iGNITE Technologies

ANDROID PENTEST LAB SETUP

ADB CHEATSHEET

WWW.HACKINGARTICLES.IN

TABLE OF CONTENTS

Abstract

Android is a mobile operating system based on a modified version of the Linux kernel and other open-source software, designed primarily for touchscreen mobile devices such as smartphones and tablets. Android is developed by a consortium of developers known as the Open Handset Alliance and commercially sponsored by Google. In this Paper we'll demonstrate some tools to conduct pentest on Android APK, manual analysis of the application as well as the major vulnerabilities; how to detect and fix them. But before beginning that, it is important to understand the architecture of an Android device, how is an application compiled and most importantly the composition of an Android application.

Android OS

Android OS

Brief history

Android Inc. was founded in Palo Alto, California, in October 2003 by Andy Rubin, Rich Miner, Nick Sears, and Chris White.
In July 2005, Google acquired Android Inc. for at least $50 million. Its key employees, including Rubin, Miner and White, joined Google as part of the acquisition. At Google, the team led by Rubin developed a mobile device platform powered by the Linux kernel. Google marketed the platform to handset makers and carriers on the promise of providing a flexible, upgradeable system.
The first commercially available smartphone running Android was the HTC Dream, also known as T-Mobile G1, announced on September 23, 2008.

Hardware

Android's main hardware platform for Android is ARM with x86 and x86-64 architectures also supported in the later releases. Since Android 5.0 Lollipop, 64-bit variants of all platforms are supported in addition to 32-bit variant.

Kernel

As of 2020, Android uses versions 4.4, 4.9 or 4.14 of Linux Kernel. Android Kernel is based on Linux Kernel's Long Term Support (LTS) branch.

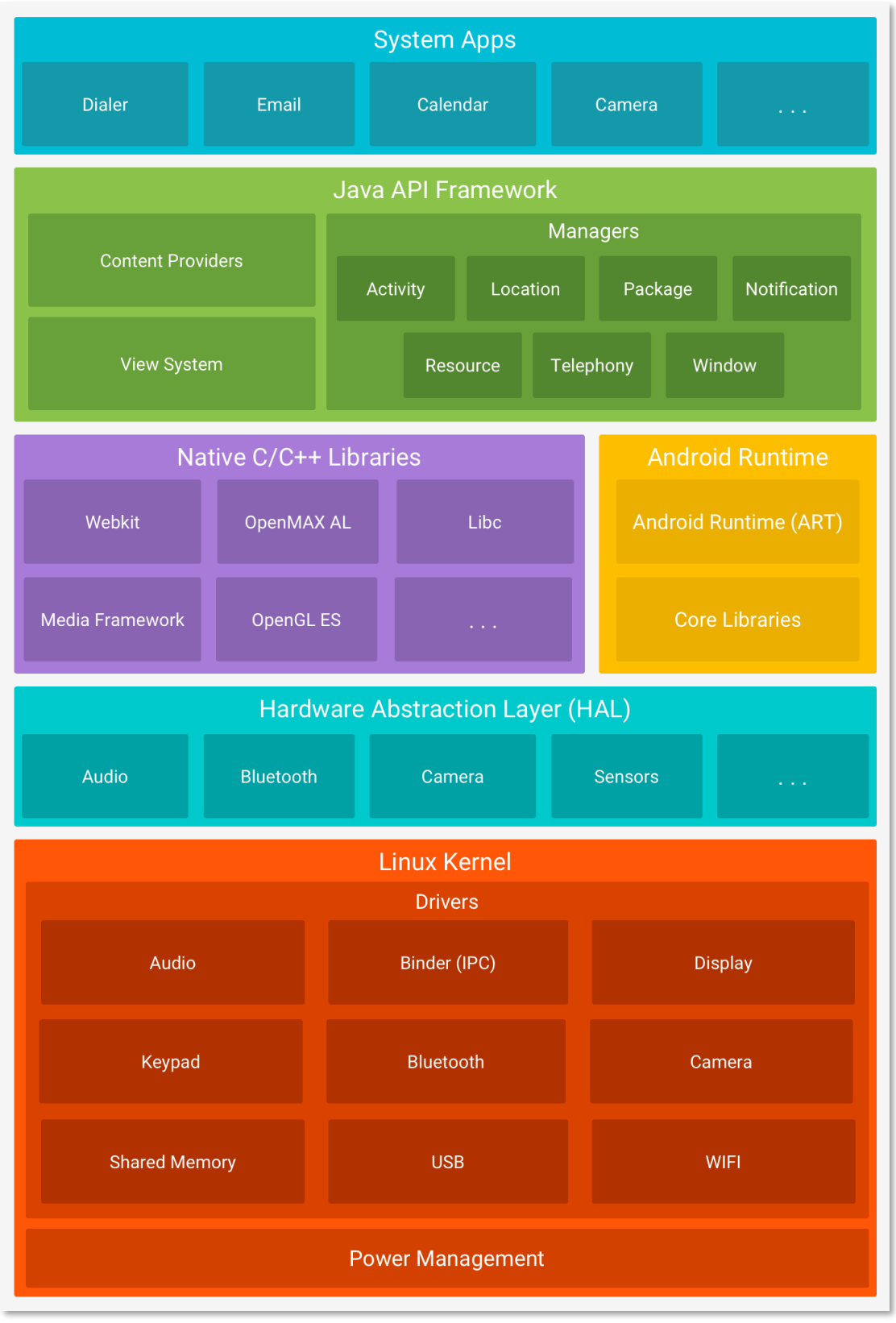
File System

Ever since the release of Android, it has made use of the YAFFS2. After Android 2.3 it used EXT4 file system, although many OEMs have experimented with F2FS. The following directories are there by default in any installation of Android:

- Boot: The partition contains the kernel, ramdisk etc. which is required for the phone to boot when powered on.
- System: Contains OS files which include Android UI and pre-installed apps
- Recovery: Alternative option to booting into OS, allows recovery and backup of partitions.
- Data: Saves user data, apps data, messaging, music etc. Can be traversed through the file browser. This is wiped when the factory reset is pressed. Sub-folders are:
    - Android – Default for app cache and saved data.
    - Alarms – Custom audio files for alarms.
    - Cardboard – Contains data for VR files.
    - DCIM – Stores pictures and videos were taken by Camera app.
    - Downloads – Stores downloaded files from the internet.
    - Notifications – Custom tones for notifications of some apps.
    - Music, Movies – Default folders to store songs and videos from third-party apps.
    - Pictures – Default folder to store pictures taken by third-party apps.
    - Podcasts – Stores podcasts files when you use a podcast app.
    - Videos – Stores downloaded videos from third-party apps.
- Cache: Storage of frequently used data and app components.
- Misc: Contains other important system setting information. Like USB config, carrier ID.

Android Architecture

Android Architecture



Kernel

The kernel is the most important part of the Android OS that provides an interface for the user to communicate with the hardware. It contains the essential drivers that are used by programs to instruct a hardware component a perform a specific function. These drivers are audio, display, Bluetooth etc.

Hardware Abstraction Layer (HAL)

A **hardware abstraction layer** (**HAL**) is a logical division of code that serves as an abstraction layer between a computer's physical hardware and its software. It provides a device driver interface allowing a program to communicate with the hardware.

Libraries

Sitting on the top of the kernel, libraries provide developer support to develop applications, resource file and even manifest. There are some native libraries like SSL, SQLite, Libc etc that are required by native codes to effectively perform a task.

Android Runtime

Android Runtime (ART) is an application runtime environment used by the Android OS. ***A runtime environment is a state in which program can send instructions to the computer's processor and access the computer's RAM. Android apps programmed in (let's say) Java, will be first converted to byte code during compilation packaged as an APK and run-on runtime.***

Android uses a Virtual Machine to execute any application so as to isolate the execution of the program from OS and protecting malicious code from the affecting system.

Before Android 4.4, that runtime used to be DVM (Dalvik Virtual Machine) which has since been replaced by Android Runtime (ART).

DVM used JIT (Just in time) compilation which worked by taking application code, analyzing it and actively translating it during runtime. ART uses AOT (ahead of time) compilation that compiles *.dex files (Dalvik Bytecode) before they are even needed. Usually, it is done during installation and stored in phone storage.

To be noted: After Android N, JIT compilation was reintroduced along with AOT and an interpreter in ARt making it hybrid to tackle against problems like installation time and memory.

Application Framework

The entire feature-set of the Android OS is available to the developer through APIs written in the Java language. These APIs make the most important components that are needed to create Android apps. These are:

- **View System:** Used to build an app's UI, including lists, text boxes, buttons etc.
- **Resource Manager:** Provides access to non-code resources like layout files, graphics etc.
- **Notification manager:** Allows the app to display custom alerts in the status bar.
- **Activity Manager:** Manages the lifecycle of apps and provides a common navigation back stack.
- **Content Providers:** Enables apps to access data from other apps like WhatsApp access data from the Contacts app.

System Applications

The pre-installed set of core applications used for basic functions like SMS, calendars, internet browsing, contacts etc.

APK Compilation and De-compilation

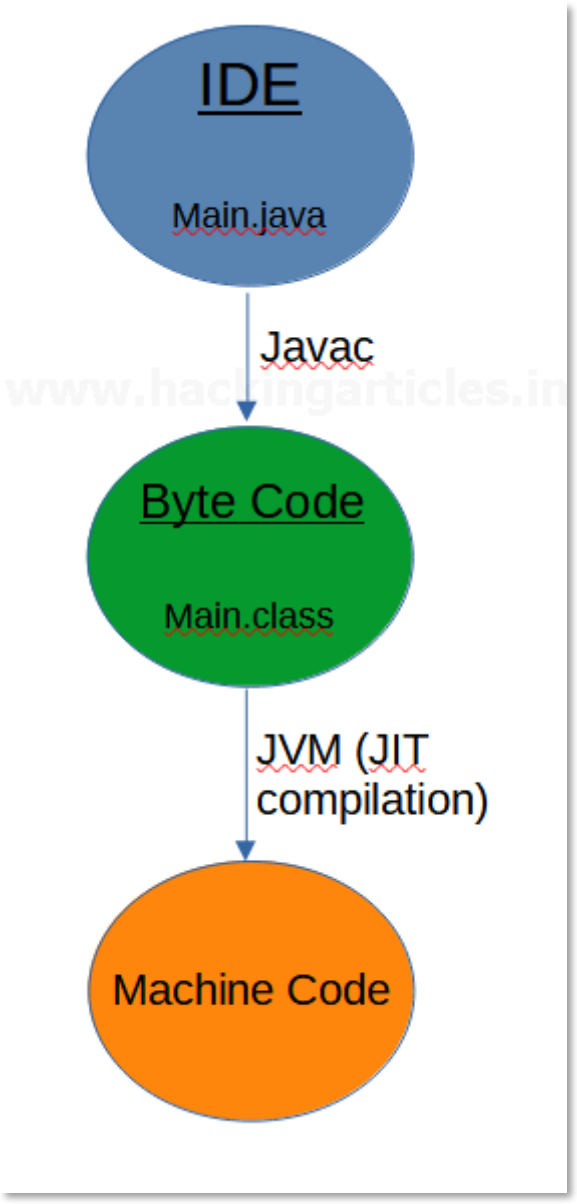
APK Compilation and De-compilation

Compilation

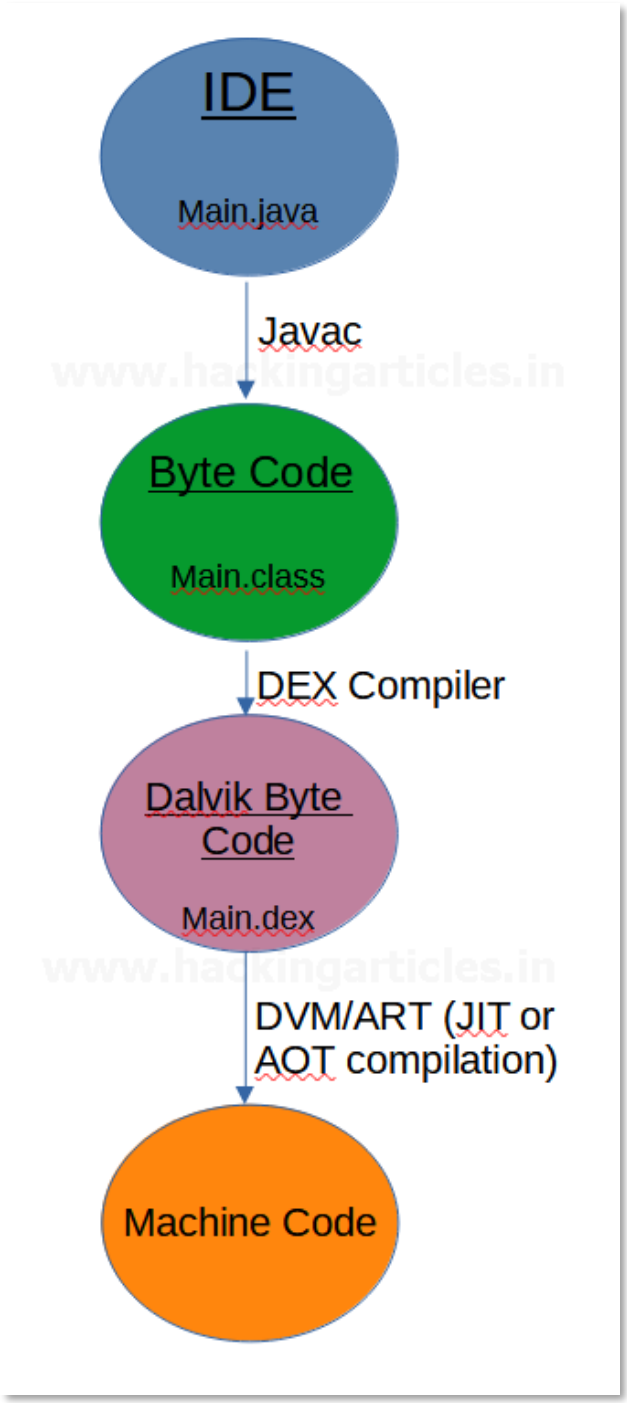
Android compilation process could be understood better when comparing to the java compilation process. To compile a java code the following steps are performed:

- A code written in main.java is compiled by javac (Java compiler).
- Javac compiles main.java into java byte code (main.class)
- A Java Virtual Machine (JVM) understands and converts Java byte code file (main.class) into machine code using JIT (Just-in-time) compiler.
- The machine code is then run by CPU.

It can be demonstrated in the following diagram:



Now, the main difference between Java and Android compilation process is that Android doesn't work with JVM because a JVM doesn't enable a variety of services with a limited processor speed and RAM. So, Android introduced a Dalvik Virtual Machine (DVM). However, in the latest Android releases ART is being used. The process is now as follows:

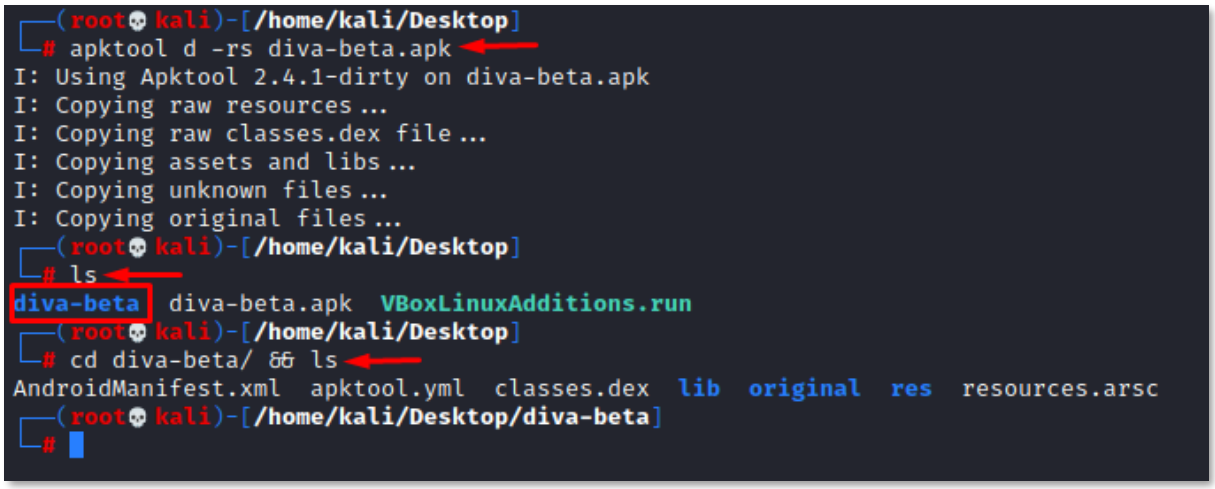


It is to be noted that even though Dalvik was replaced as the default runtime, Dalvik bytecode format (*.dex) is still in use.

Decompiling APK

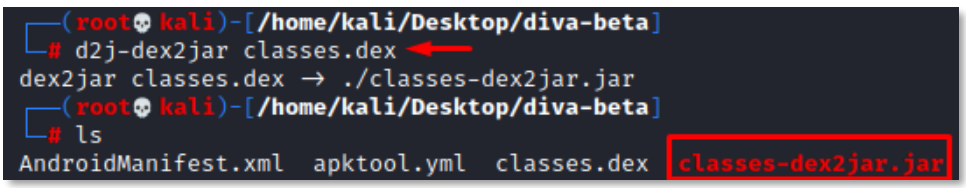
While conducting reverse engineering, APK de-compilation and re-compilation is done. We'll look after proper recompilation and reverse engineering in further articles, let's have a look at de-compilation to understand how an APK is essentially structured when packed in the format.
It is to be noted that an APK file is just a ZIP archive that contains XML files, dex code, resource files and other files.
So, to decompile it we can either:
Unzip <name>.apk
Or we can take aid of a great tool called apktool. So, we'll run the following command:

```
apktool -d -rs diva-beta.apk

ls

cd diva-beta/ && ls
```



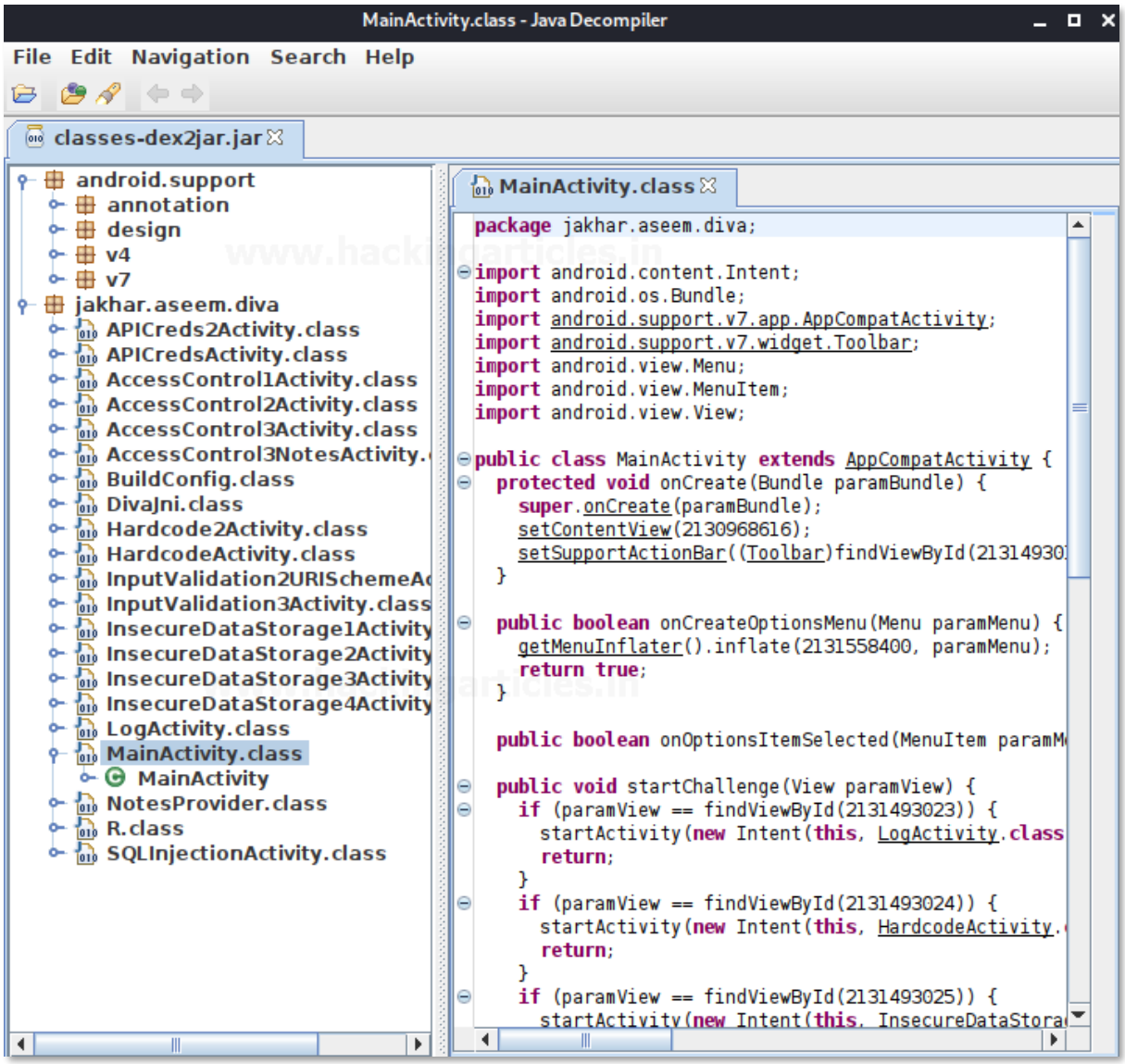
Here, we see that a folder has been created that essentially packs Manifest file, yml file, resources and a file named classes.dex
This dex file contains the dalvik byte code and we'll further disassemble this file into standard class files using the following command:

```
d2j-dex2jar classes.dex
```



This command further creates a .jar code structure. We will use a java decompiler to inspect the source code:

```
Jd-gui classes-dex2jar.jar
```

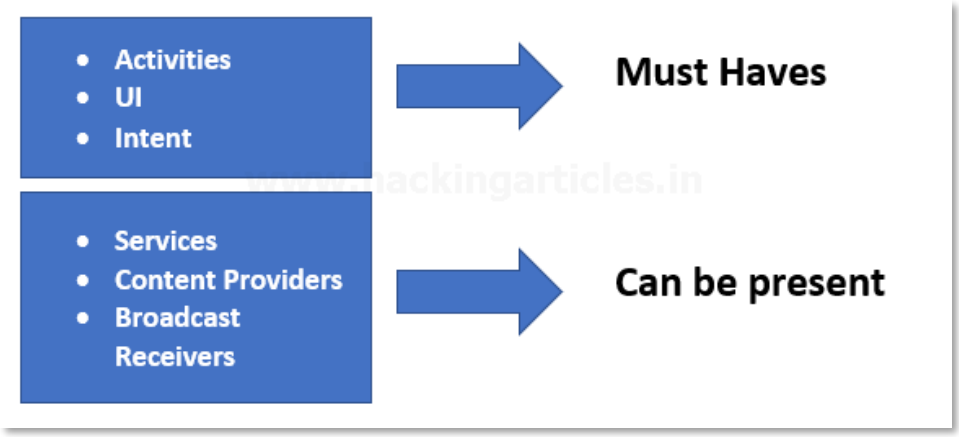


It is to be noted that the readability of this code entirely depends on whether or not an obfuscator (like ProGuard) was used while creating an APK file.

Composition of an APK

Composition of an APK

Any android application comprises of 6 essential things which are bound by the application manifest file that contains the description of how these interact with each other. Out of these, activities, UI and intents are a must have in any app while services, content providers and broadcast receivers can be present depending on needs. The components are:



Activities: Activity is a class in Android which, when implemented, represents a single screen with a user interface just like window or frame of Java. Every application has a ***MainActivity*** *(**java*** if code is in Java*)* which automatically runs by default for the first time when an application is run. All the actions performed in Activity pages are done using callbacks. There are 7 callbacks:

Sr.No	Callback & Description
1	**onCreate()** This is the first callback and called when the activity is first created.
2	**onStart()** This callback is called when the activity becomes visible to the user.
3	**onResume()** This is called when the user starts interacting with the application.
4	**onPause()** The paused activity does not receive user input and cannot execute any code and called when the current activity is being paused and the previous activity is being resumed.
5	**onStop()** This callback is called when the activity is no longer visible.
6	**onDestroy()** This callback is called before the activity is destroyed by the system.
7	**onRestart()** This callback is called when the activity restarts after stopping it.

Services: A **service** is a component that runs in the background to perform long-running operations without needing to interact with the user and it works even if application is destroyed. A service can essentially take two states:

1. **Started:** A service is **started** when an application component, such as an activity, starts it by calling *startService()*. Once started, a service can run in the background indefinitely, even if the component that started it is destroyed.
2. **Bound:** A service is **bound** when an application component binds to it by calling *bindService()*. A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with interprocess communication (IPC).

The **Service** base class defines various callback methods and the most important are given below:

Sr. No.	Callback and Description
1	**onStartCommand():** The system calls this method when another component, such as an activity, requests that the service be started, by calling *startService()*.
2	**onBind():** The system calls this method when another component wants to bind with the service by calling *bindService()*
3	**onUnbind():** The system calls this method when all clients have disconnected from a particular interface published by the service.
4	**onRebind():** The system calls this method when new clients have connected to the service, after it had previously been notified that all had disconnected in its *onUnbind(Intent)*.
5	**onCreate():** The system calls this method when the service is first created using *onStartCommand()* or *onBind()*. This call is required to perform one-time set-up.
6	**onDestroy():** The system calls this method when the service is no longer used and is being destroyed.

Content Providers: Content providers provide the content/data to Android applications from an Android system or other Android applications. To create a service, you create a Java class that extends the Service base class or one of its existing subclasses. Content providers provide the following four basic operations. These are also known as **CRUD operations**, where:

- **C – Create:** It is used for the creation of data in content providers.
- **R – Read:** It reads the data stored in the content provider.
- **U – Update:** It lets the editing in existing data in content providers.
- **D – Delete:** It deletes the existing data stored in its Storage.

To implement content providers, we'd use the following callbacks in Java:

Sr. No.	Callbacks and Description
1.	**onCreate() –** This method in Android initializes the provider as soon as the receiver is created.
2.	**query() –** It receives a request in the form of a query from the user and responds with a cursor in Android
3.	**insert() –** This method is used to insert the data into our content provider.
4.	**update() –** This method is used to update existing data in a row and return the updated row data.
5.	**delete() –** This method deletes existing data from the content provider.
6.	**getType() –** It returns the Multipurpose Internet Mail Extension type of data to the given Content URI.

Broadcast Receivers: Broadcast Receivers simply respond to broadcast messages from other applications or from the system itself. These messages are sometime called events or intents. For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is a broadcast receiver who will intercept this communication and will initiate appropriate action. Some of the broadcast receivers are:

- **intent.action.REBOOT**
- **intent.action.DATE_CHANGED**
- **intent.action.BOOT_COMPLETED**

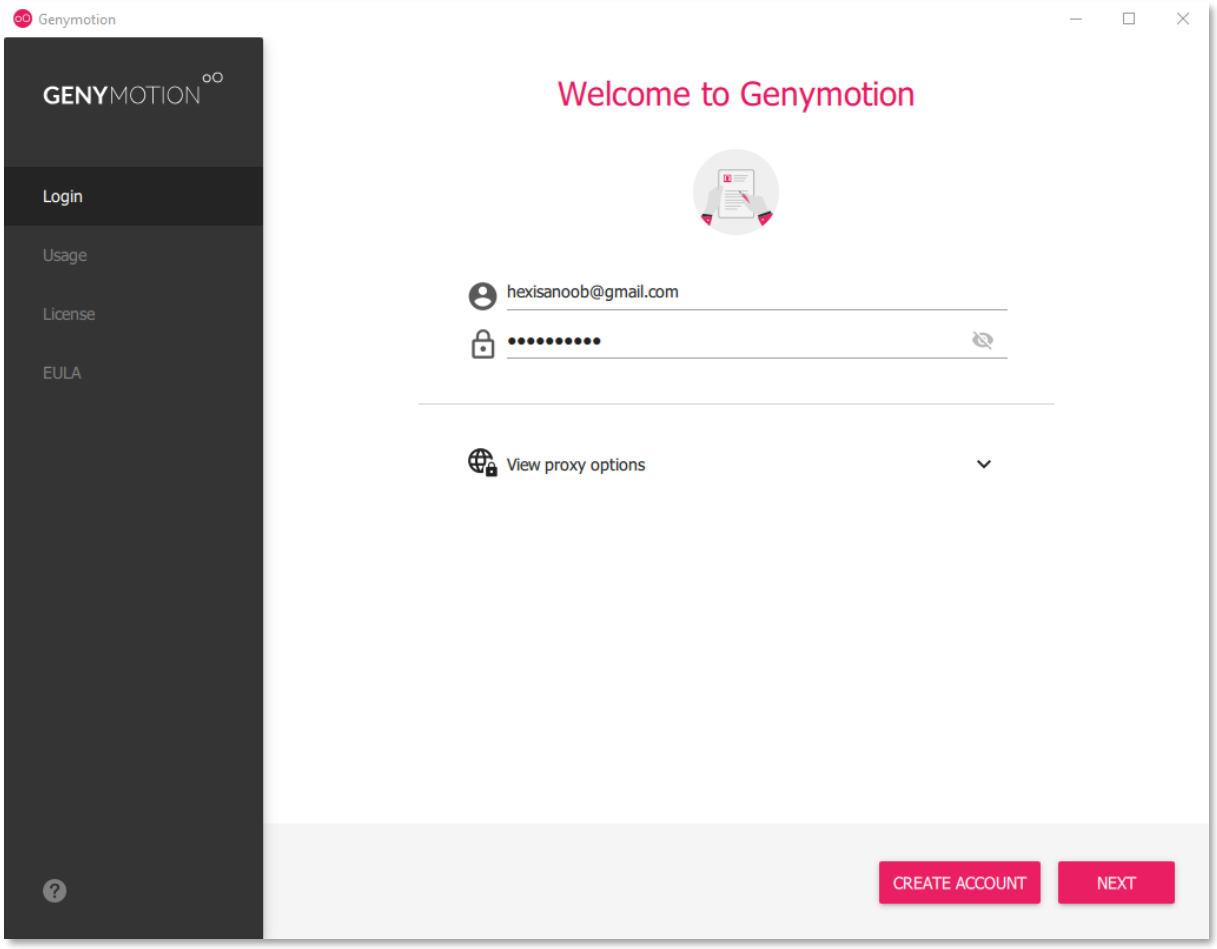
Intents: An Android **Intent** is an abstract description of an operation to be performed. It can be used with **startActivity** to launch an Activity, **broadcastIntent** to send it to any interested BroadcastReceiver components, and **startService(Intent)** or **bindService(Intent, ServiceConnection,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

Views: All the interaction of a user with the Android application is through the user interface(UI). **View** is the basic building block of UI (User Interface) in android. View refers to the android.view.View class, which is the **super class** for all the GUI components like TextView, ImageView, Button etc.

Installation of Virtual Environment

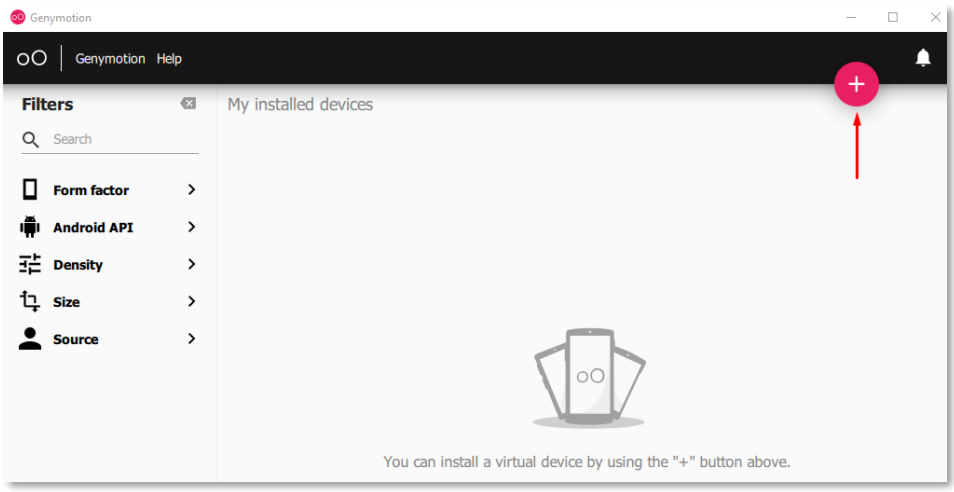
Installation of Virtual Environment

To download genymotion for free head over to **this** website and download genymotion. If you have VirtualBox installed you can choose without Virtualbox edition and otherwise with VirtualBox. Once you have downloaded you can login on the main screen and accept the license document to get started.

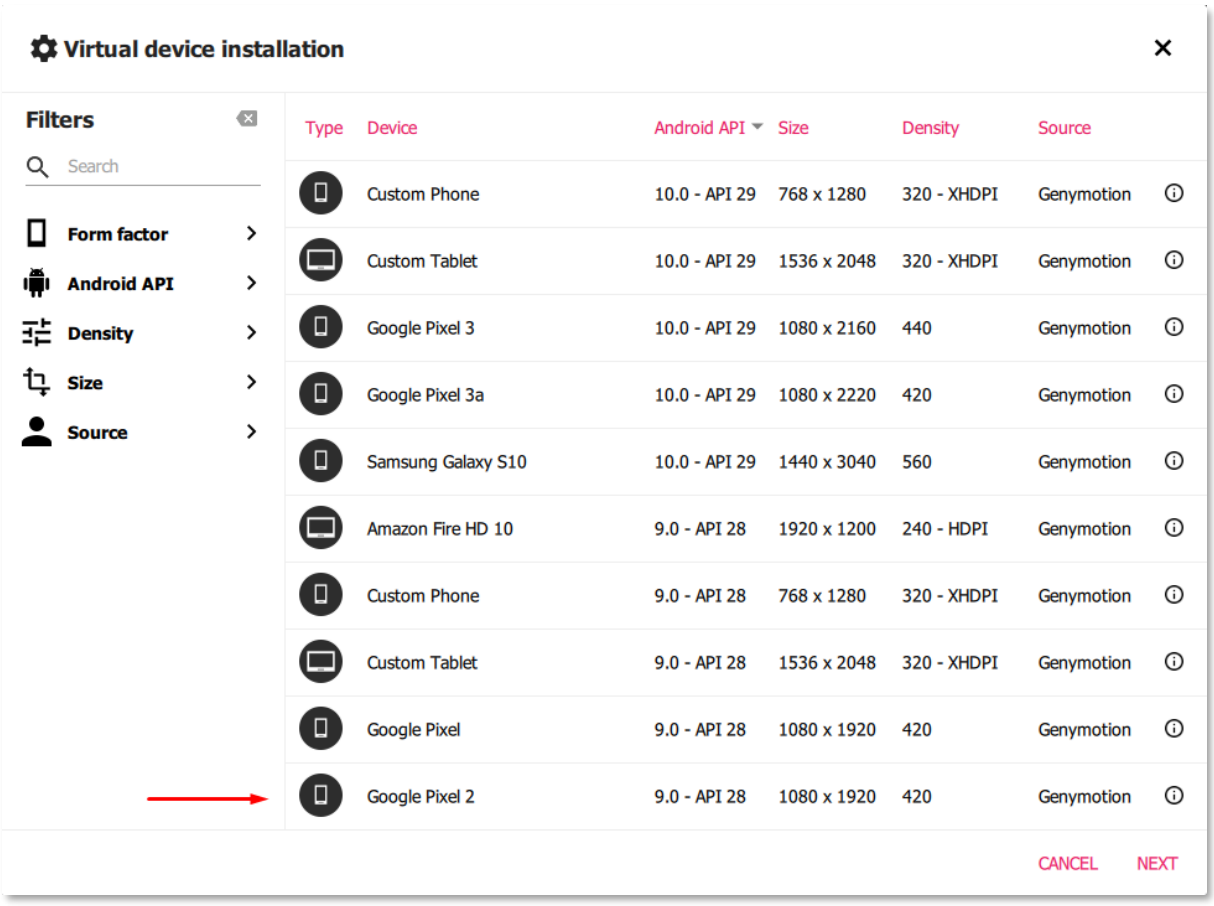


Creation of Android Virtual Machine (AVM)

After logging in you'll see a blank screen having 5 different options on the sidebar and a big plus **button (+)** that is used to create an Android virtual machine. Click on this plus sign.



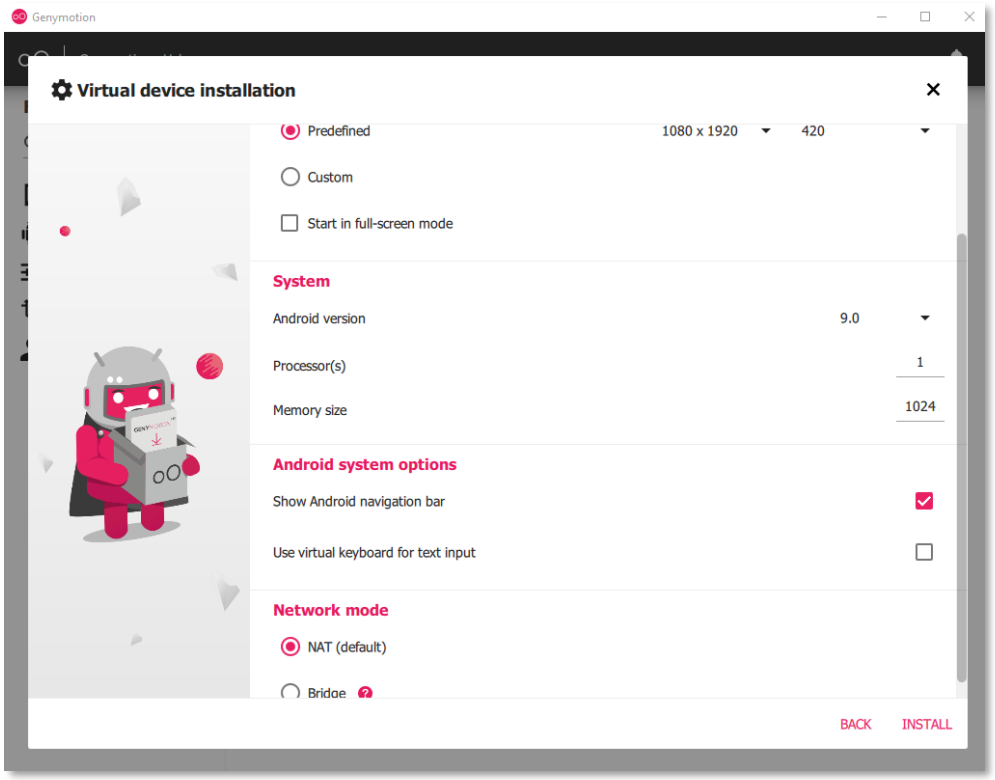
Upon clicking you'll see a screen like following that'll have various options available for different specifications of the virtual machine. You can choose your testing device and Android API you want to install. We'll be using **Google Pixel 2** with Android **API 28.**



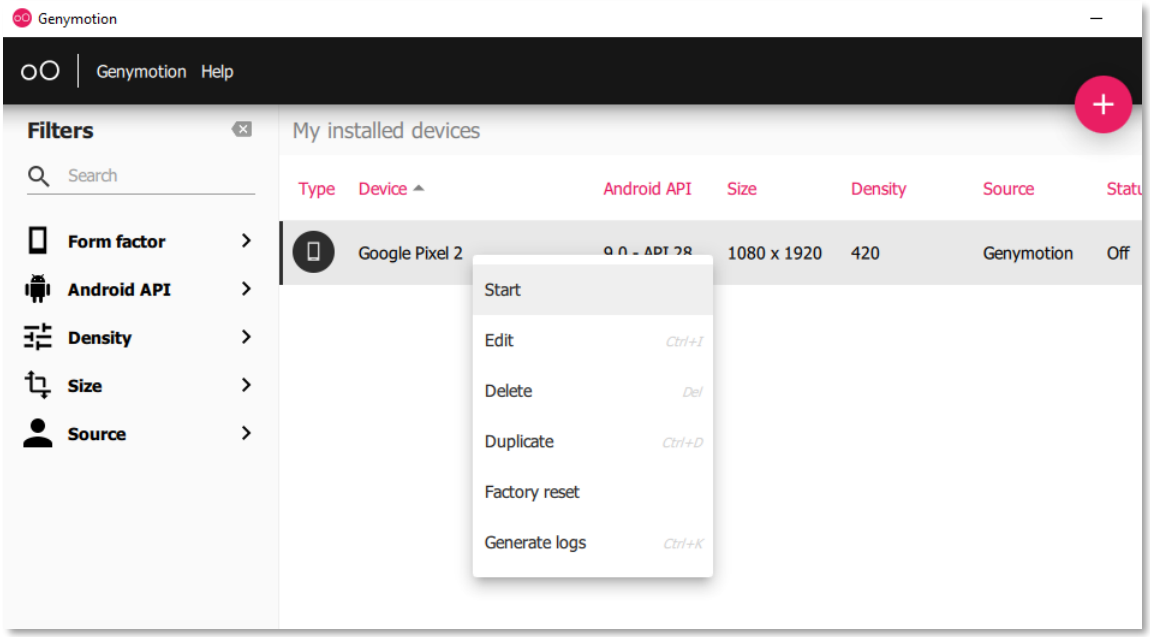
What is Android APIs? Android API refers to the collection of various software modules having different versions that make an Android SDK. In simpler words, A specific API version will correspond to a specific Android release as follows: –

Platform Version	API Level	VERSION_CODE
Android 11	30	R
Android 10	29	Q
Android 9	28	Pie
Android 8.1	27	Oreo_MR1
Android 8.0	26	Oreo
Android 7.1,7.1.1	25	N_MR1
Android 7.0	24	N
Android 6.0	23	M
Android 5.1	22	LOLLIPOP_MR1
Android 5.0	21	LOLLIPOP
Android 4.4W	20	KITKAT_WATCH
Android 4.4	19	KITKAT
Android 4.3	18	JELLY_BEAN_MR2
Android 4.2, 4.2.2	17	JELLY_BEAN_MR1
Android 4.1, 4.1.1	16	JELLY_BEAN
Android 4.0.3, 4.0.4	15	ICE_CREAM_SANDWICH_MR1
Android 4.0, 4.0.1, 4.0.2	14	ICE_CREAM_SANDWICH
Android 3.2	13	HONEYCOMB_MR2
Android 3.1.x	12	HONEYCOMB_MR1
Android 3.0.x	11	HONEYCOMB
Android 2.3.4 **Android 2.3.3**	10	GINGERBREAD_MR1
Android 2.3, 2.3.2,2.3.2	09	GINGERBREAD
Android 2.2.x	08	FROYO
Android 2.1.x	07	ECLAIR_MR1
Android 2.0.1	06	ECLAIR_0_1
Android 2.0	05	ECLAIR
Android 1.6	04	DONUT
Android 1.5	03	CUPCAKE
Android 1.1	02	BASE_1_1
Android 1.0	01	BASE

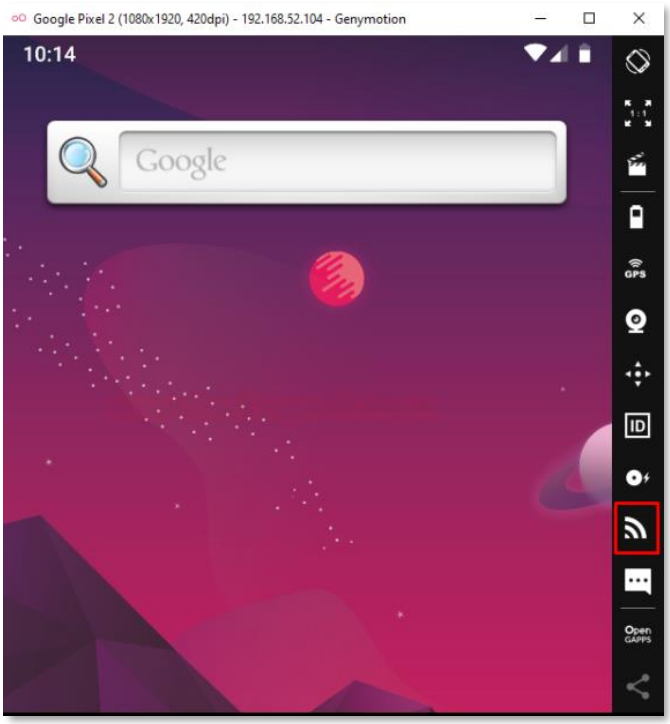
It is recommended to perform testing on newer APIs only.
Next step is to configure cores, RAM and networking mode. I'll be setting up 1 core for processing, 1GB of RAM and NAT mode as follows



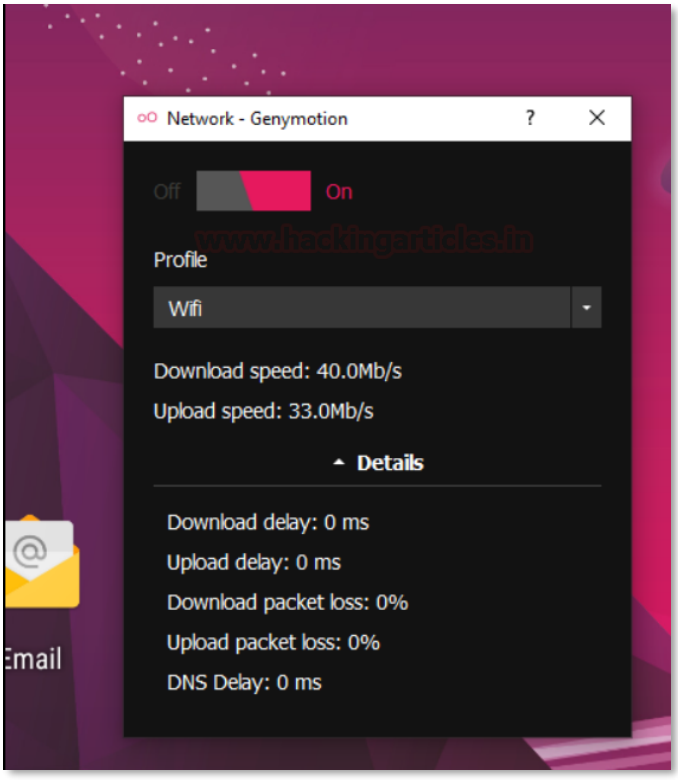
Once the install button is clicked, the respective API will first be downloaded if not already available in the SDK and then the machine will be ready to launch like following. Right click on it and press start



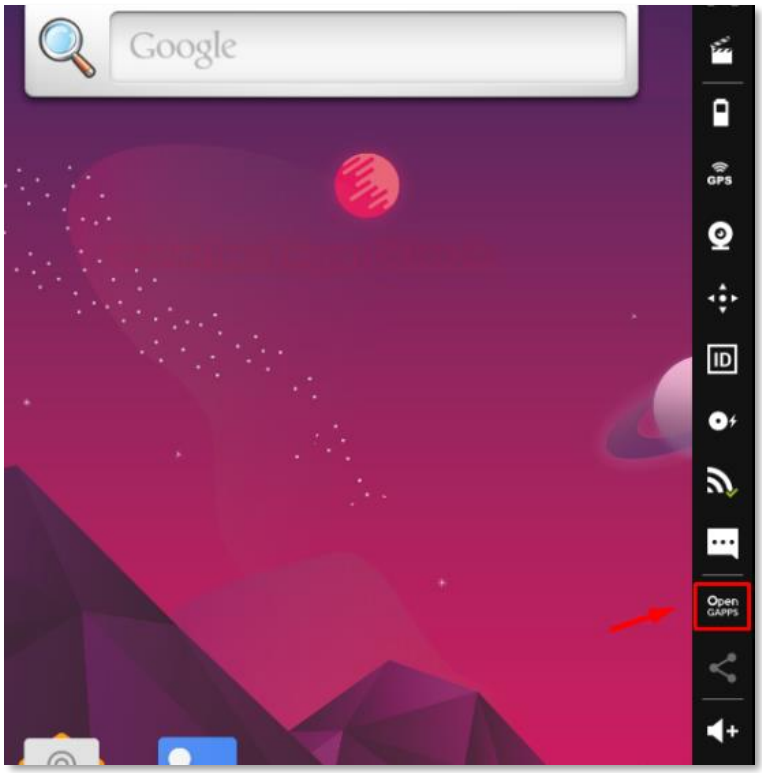
This will initiate a virtual box instance under genymotion's window and the first thing that we'll do is start networking on the virtual machine by clicking the signal icon.



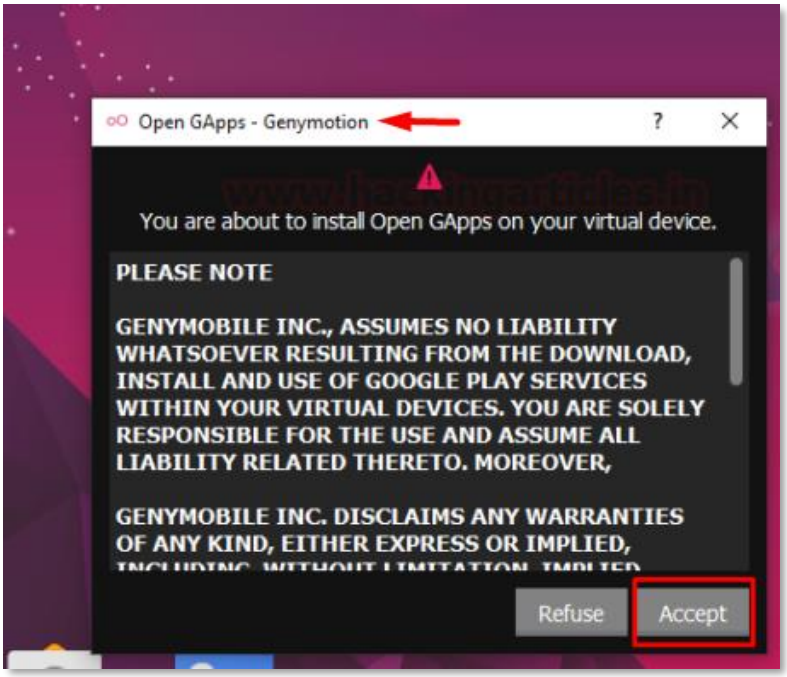
next is to turn the switch to ON and choose the respective networking adapter you want to allocate. In my case a Wi-Fi connection is active so I'll be allocating Wi-Fi to the machine.



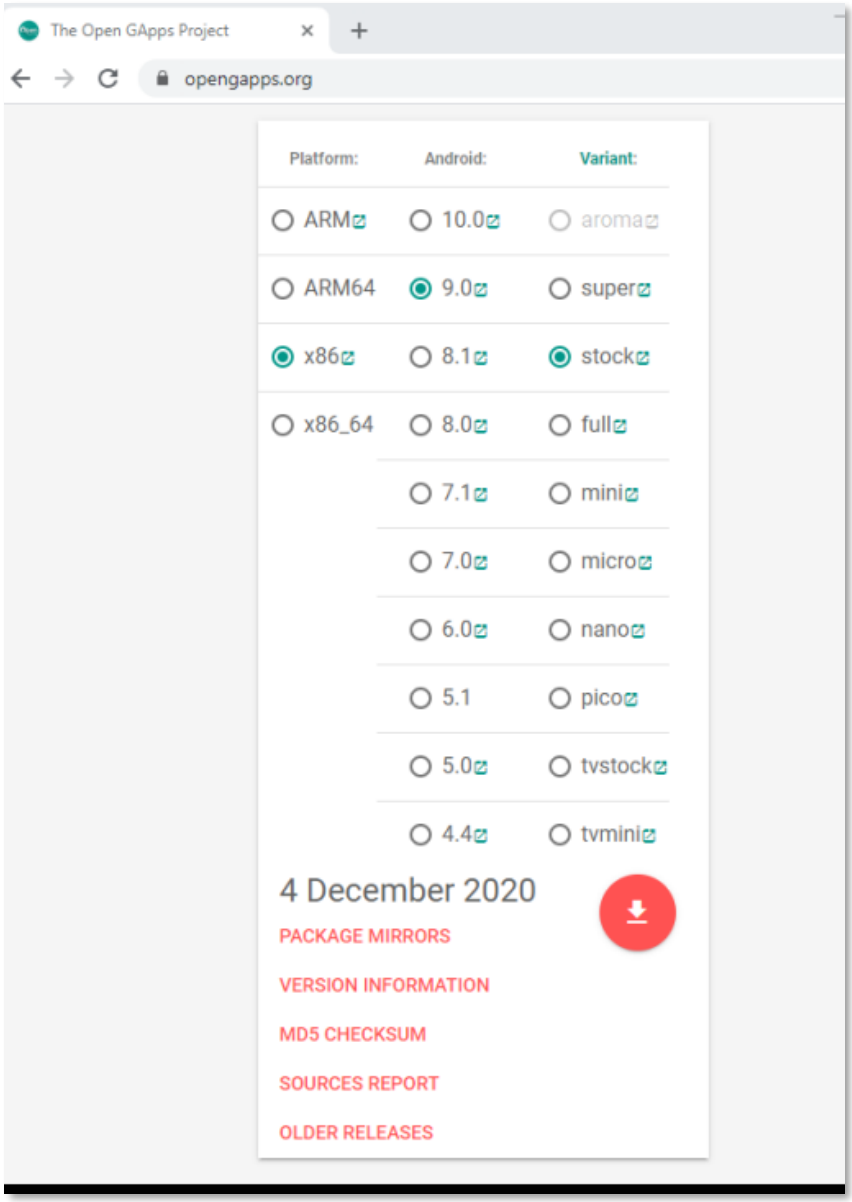
Next is to install Google apps in the virtual machine which won't come preloaded in the API. This is needed in situations where an app from the play store has to be downloaded and tested or some other google app has to be used within this virtual machine. We'll do this by clicking the Open GAPPS button.



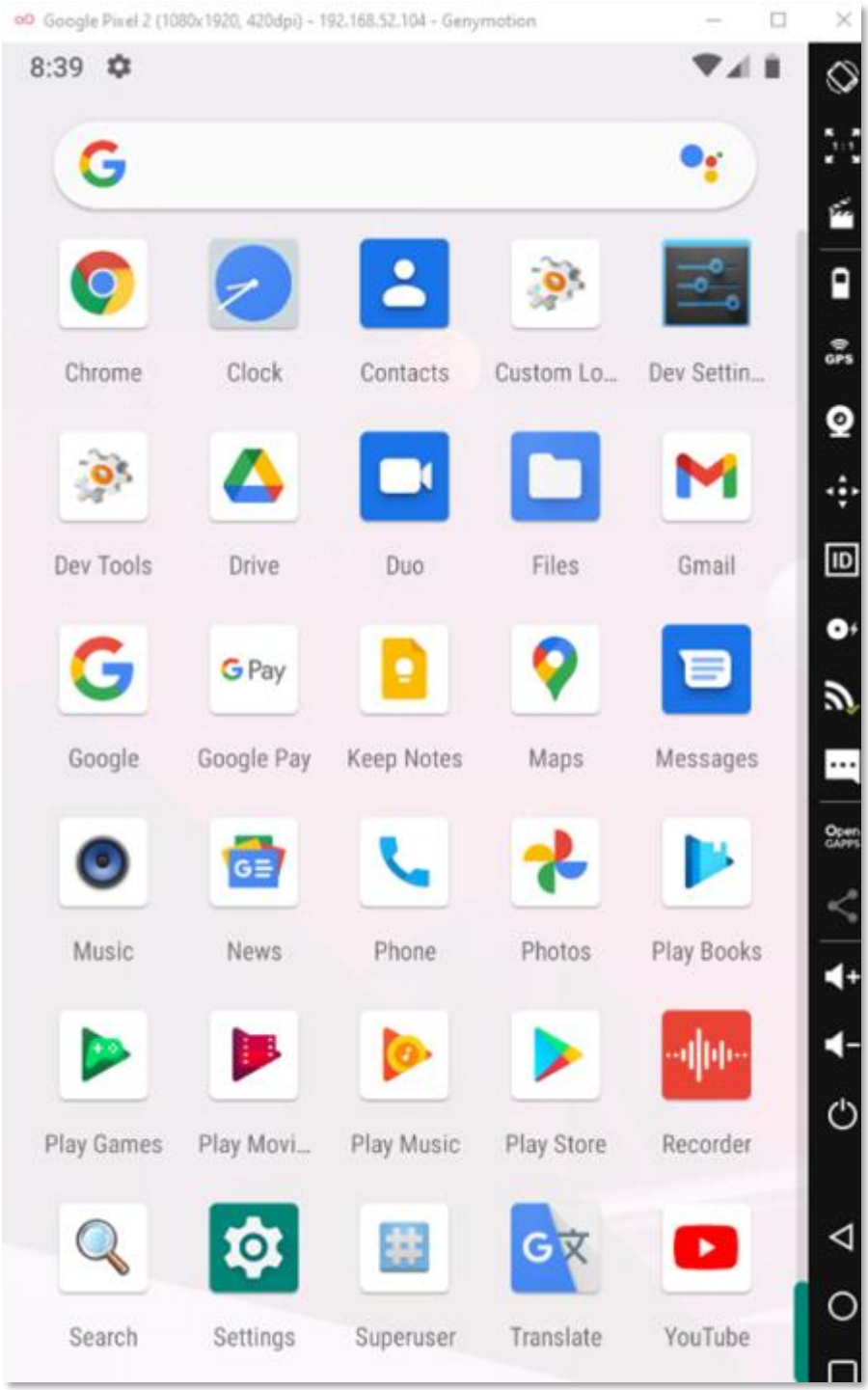
Simply accept the agreement and we're good to go.



IMPORTANT: In many scenarios, this type of installation may fail giving an "**archive error**". In such cases, it is recommended to download Open GApps archive from their website **here**.
I chose x86 architecture and android version 9.0 with stock variant



After download, simply drag and drop the archive in genymotion window to install the apps. After installation and reboot, you'll see the Google Apps have been successfully installed.



Getting Started With Debugging

Getting Started With Debugging

Now we are done with our environment setup. In previous screenshots, you must have noticed the IP address of the virtual machine on the title bar. In my case, it is **192.168.52.104**. It is time to try and connect to this virtual machine using the Android Debug Bridge (adb tool).
We head over to Kali and type the following command: –

```
apt install adb

adb connect 192.168.52.104

adb devices -l
```

What is USB debugging? Debugging is a way for an Android device to communicate with the Android SDK over a USB connection. It allows an Android device to receive commands, files etc from the PC, and allows the PC to pull crucial information like log files from the Android device.

What is ADB? Android Debug Bridge is a utility that provides debugging features for android devices. ADB can be used to conduct debugging over USB as well as over TCP. We'll learn more about ADB in detail in the next section.

ADB Command Cheatsheet

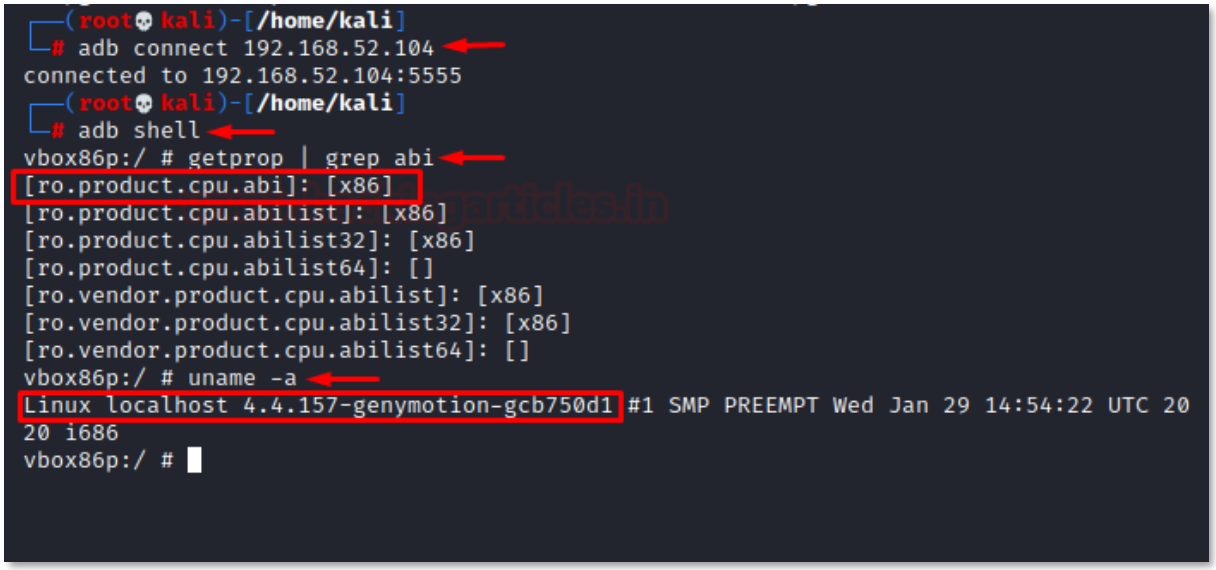
ADB Command Cheatsheet

Basics

1. Shell – As we are aware, Android has a Linux kernel. To spawn a shell in the connected device using ADB, we'll use the command:

```
adb connect 192.168.52.104

adb shell

getprop | grep abi
```

The last command helps you view the architecture of the device you're using.

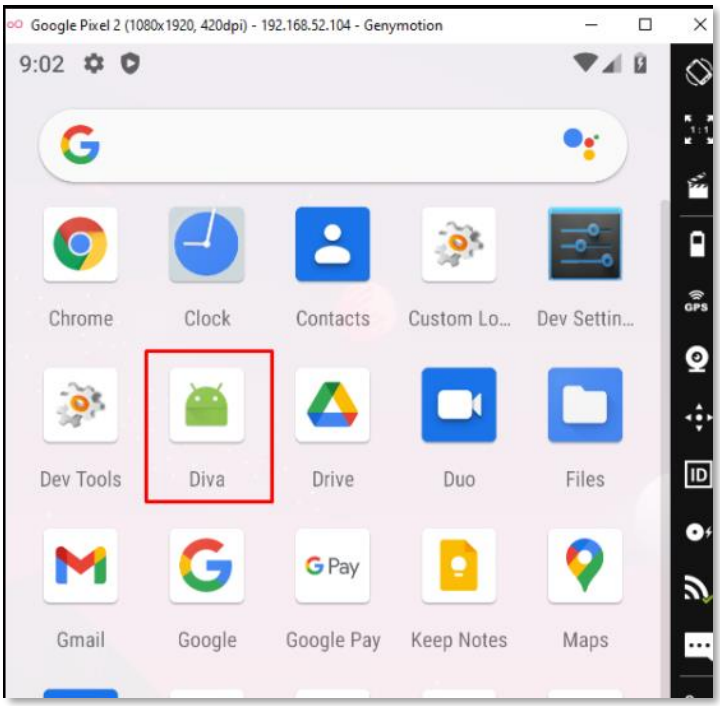


2. Installation of APK in device - Let's say we have an APK kept in the downloads folder of our system that we want to install in one of the devices. Here, I've downloaded DIVA from **this** link. We can do that in adb using the following command:

```
cd Downloads/

tar -xvf diva-beta.tar.gz

adb install diva-beta.apk
```

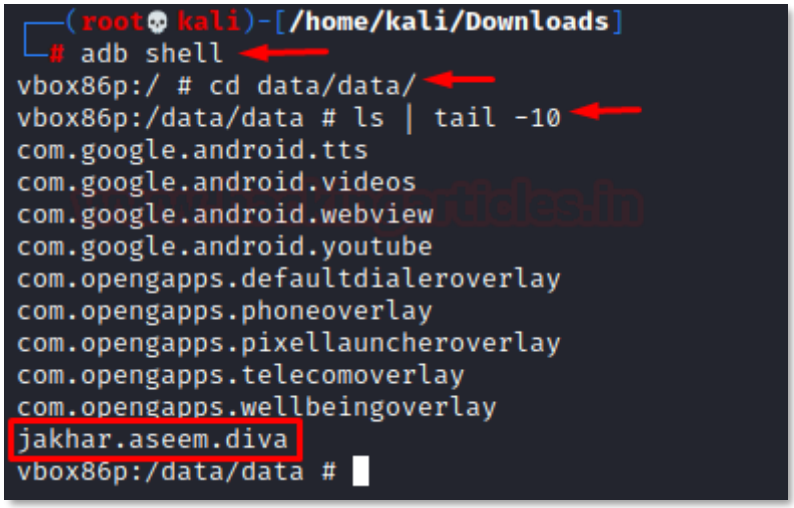
```
┌──(root💀kali)-[/home/kali]
└─# cd Downloads/
┌──(root💀kali)-[/home/kali/Downloads]
└─# ls
diva-beta.tar.gz
┌──(root💀kali)-[/home/kali/Downloads]
└─# tar -xvf diva-beta.tar.gz
diva-beta.apk
┌──(root💀kali)-[/home/kali/Downloads]
└─# ls
diva-beta.apk  diva-beta.tar.gz
┌──(root💀kali)-[/home/kali/Downloads]
└─# adb install diva-beta.apk
Success
┌──(root💀kali)-[/home/kali/Downloads]
└─#
```

And sure enough, the application was found in the device.



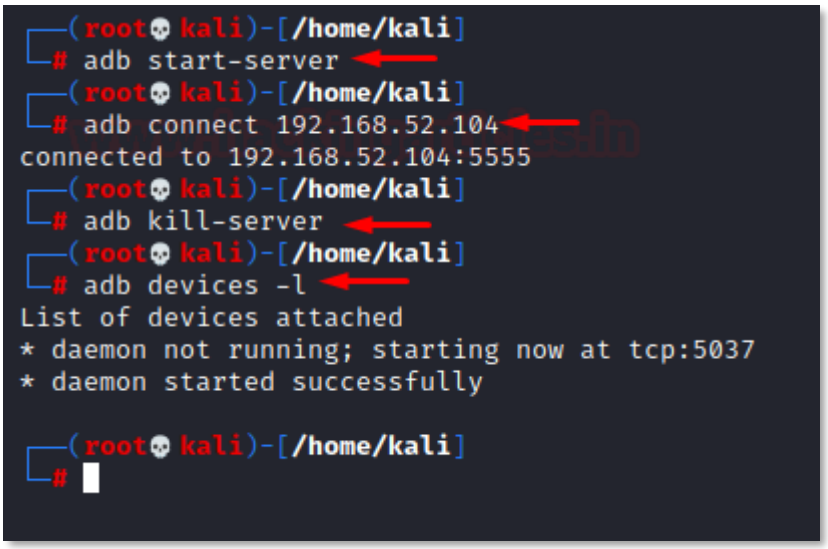
3. Viewing Installed Applications - All the applications installed in the device is kept under /data/data folder. So, we'll simply traverse to the folder and view the 10 recent installed applications.

```
adb shell

cd data/data/

ls | tail -10
```



4. Starting and Stopping adb service - This is simply done by the following two commands:

```
adb start-server
adb kill-server
```

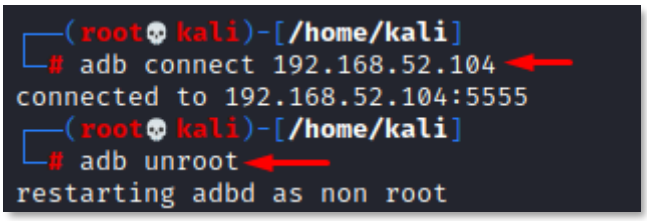


Please note that many of the commands in the upcoming demonstration would require you to run them as root on the android device and hence, we'll run adb as root. To run it as root you need the following commands:

```
adb root
```

To revert back to unroot status:

```
adb unroot
```

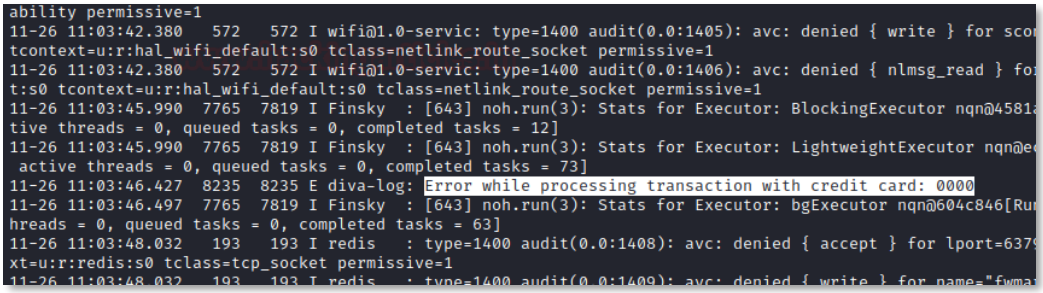


Logs

To monitor device logs we'll use the logcat tool.

```
adb logcat
```

As you can see all types of logs are now visible. We input credit card number in one of the functions in DIVA and that's also visible here demonstrating insecure logging vulnerability.

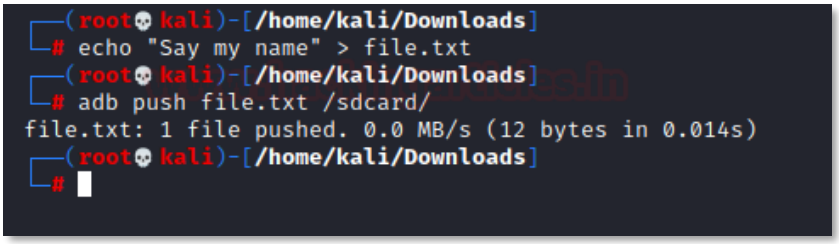


We can explore much other filters and options in logcat under the man page like prioritizing logs, writing logs to log files, dumping logs, rotating log files etc in the master page **here**.

Pulling and Pushing Files

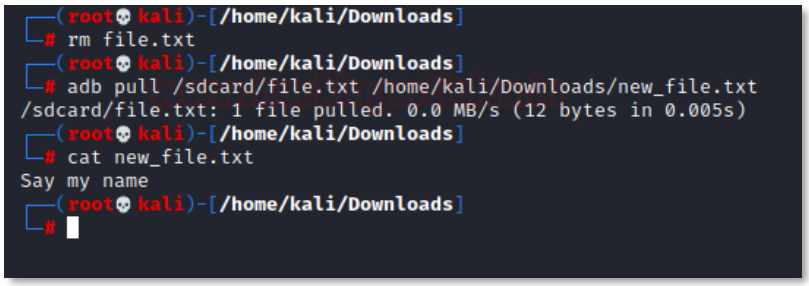
To copy a file from system to android and android to the system, one can use the adb push and pull. Let's consider copying file to the Android device using push.

```
echo "Say my name" > file.txt
adb push file.txt /sdcard/
```



To ensure that the file had got transferred to the location /sdcard/ we first delete the original file and then pull that file and access it.

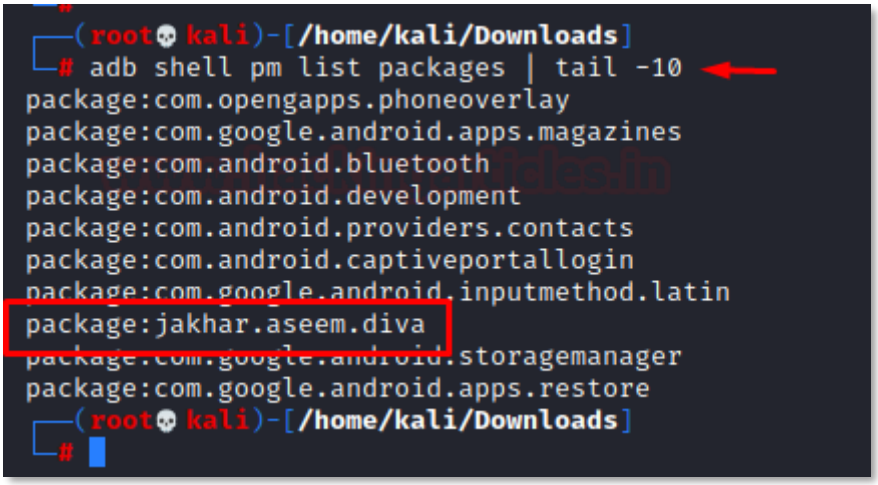
```
rm file.txt

adb pull /sdcard/file.txt /home/kali/Downloads/new_file.txt

cat new_file.txt
```



pm tool

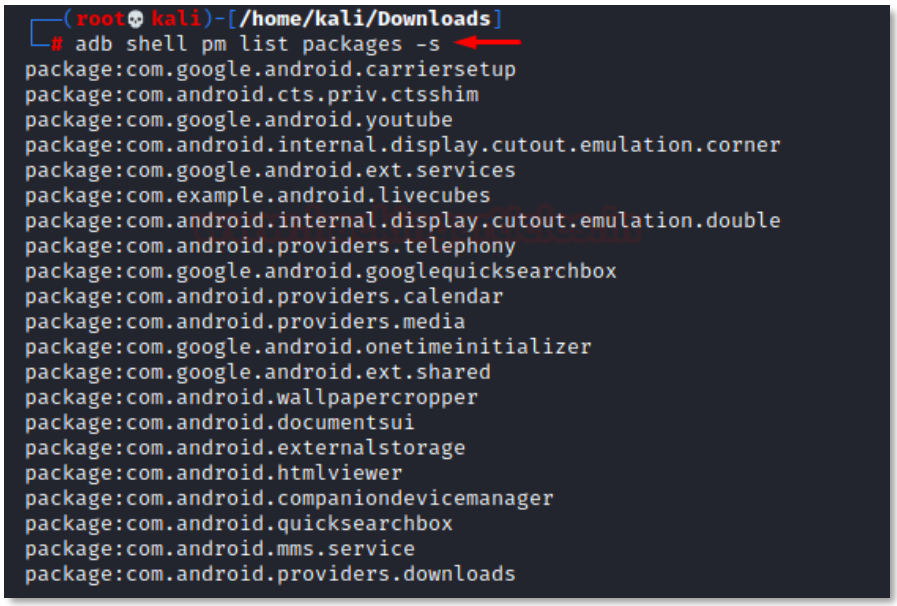
Listing: Package Management in Android refers to the management of all the installed packages/ applications in the android. ex: **DIVA** app's package name is **jakhar.aseem.diva** as visible in the list packages command below:

```
adb shell pm list packages | tail -10
```



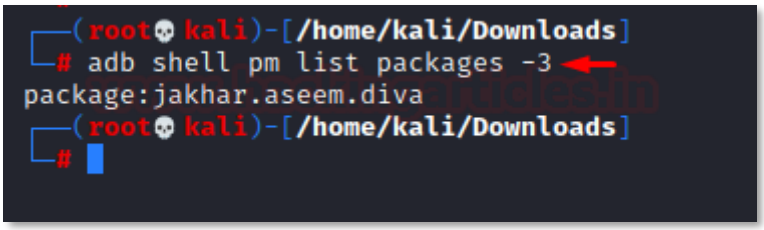
Listing system apps: There are some filters you can apply to sort these packages, like to list all the system apps:

```
adb shell pm list packages -s
```



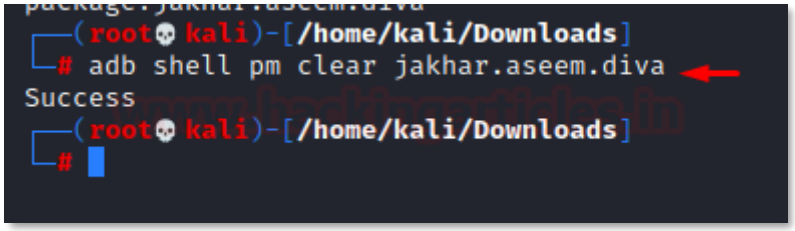
Listing third-party apps: Similarly, to view all the third-party apps, we have the command:

```
adb shell pm list packages -3
```



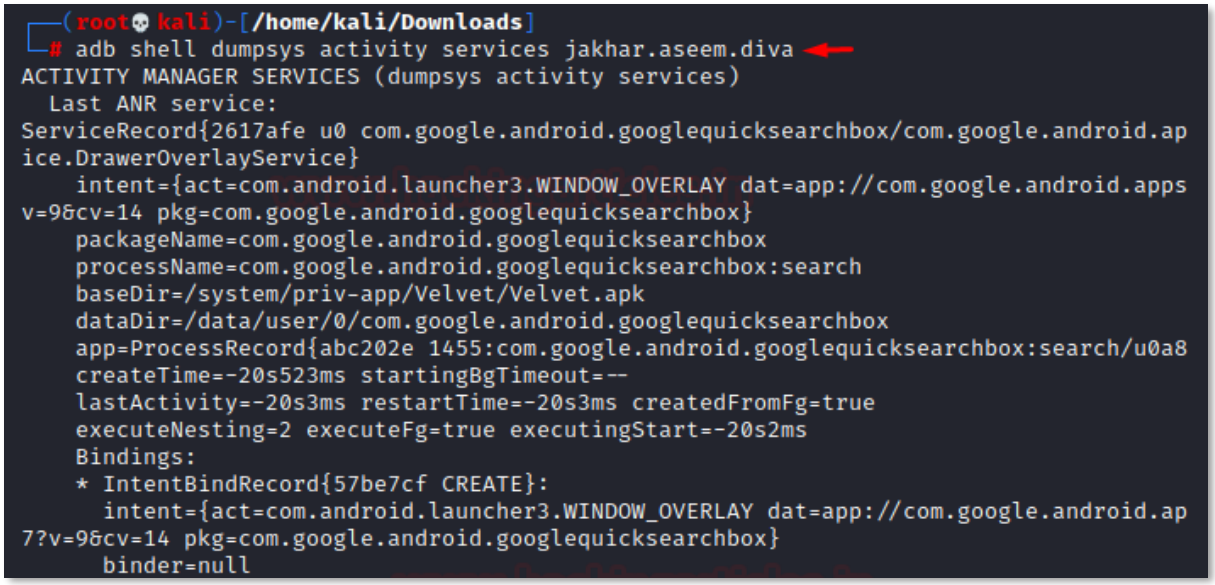
Clearing data of an application: This process is the same as going to the settings and getting rid of its data. To do that we have the following command:

```
adb shell pm clear jakhar.aseem.diva
```



Display installation path of a package: Can be done using the pm tool like:

```
adb shell pm path jakhar.aseem.diva
```

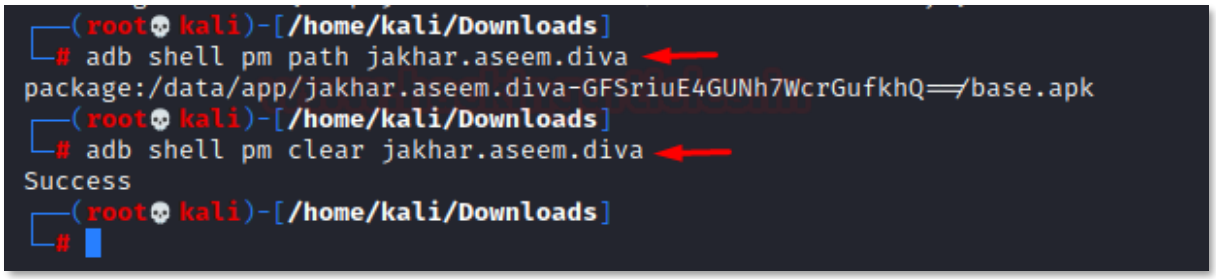


dumpsys tool

View running services in a package: It is done using **dumpsys** command in shell. Dumpsys is a tool that runs on android devices and provides info about system services which can be called from the command line using adb. To view running services in a package:

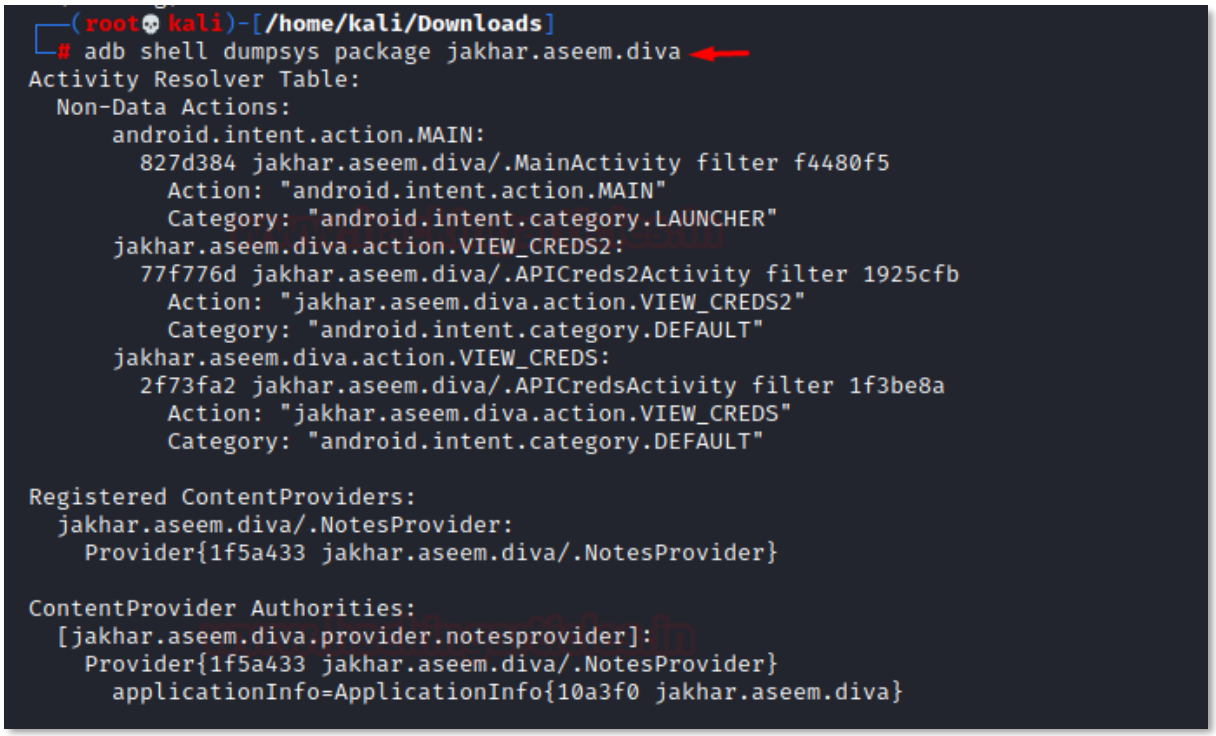
```
adb shell dumpsys activity services
jakhar.aseem.diva
```

It is interesting to note that this displays services related to diva package. If we input this command without a package name all the services will be output.



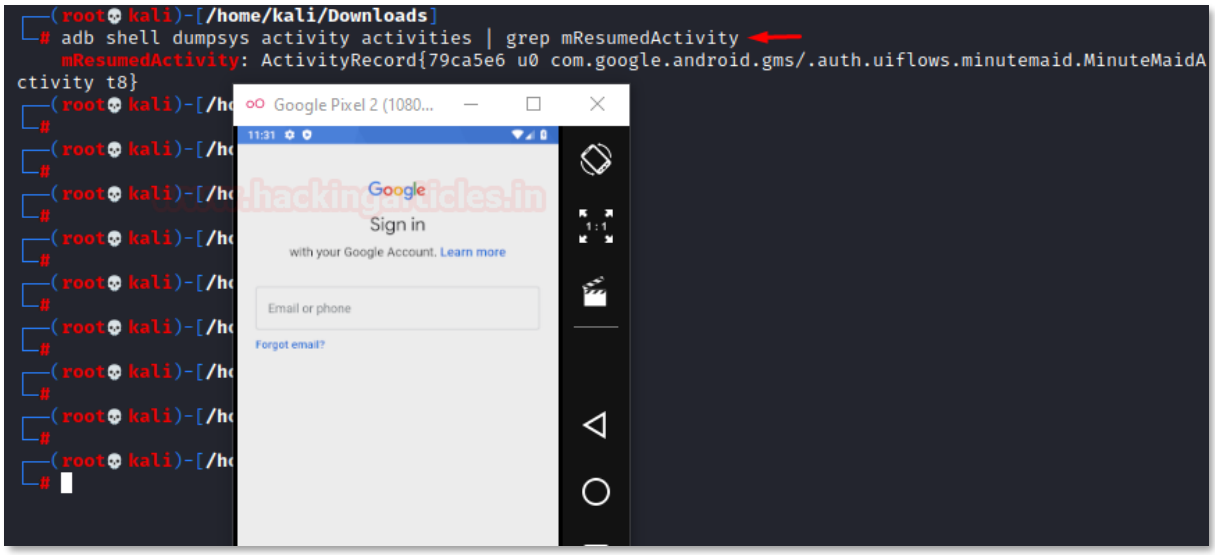
Extracting information about a package: A package will have components like actions, activities, content providers etc as mentioned in the previous article and to view this information about a specific package we use the following command:

```
adb shell dumpsys package jakhar.aseem.diva
```



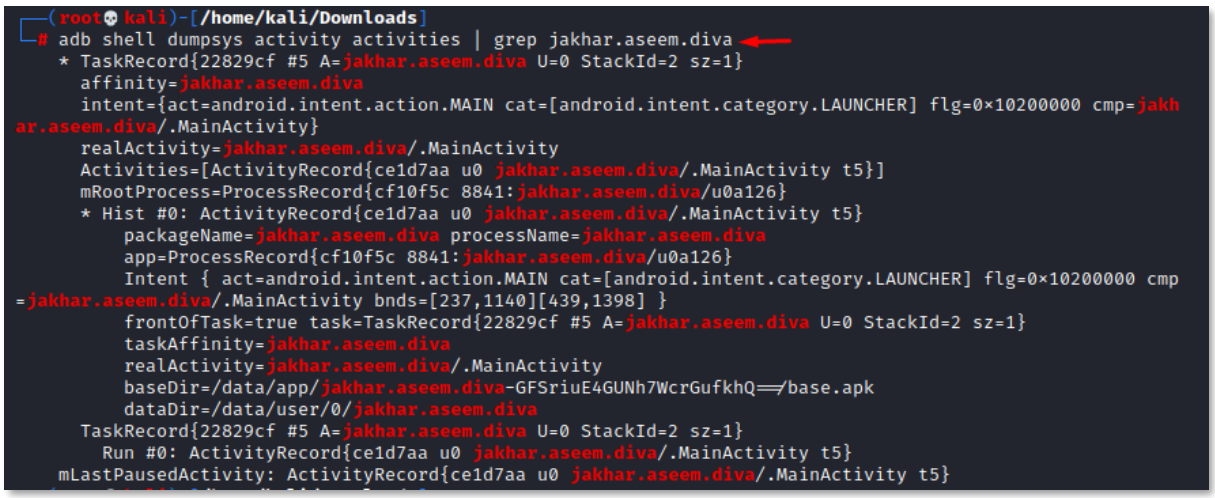
View foreground activity: The activity that is currently on the main screen can be displayed with the following command:

```
adb shell dumpsys activity activities |
grep mResumedActivity
```



Information about activities in a specific package: To view the details of activities like current activity paused, history of all the activities opened etc, type in the following command:

```
adb shell dumpsys activity activities |
grep jakhar.aseem.diva
```



Viewing running services of a package: The package name is optional in this command as that will open the running services related to that package. It is as follows:

```
adb shell dumpsys activity services
jakhar.aseem.diva
```

```
┌──(root💀kali)-[/home/kali/Downloads]
└─# adb shell dumpsys activity services jakhar.aseem.diva
ACTIVITY MANAGER SERVICES (dumpsys activity services)
  Last ANR service:
ServiceRecord{2617afe u0 com.google.android.googlequicksearchbox/com.google.android.apps.
gsa.nowoverlayservice.DrawerOverlayService}
    intent={act=com.android.launcher3.WINDOW_OVERLAY dat=app://com.google.android.apps.ne
xuslauncher:10087?v=9&cv=14 pkg=com.google.android.googlequicksearchbox}
    packageName=com.google.android.googlequicksearchbox
    processName=com.google.android.googlequicksearchbox:search
    baseDir=/system/priv-app/Velvet/Velvet.apk
    dataDir=/data/user/0/com.google.android.googlequicksearchbox
    app=ProcessRecord{abc202e 1455:com.google.android.googlequicksearchbox:search/u0a85}
    createTime=-20s523ms startingBgTimeout=--
    lastActivity=-20s3ms restartTime=-20s3ms createdFromFg=true
    executeNesting=2 executeFg=true executingStart=-20s2ms
    Bindings:
    * IntentBindRecord{57be7cf CREATE}:
      intent={act=com.android.launcher3.WINDOW_OVERLAY dat=app://com.google.android.apps.
nexuslauncher:10087?v=9&cv=14 pkg=com.google.android.googlequicksearchbox}
      binder=null
```

Viewing detailed information about a package: A package has many details ike components, permissions info, version name and code etc. To view these details:

```
adb shell dumpsys package jakhar.aseem.diva
```

```
┌──(root💀kali)-[/home/kali/Downloads]
└─# adb shell dumpsys package jakhar.aseem.diva
Activity Resolver Table:
  Non-Data Actions:
      android.intent.action.MAIN:
        827d384 jakhar.aseem.diva/.MainActivity filter f4480f5
          Action: "android.intent.action.MAIN"
          Category: "android.intent.category.LAUNCHER"
      jakhar.aseem.diva.action.VIEW_CREDS2:
        77f776d jakhar.aseem.diva/.APICreds2Activity filter 1925cfb
          Action: "jakhar.aseem.diva.action.VIEW_CREDS2"
          Category: "android.intent.category.DEFAULT"
      jakhar.aseem.diva.action.VIEW_CREDS:
        2f73fa2 jakhar.aseem.diva/.APICredsActivity filter 1f3be8a
          Action: "jakhar.aseem.diva.action.VIEW_CREDS"
          Category: "android.intent.category.DEFAULT"

Registered ContentProviders:
  jakhar.aseem.diva/.NotesProvider:
    Provider{1f5a433 jakhar.aseem.diva/.NotesProvider}

ContentProvider Authorities:
  [jakhar.aseem.diva.provider.notesprovider]:
    Provider{1f5a433 jakhar.aseem.diva/.NotesProvider}
      applicationInfo=ApplicationInfo{10a3f0 jakhar.aseem.diva}
```

am tool

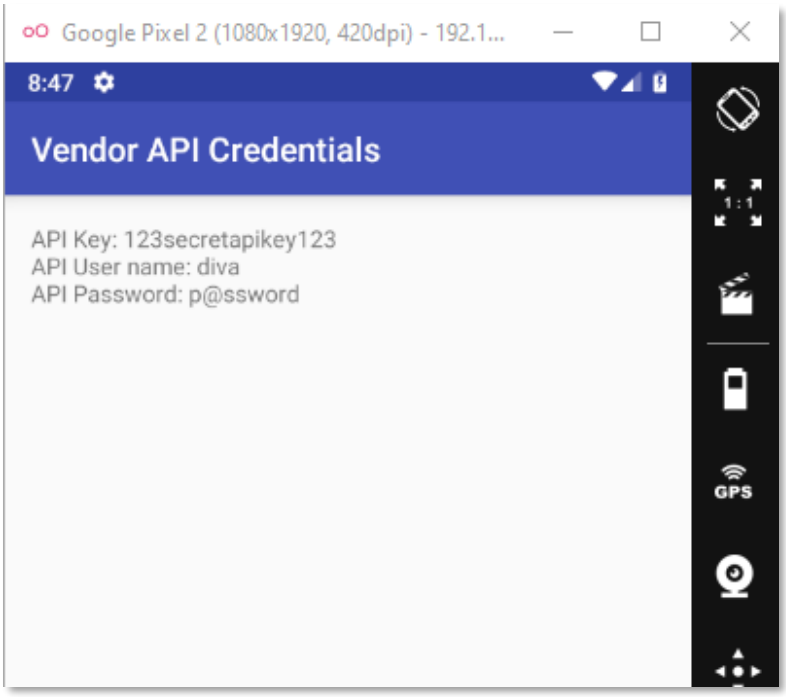
Activities are any page coded in java that performs a specific task. What's interesting is that these activities can also be played around with adb. Many of these functions are performed using another tool called **am** that is installed in android to give information about activities.

Starting Activity: To start an activity, in this case, main activity using adb we type in the following command:

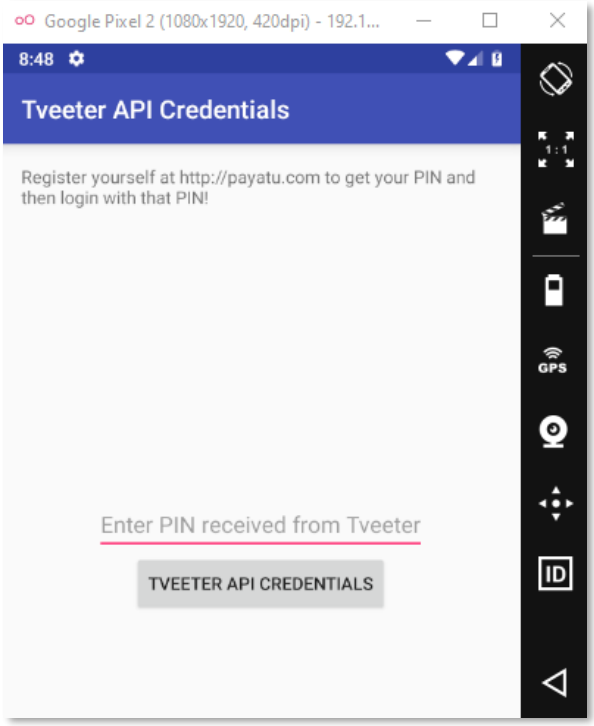
```
adb shell am start -n jakhar.aseem.diva/.APICredsActivity
adb shell am start -n jakhar.aseem.diva/.APICreds2Activity
```

```
┌──(root💀 kali)-[/home/kali/Downloads]
└─# adb shell am start -n jakhar.aseem.diva/.APICredsActivity
Starting: Intent { cmp=jakhar.aseem.diva/.APICredsActivity }
┌──(root💀 kali)-[/home/kali/Downloads]
└─# adb shell am start -n jakhar.aseem.diva/.APICreds2Activity
Starting: Intent { cmp=jakhar.aseem.diva/.APICreds2Activity }
┌──(root💀 kali)-[/home/kali/Downloads]
└─#
```

This would open up APICredsActivity activity like this:



And the other activity too:



Start/Stop Service: To stop a service we use the following command:

```
adb shell am stopservice -n
com.android.systemui/.SystemUIService
```

This command will stop the System UI. There would be a blank screen until we start it again using this command:

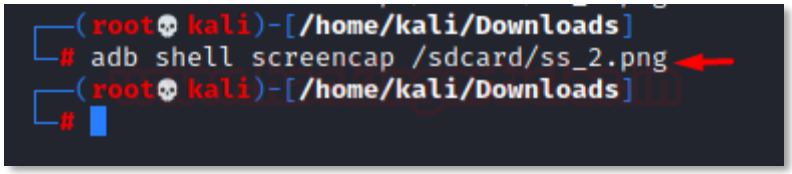
```
adb shell am startservice -n
com.android.systemui/.SystemUIService
```

```
┌──(root💀kali)-[/home/kali/Downloads]
└─# adb shell am stopservice -n com.android.systemui/.SystemUIService
Stopping service: Intent { cmp=com.android.systemui/.SystemUIService }
Service stopped
┌──(root💀kali)-[/home/kali/Downloads]
└─# adb shell am startservice -n com.android.systemui/.SystemUIService
Starting service: Intent { cmp=com.android.systemui/.SystemUIService }
```

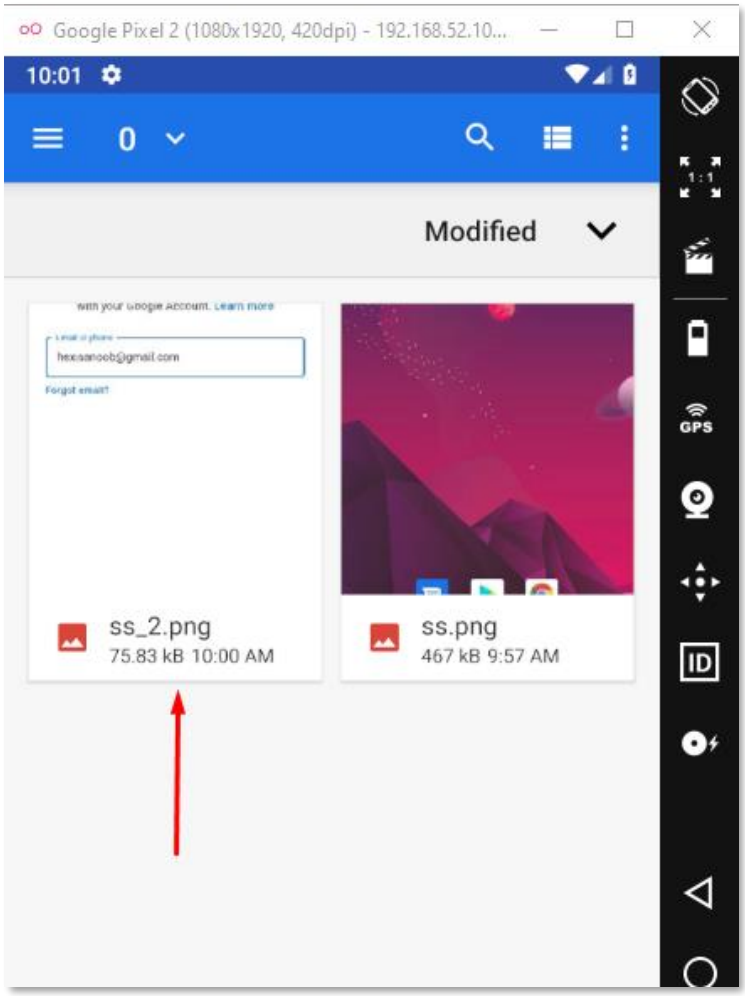
Miscellaneous

Capturing a screenshot: screencap is a handy tool installed in android to take screenshots. This tool is running behind the screenshot feature and can be called using adb like:

```
adb shell screencap /sdcard/ss_2.png
```



2nd argument is the local path where the screenshot will be stored and we can customise it. In this case, I've put in **sdcard/** as the path and we traverse to that folder to see ss_2.png there:

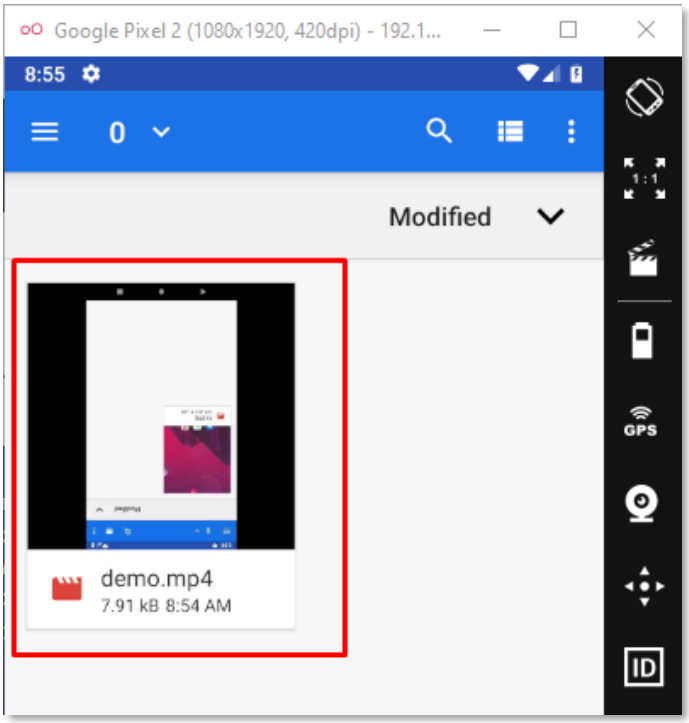


Recording the screen: Just like screenshots can be taken, a video recording of the screen is also possible using the **screenrecord** tool in android with the following command:

```
adb shell screenrecord /sdcard/demo.mp4 --size 400x400 --
    bit-rate 200000 --time-limit 5 --rotate --verbose
```

```
┌──(root💀kali)-[/home/kali/Downloads]
└─# adb shell screenrecord /sdcard/demo.mp4 --size 400×400 --bit-rate 200000 --time-limit 5 --r
otate --verbose
Main display is 1080×1920 @60.00fps (orientation=0)
Configuring recorder for 400×400 video/avc at 0.20Mbps
Rotated content area is 400×225 at offset x=0 y=87
Time limit reached
Encoder stopping; recorded 1 frames in 5 seconds
Stopping encoder and muxer
Executing: /system/bin/am broadcast -a android.intent.action.MEDIA_SCANNER_SCAN_FILE -d file://
/sdcard/demo.mp4
Broadcasting: Intent { act=android.intent.action.MEDIA_SCANNER_SCAN_FILE dat=file:///sdcard/dem
o.mp4 flg=0×400000 }
Broadcast completed: result=0
┌──(root💀kali)-[/home/kali/Downloads]
└─#
```

/sdcard/demo.mp4 – Path to store the recording
size – The size in pixelxpixel (heightxwidth) in which the video will be recorded
bit-rate – Corresponds to the quality of the video in short
time-limit – Max time to which screen will be recorded
rotate – Makes default portrait orientation upside down and vice versa in the recording. Can be customised for specific angles.
We'll see output like the following:

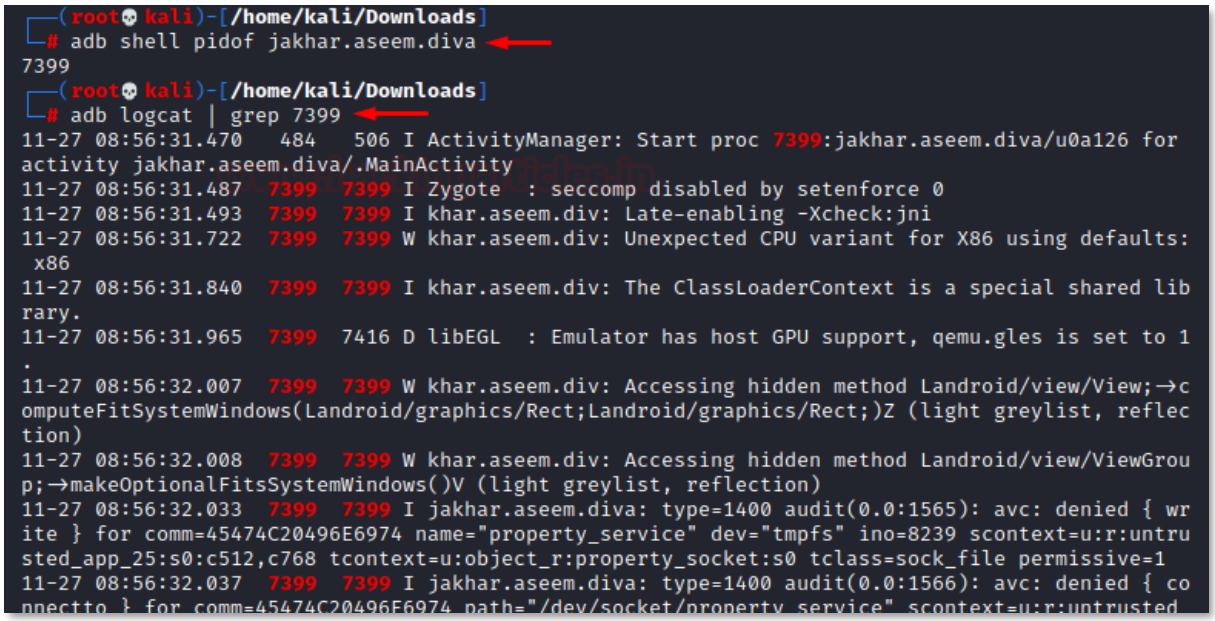


Viewing process ID of a package: It is important from a pentester's point of view to know the process ID of an application package. This can be done by:

```
adb shell pidof jakhar.aseem.diva
```

For example, I have to inspect the logs of this package, I can filter it out using grep and the PID like:

```
adb logcat | grep 7399
```



Viewing Battery Status: All the information, from voltage to level can be dumped using dumpsys battery command:

```
adb shell dumpsys battery
```

We can play around with the battery level by the following command:

```
adb shell dumpsys battery set level 0
```

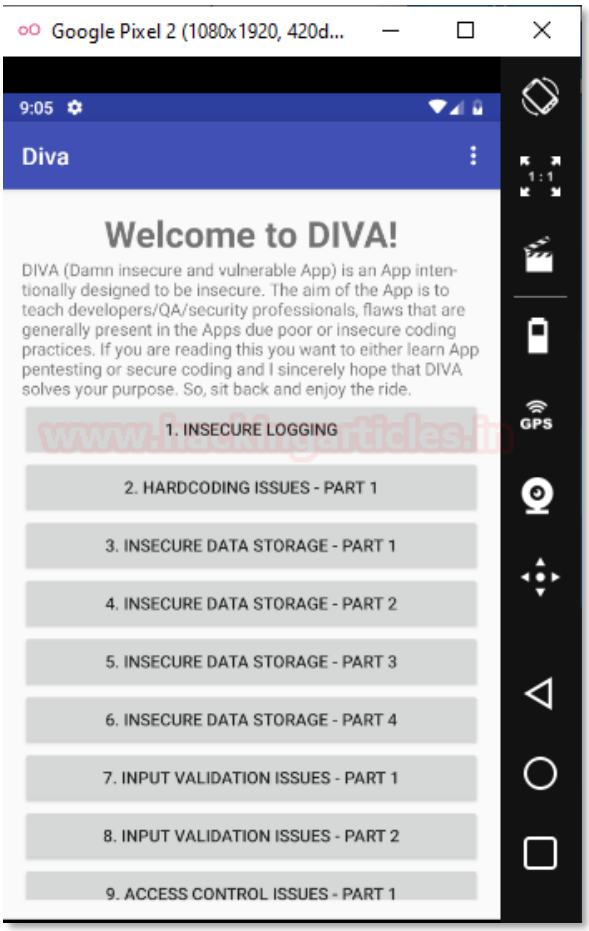
```
┌──(root💀kali)-[/home/kali/Downloads]
└─# adb shell dumpsys battery
Current Battery Service state:
  AC powered: true
  USB powered: false
  Wireless powered: false
  Max charging current: 0
  Max charging voltage: 0
  Charge counter: 0
  status: 2
  health: 1
  present: true
  level: 93
  scale: 100
  voltage: 10000
  temperature: 0
  technology: Unknown
┌──(root💀kali)-[/home/kali/Downloads]
└─# adb shell dumpsys battery set level 0
┌──(root💀kali)-[/home/kali/Downloads]
└─#
```

Sure enough, let's see how the battery now looks like:

Keyevents in Android: Each key in Android is described by a sequence of key events. These can be described as follows:

S.No.	Keycode	Meaning
1	3	HOME Button
2	4	Return Key
3	5	Open Dialing Application
4	6	Hang Up
5	24	Increase V
6	25	olume
7	26	Volume Down
8	27	Power button
9	64	Taking Pictures (in the camera application needs in)
10	82	Open Browser
11	85	Menu Button

Let's try out keyevent 3. Here is a page with a random activity opened up:



Now, the command to send key event 3 is:

```
adb shell input keyevent 3
```

```
┌──(root💀kali)-[/home/kali/Downloads]
└─# adb shell input keyevent 3
┌──(root💀kali)-[/home/kali/Downloads]
└─#
```

After the command, a home button has been pressed and the screen looks like:

Viewing kernel logs: To view kernel logs for testing purposes we can type in the following command:

```
adb shell dmesg
```

The list would be too long so I'll just view first 20 logs:

```
adb shell dmesg | head -20
```

```
┌──(root💀 kali)-[/home/kali/Downloads]
└─# adb shell dmesg | head -20
[ 2161.432947] type=1400 audit(1606485022.193:1416): avc: denied { read } for pid=557 com
m="health@2.0-serv" name="capacity" dev="fuse" ino=8 scontext=u:r:hal_health_default:s0 t
context=u:object_r:fuse:s0 tclass=file permissive=1
[ 2161.432987] type=1400 audit(1606485022.193:1416): avc: denied { read } for pid=557 com
m="health@2.0-serv" name="capacity" dev="fuse" ino=8 scontext=u:r:hal_health_default:s0 t
context=u:object_r:fuse:s0 tclass=file permissive=1
[ 2161.432991] type=1400 audit(1606485022.193:1417): avc: denied { open } for pid=557 com
m="health@2.0-serv" path="/dev/pipe/battery/BAT0/capacity" dev="fuse" ino=8 scontext=u:r:
hal_health_default:s0 tcontext=u:object_r:fuse:s0 tclass=file permissive=1
[ 2161.537851] ieee80211 phy0: hw scan 5580 MHz
[ 2161.658181] ieee80211 phy0: hw scan 5600 MHz
[ 2161.778817] ieee80211 phy0: hw scan 5620 MHz
[ 2161.897841] ieee80211 phy0: hw scan 5640 MHz
[ 2162.018341] ieee80211 phy0: hw scan 5660 MHz
[ 2162.138163] ieee80211 phy0: hw scan 5680 MHz
[ 2162.259060] ieee80211 phy0: hw scan 5700 MHz
[ 2162.378277] ieee80211 phy0: hw scan 5745 MHz
[ 2162.497902] ieee80211 phy0: hw scan 5765 MHz
[ 2162.618430] ieee80211 phy0: hw scan 5785 MHz
[ 2162.737921] ieee80211 phy0: hw scan 5805 MHz
[ 2162.858875] ieee80211 phy0: hw scan 5825 MHz
[ 2162.977994] ieee80211 phy0: hw scan complete
[ 2162.978238] ieee80211 phy0: mac80211_hwsim_get_survey (idx=0)
[ 2162.978244] ieee80211 phy0: mac80211_hwsim_get_survey (idx=1)
[ 2162.979239] type=1400 audit(1606485022.193:1417): avc: denied { open } for pid=557 com
m="health@2.0-serv" path="/dev/pipe/battery/BAT0/capacity" dev="fuse" ino=8 scontext=u:r:
hal_health_default:s0 tcontext=u:object_r:fuse:s0 tclass=file permissive=1
[ 2162.979265] type=1400 audit(1606485023.741:1422): avc: denied { read } for pid=325 com
m="hostapd" scontext=u:r:execns:s0 tcontext=u:r:execns:s0 tclass=netlink_generic_socket p
ermissive=1
┌──(root💀 kali)-[/home/kali/Downloads]
└─#
```

Enumerating device:

To find out the model of the phone you're connected to getprop tool in android is usable:

```
adb shell getprop ro.product.model
```

wm tool– wm tool in android relates and gives information about windows. To find out the window resolution we use the following command:

adb shell wm size

To find information about display screen like DPI, Display ID number etc we type in the following command:

```
adb shell dumpsys window displays
```

```
┌──(root💀kali)-[/home/kali/Downloads]
└─# adb shell getprop ro.product.model
Google Pixel 2
┌──(root💀kali)-[/home/kali/Downloads]
└─# adb shell wm size
Physical size: 1080×1920
┌──(root💀kali)-[/home/kali/Downloads]
└─# adb shell wm density
Physical density: 420
┌──(root💀kali)-[/home/kali/Downloads]
└─# adb shell dumpsys window displays
WINDOW MANAGER DISPLAY CONTENTS (dumpsys window displays)
  Display: mDisplayId=0
    init=1080×1920 420dpi cur=1080×1920 app=1080×1794 rng=1080×1017-1794×1731
    deferred=false mLayoutNeeded=true mTouchExcludeRegion=SkRegion((0,0,1080,1

  mLayoutSeq=400

  Application tokens in top down Z order:
    mStackId=0
    mDeferRemoval=false
    mBounds=[0,0][1080,1920]
      taskId=23
        mBounds=[0,0][1080,1920]
        mdr=false
        appTokens=[AppWindowToken{a821c49 token=Token{84d250 ActivityRecord{9b
```

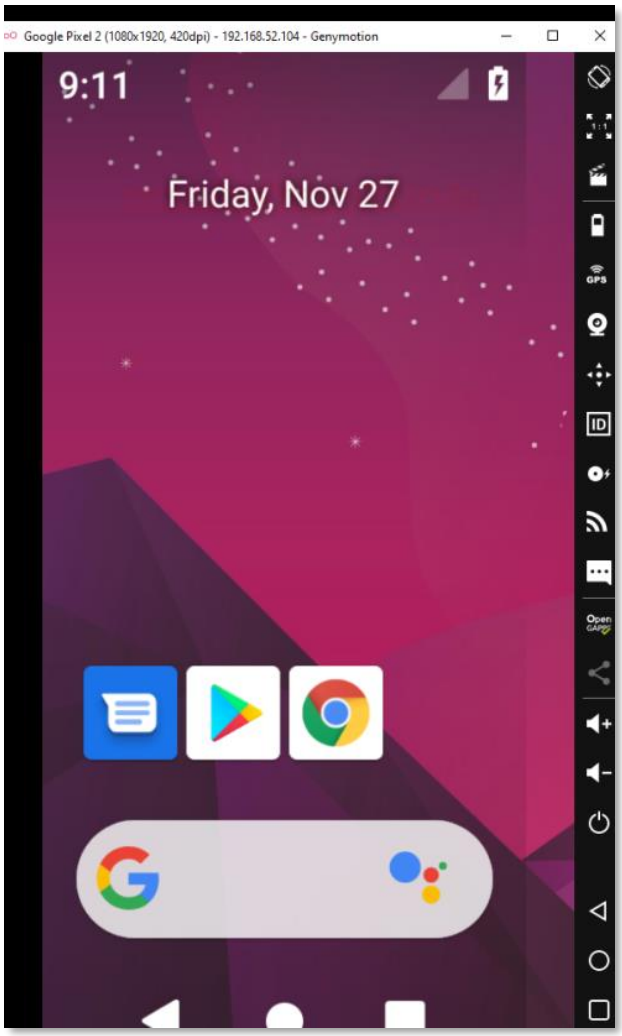
We can modify the screen resolution as well with wm tool using the following command:

```
adb shell wm size 480x1024
adb reboot
```

where the size is in pixels.

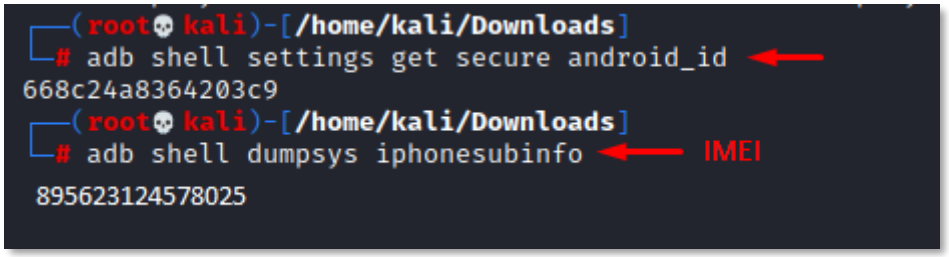
```
┌──(root💀kali)-[/home/kali/Downloads]
└─# adb shell wm size 480×1024
┌──(root💀kali)-[/home/kali/Downloads]
└─# adb reboot
┌──(root💀kali)-[/home/kali/Downloads]
└─#
```

After reboot the window will look something like this:



Secure Android ID and IMEI number: Secure Android ID is a 64 bit number that is generated on the first boot. This ID is also available in the settings. This remains constant. On the other hand, IMEI number is a unique telephony number. They can be viewed with the following commands:

```
adb shell settings get secure android_id
adb shell dumpsys iphonesubinfo
```



Reference

- https://www.hackingarticles.in/android-application-framework-beginners-guide/
- https://www.hackingarticles.in/android-pentest-lab-setup-adb-command-cheatsheet/

About Us

About Us

"Simple training makes Deep Learning"

"IGNITE" is a worldwide name in IT field. As we provide high-quality cybersecurity training and consulting services that fulfil students, government and corporate requirements.
We are working towards the vision to "Develop India as a Cyber Secured Country". With an outreach to over eighty thousand students and over a thousand major colleges, Ignite Technologies stood out to be a trusted brand in the Education and the Information Security structure.

We provide training and education in the field of Ethical Hacking & Information Security to the students of schools and colleges along with the corporate world. The training can be provided at the client's location or even at Ignite's Training Center.
We have trained over 10,000 + individuals across the globe, ranging from students to security experts from different fields. Our trainers are acknowledged as Security Researcher by the Top Companies like - Facebook, Google, Microsoft, Adobe, Nokia, Paypal, Blackberry, AT&T and many more. Even the trained students are placed into a number of top MNC's all around the globe. Over with this, we are having International experience of training more than 400+ individuals.

The two brands, Ignite Technologies & Hacking Articles have been collaboratively working from past 10+ Years with about more than 100+ security researchers, who themselves have been recognized by several research paper publishing organizations, The Big 4 companies, Bug Bounty research programs and many more.

Along with all these things, all the major certification organizations recommend Ignite's training for its resources and guidance.
Ignite's research had been a part of number of global Institutes and colleges, and even a multitude of research papers shares Ignite's researchers in their reference.

What We Offer

Ethical Hacking

The Ethical Hacking course has been structured in such a way that a technical or a non-technical applicant can easily absorb its features and indulge his/her career in the field of IT security.

Bug Bounty 2.0

A bug bounty program is a pact offered by many websites and web developers by which folks can receive appreciation and reimbursement for reporting bugs, especially those affecting to exploits and vulnerabilities.
Over with this training, an indivisual is thus able to determine and report bugs to the authorized before the general public is aware of them, preventing incidents of widespread abuse.

Network Penetration Testing 2.0

The Network Penetration Testing training will build up the basic as well advance skills of an indivisual with the concept of Network Security & Organizational Infrastructure. Thereby this course will make the indivisual stand out of the crowd within just 45 days.

Red Teaming

This training will make you think like an "Adversary" with its systematic structure & real Environment Practice that contains more than 75 practicals on Windows Server 2016 & Windows 10. This course is especially designed for the professionals to enhance their Cyber Security Skills

CTF 2.0

The CTF 2.0 is the latest edition that provides more advance module connecting to real infrastructure organization as well as supporting other students preparing for global certification. This curriculum is very easily designed to allow a fresher or specialist to become familiar with the entire content of the course.

Infrastructure Penetration Testing

This course is designed for Professional and provides an hands-on experience in Vulnerability Assessment Penetration Testing & Secure configuration Testing for Applications Servers, Network Deivces, Container and etc.

Digital Forensic

Digital forensics provides a taster in the understanding of how to conduct investigations in order for business and legal audien ces to correctly gather and analyze digital evidence.