or .db or sqlitedb) with sqlite database browser.





[Owasp Category – M2. Insecure Data Storage]

# 7. CatLog App

**Step 1:**

Open CatLog app on the device and start the log reading on the app.

**Step 2:**

Open the application you are testing. Use it for a while.

**Step 3:**

Switch back to CatLog app and stop the log reading. Check for any sensitive data present in the logs.

[Owasp Category – M4. Unintended Data Leakage]

# 8. Class Name DE obfuscator

This is a simple script to parse through the .smali files produced by apktool and extract the .source annotation lines.

Obfuscation can be a pain to deal with when reversing an app. However, some apps do not have the .source annotation line removed/mangled druing the obfuscation process. This leaves the original Java class file name intact in the obfuscated code. We can abuse this to partially deobfuscate the class names.

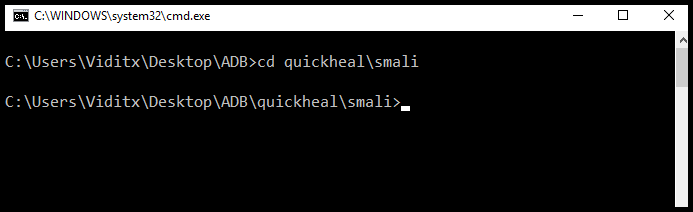
**Step 1:**

Let us assume that you have run apktool by -

apktool d <appname>.apk

Then go to the path where smali files are present

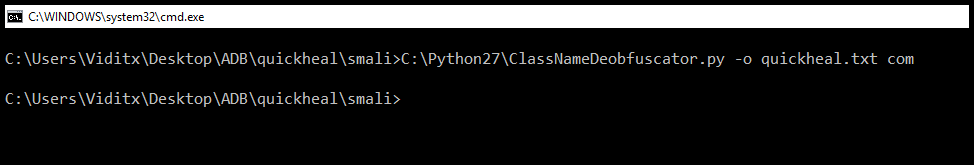
cd <appname>\smali



**Step 2:**

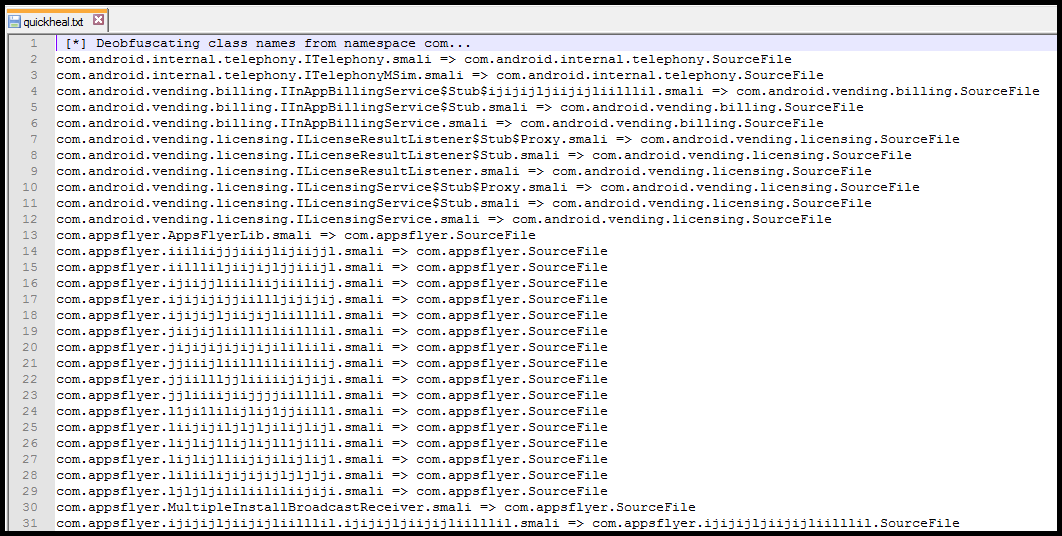
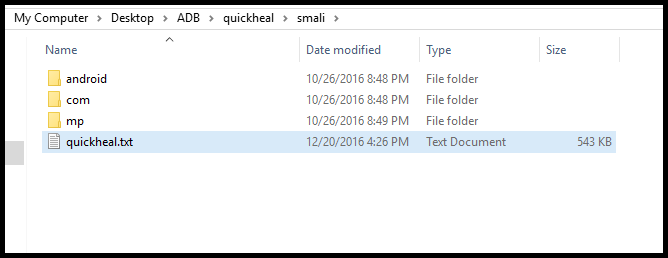
Run the following command to Save results to file:

python \path\ClassNameDeobfuscator.py -o <filename>.txt <foldername>



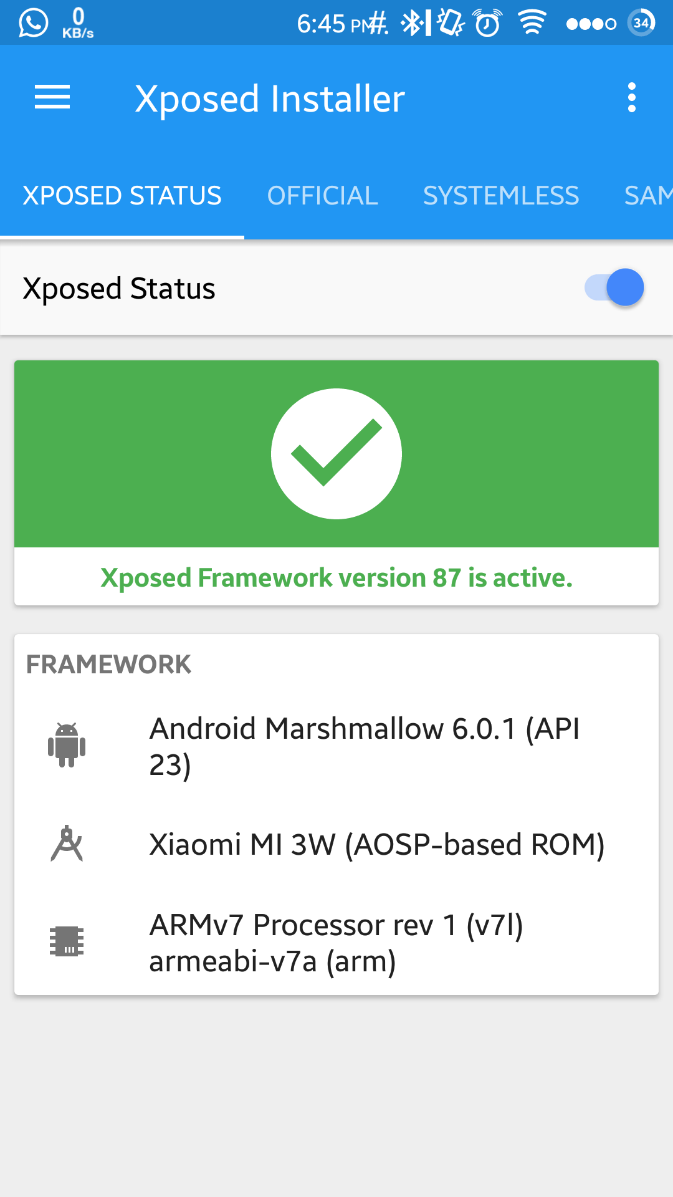
A new file will be created by the given name.

Open the file with notepad.



# 9. [Xposed Framework](http://forum.xda-developers.com/xposed/xposed-installer-versions-changelog-t2714053) –

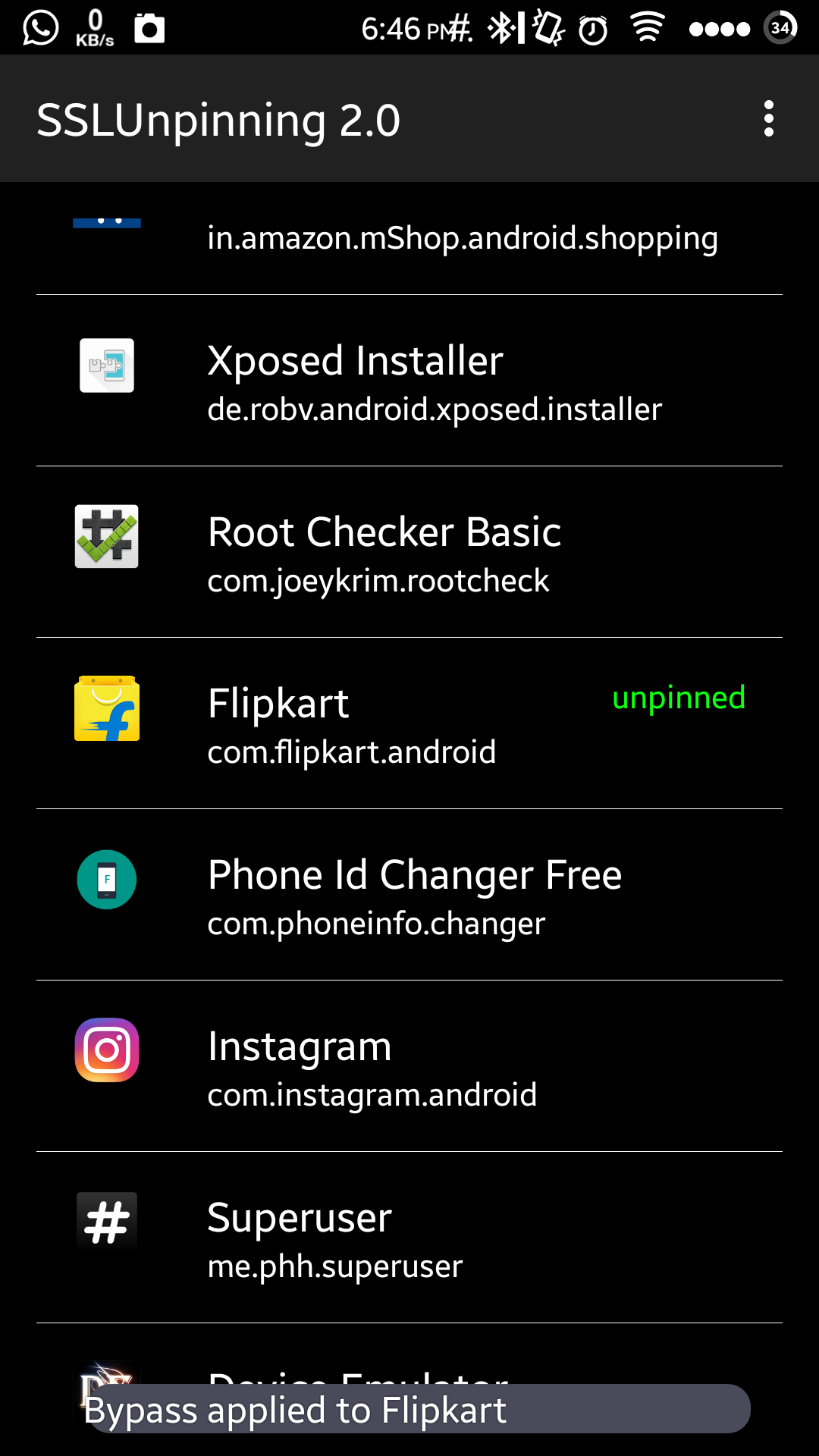
Xposed framework enables you to modify the system or application aspect and behaviour at runtime, without modifying any Android application package(APK) or re-flashing.



**Bypassing Root Detection and SSL Pinning with Xposed Framework**

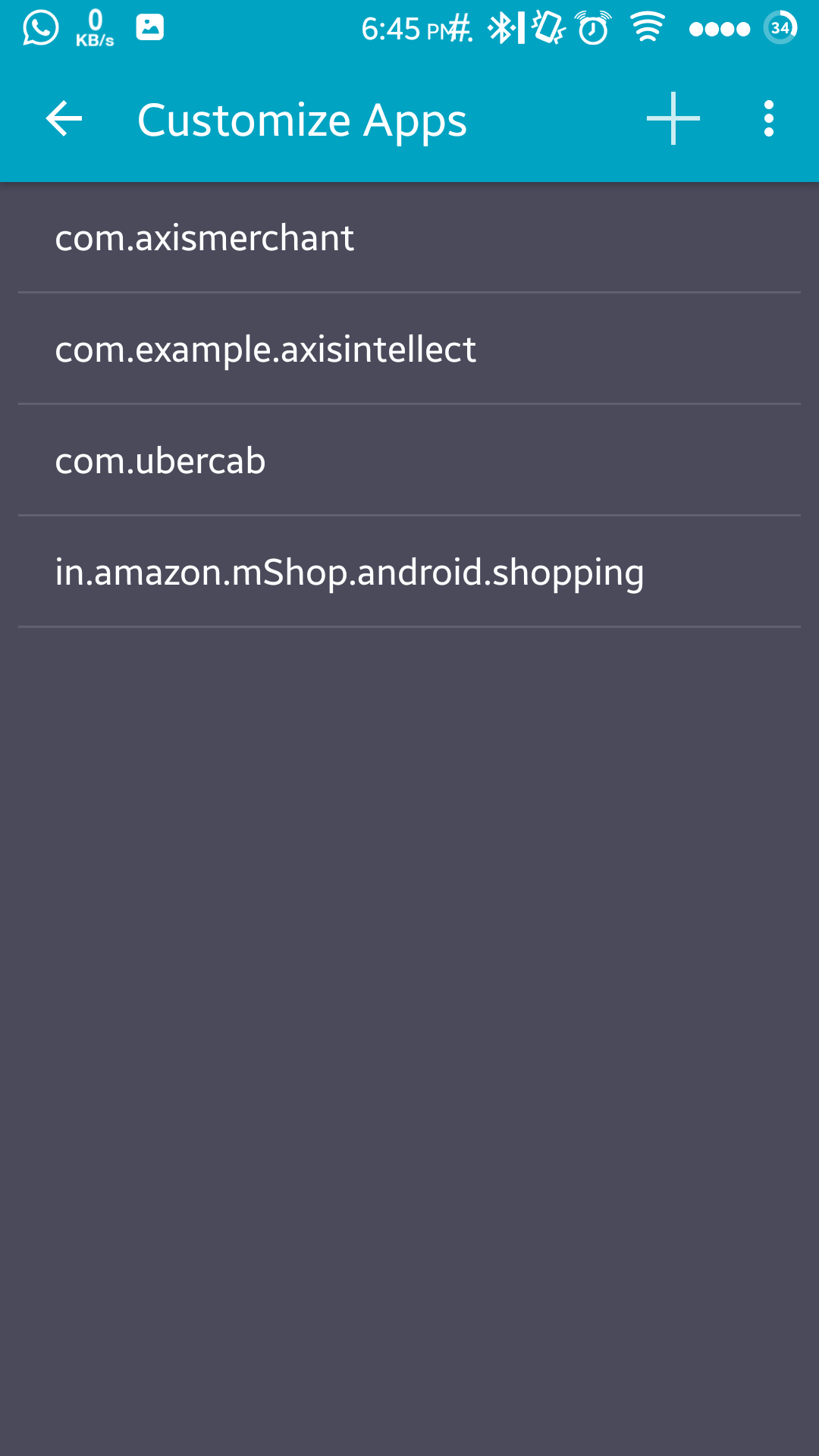
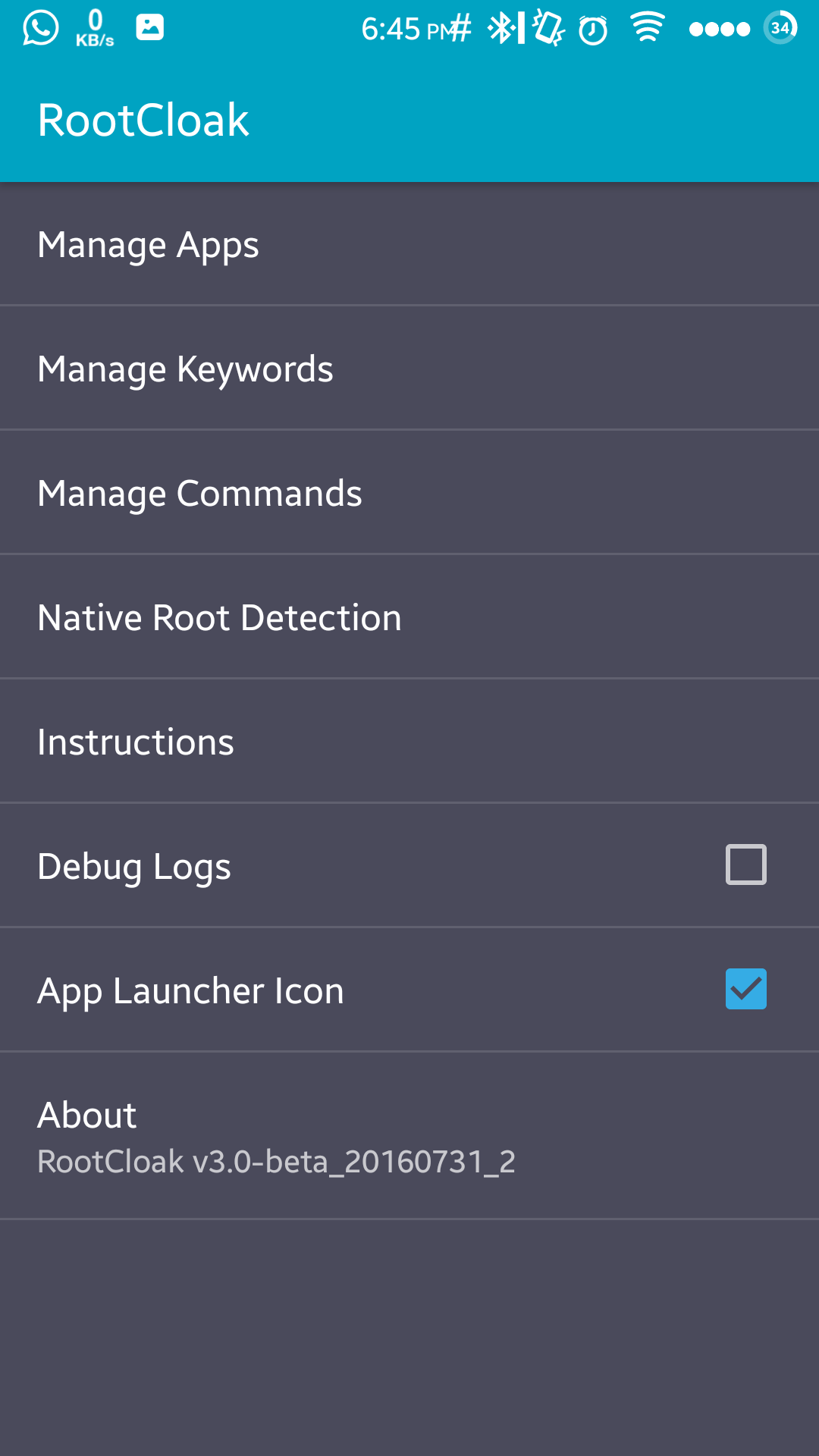
**SSLUnpinning - Xposed Module**

If you need to intercept the traffic from an app which uses certificate pinning, with a tool like Burp Proxy, the SSLUnpinning will help you with this hard work! The SSLUnpinning through Xposed Framework, makes several hooks in SSL classes to bypass the certificate verifications for one specific app, then you can intercept all your traffic.



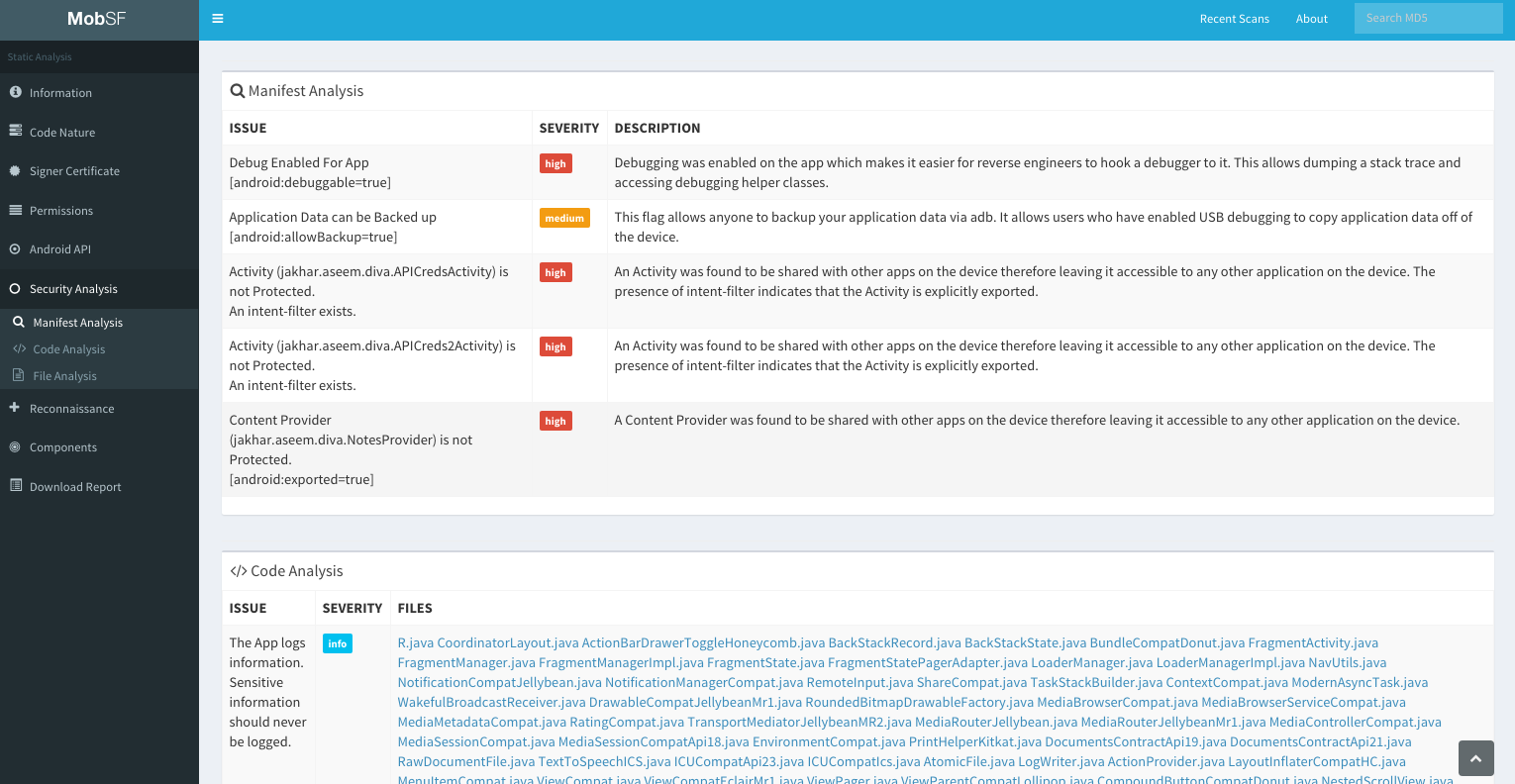
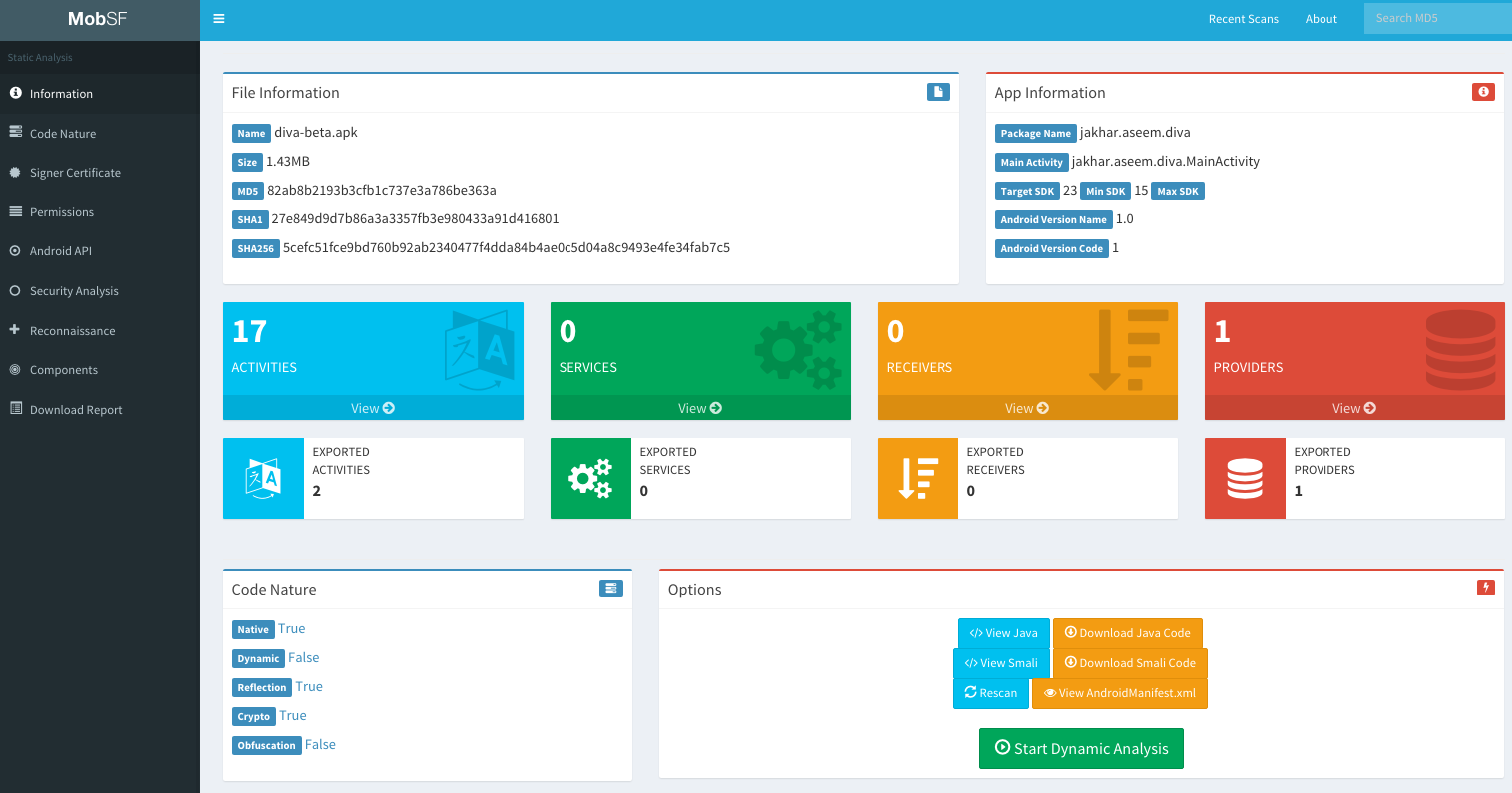
**RootCloak Plus – Xposed Module**

This allows you to run apps that detect root without disabling root. You select from a list of your installed apps (or add a custom entry), and using a variety of methods, it will completely hide root from that app. This includes hiding the su binary, superuser/supersu apks, processes run by root, and more.



# 10. [Mobile-Security-Framework-MobSF](https://github.com/ajinabraham/Mobile-Security-Framework-MobSF)

Mobile Security Framework (MobSF) is an intelligent, all-in-one open source mobile application (Android/iOS/Windows) automated pen-testing framework capable of performing static and dynamic analysis. It can be used for effective and fast security analysis of Android, iOS and Windows mobile Applications and supports both binaries (APK, IPA & APPX ) and zipped source code. MobSF can also perform Web API Security testing with it's API Fuzzer that can do Information Gathering, analyze Security Headers, identify Mobile API specific vulnerabilities like XXE, SSRF, Path Traversal, IDOR, and other logical issues related to Session and API Rate Limiting.

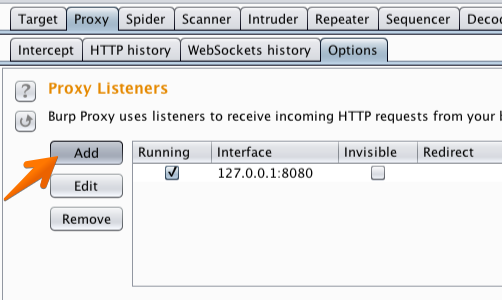


# 11. Configuring an Android Device to Work With Burp

Configure the Burp Proxy listener

In Burp, go to the “Proxy” tab and then the “Options” tab.

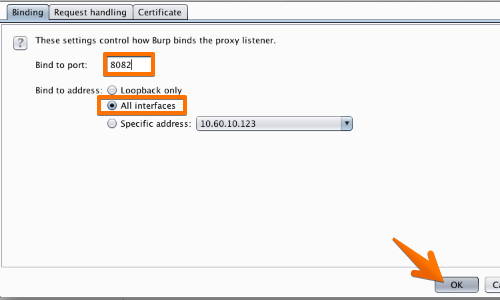
In the “Proxy Listeners" section, click the “Add” button.



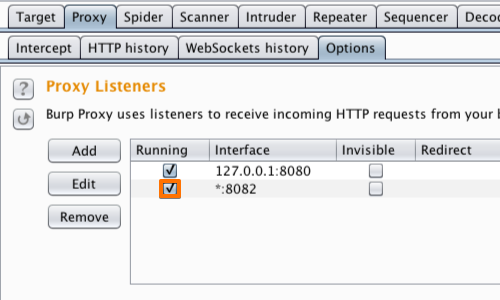
In the "Binding" tab, in the “Bind to port:” box, enter a port number that is not currently in use, e.g. “8082”.

Then select the “All interfaces” option, and click "OK".

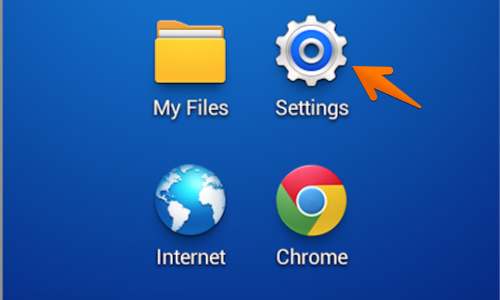
Note: You could alternatively edit the existing default proxy listener to listen on all interfaces. However, using different listeners for desktop and mobile devices enables you to filter these in the Proxy history view.



The Proxy listener should now be configured and running.

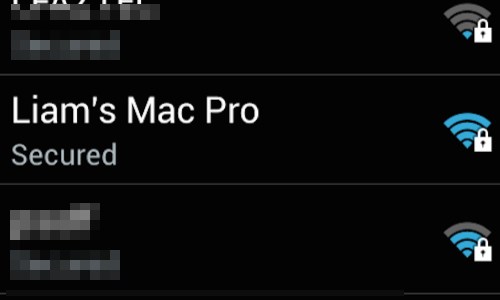


Configure your device to use the proxy



In your Android device, go to the“Settings” menu

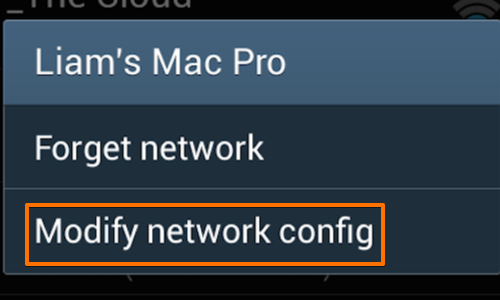
If your device is not already connected to the wireless network you are using, then switch the "Wi-Fi" button on, and tap the “Wi-Fi” button to access the "Wi-Fi" menu.



In the "Wi-Fi networks" table, find your network and tap it to bring up the connection menu.

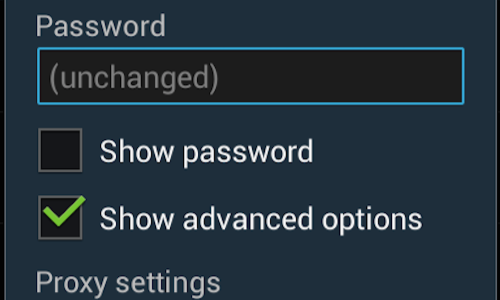
Tap "Connect".

If you have configured a password, enter it and continue.

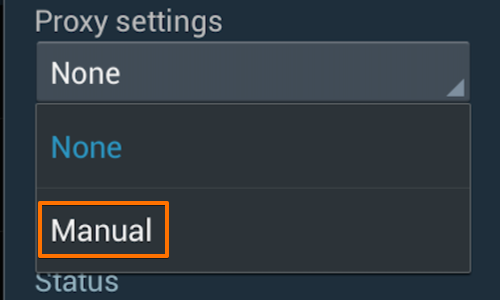


Once you are connected hold down on the network button to bring up the context menu.

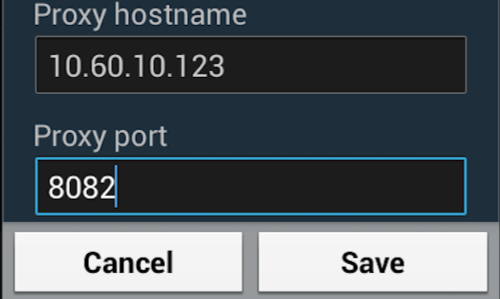
Tap “Modify network config”.



Ensure that the “Show advanced options” box is ticked.



Change the “Proxy settings” to “Manual” by tapping the button.

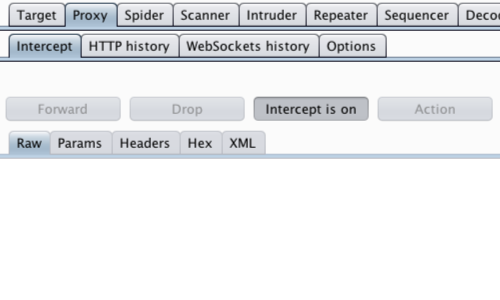


Then enter the IP of the computer running Burp into the “Proxy hostname”.

Enter the port number configured in the “Proxy Listeners” section earlier, in this example “8082”.

Tap "Save".

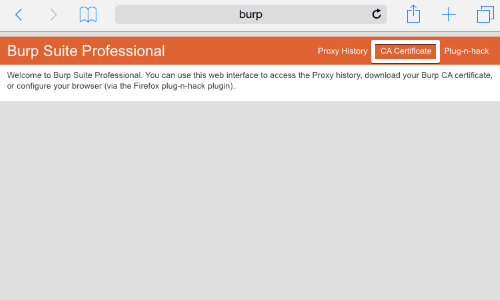
Test the configuration



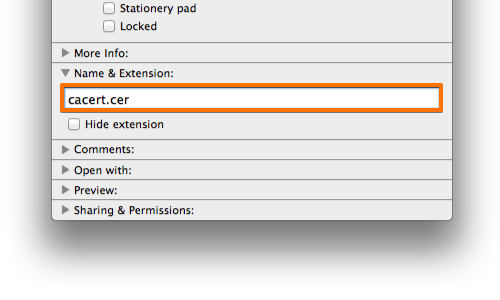
In Burp, go to the "Proxy Intercept" tab, and ensure that intercept is “on” (if the button says “Intercept is off" then click it to toggle the interception status).

# 12. Installing Burp's CA Certificate in an Android Device

On your mobile browser with Burp running, visit http://burp and click the "CA Certificate" link. Save the certificate file.

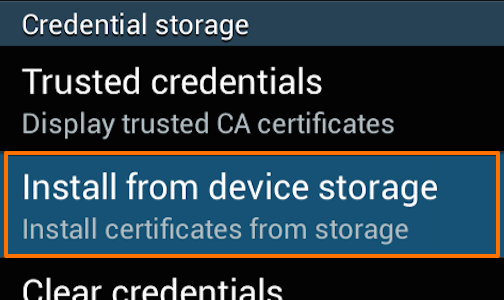


On EsFile Explorer and rename the file with the .cer file extension.



Next locate and tap the "Settings” icon. This may be located in the “Apps” menu or on one of the device's home screens.

In the “Security” menu select the “Install from device storage” from beneath the "Credential storage" header.



You will now be asked to “Name the certificate”, leave the certificate name as it is and tap “OK”.

The phone will revert to the security menu and will inform you via a small pop up that the certificate is installed.

You should now be able to visit any HTTPS URL via Burp without any security warnings.