iGNITE Technologies

Android Penetration Testing

ADB Cheat Sheet

WWW.HACKINGARTICLES.IN

Contents

Introduction.....3

Installation of Virtual Environment 3

Getting Started With Debugging.....15

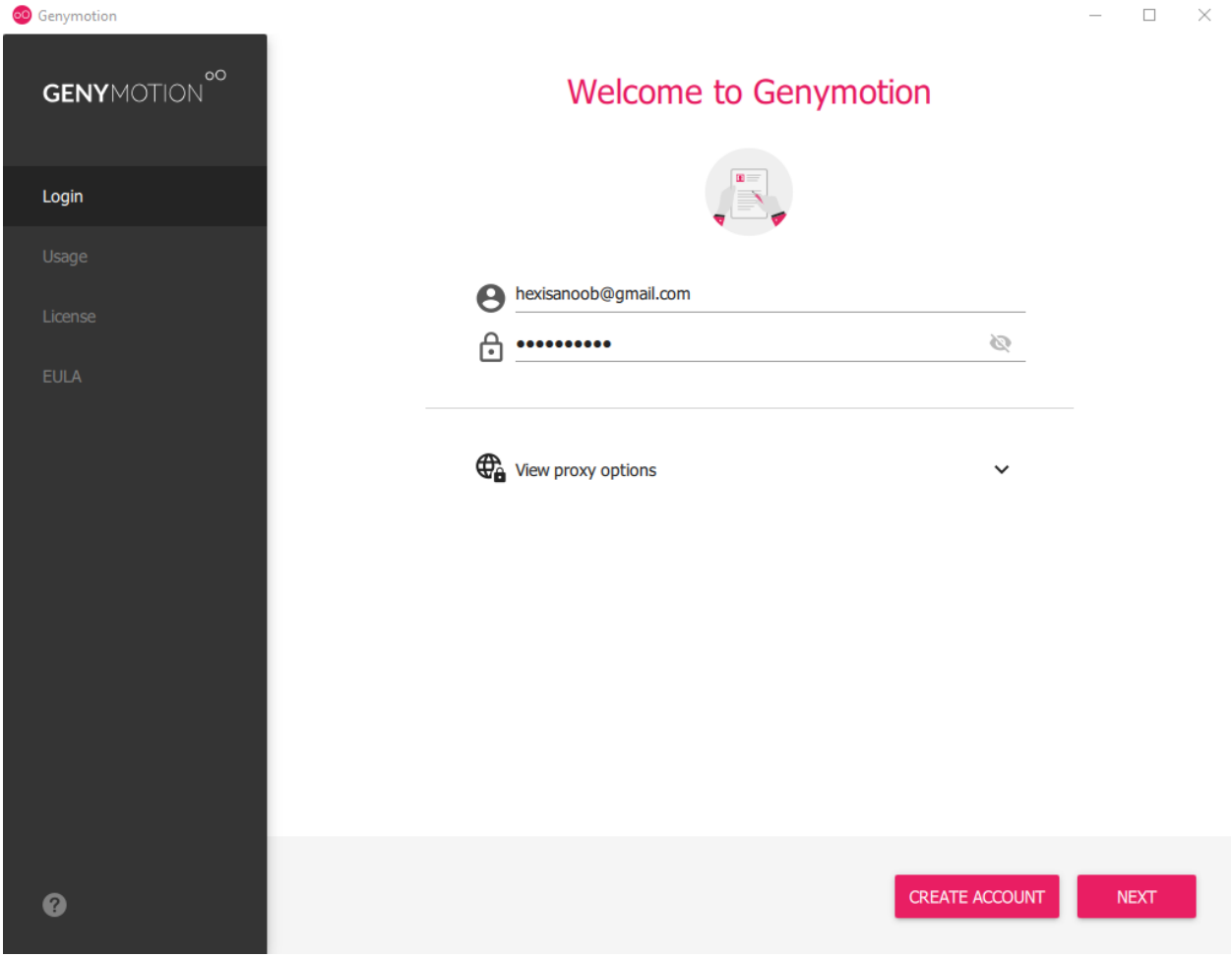
Conclusion 39

Introduction

To learn android pentest in a much handier way we'll be setting up Android Pentest environment in our own system rather than conducting an experiment on a live device. It is to be noted that these practicals can be conducted on a phone with USB debugging option on as well, but we'll be using an Android virtual machine, also known as, Android Emulator. Genymotion is one example of android emulators that are available to download in the market that can be very helpful for Android penetration testing. It is a preferred choice while setting up a lab for android pen testing because it is easy to setup, offers ADB support and Google Apps support. We'll be using Genymotion to demonstrate all of our examples in this series, but an individual can use any other emulator that provides basic functionality support as Genymotion does. Let's begin.

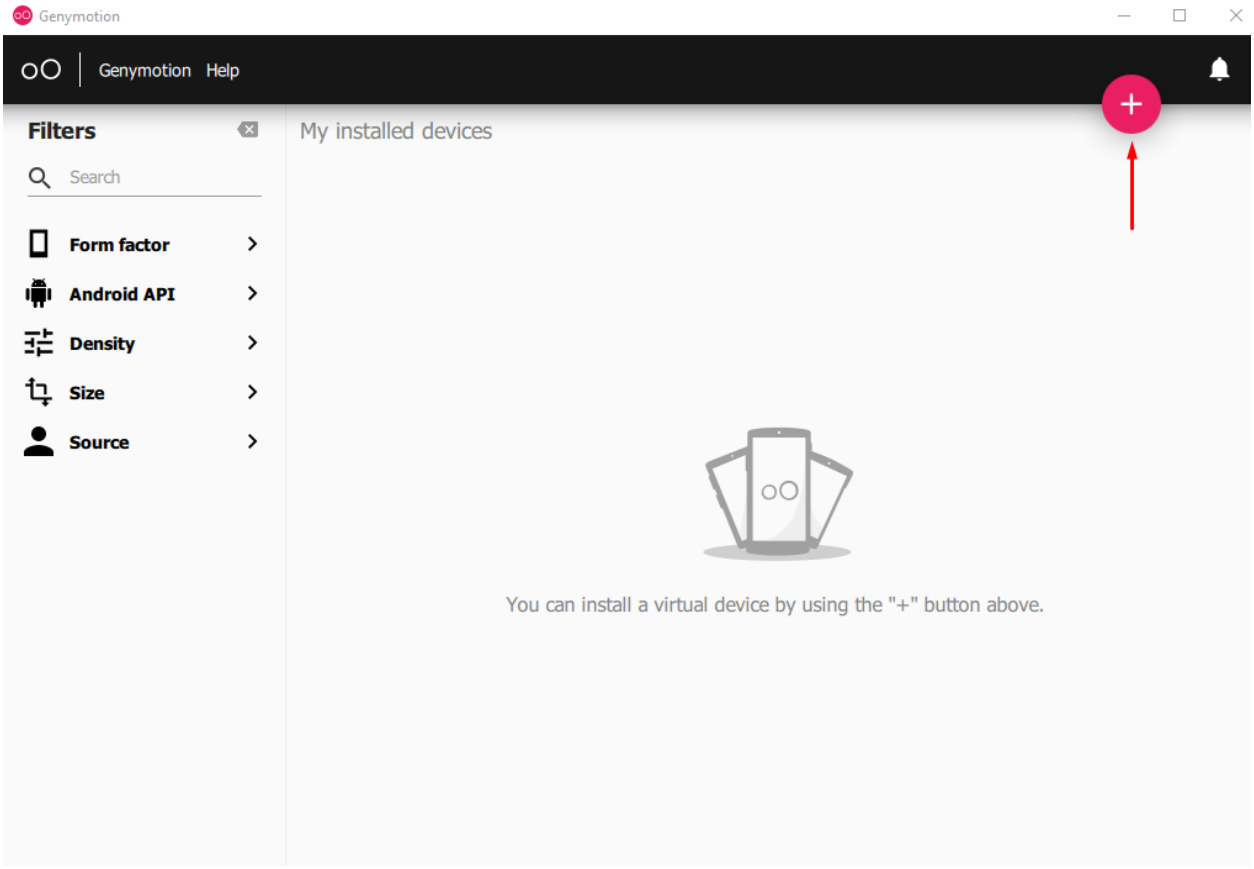
Installation of Virtual Environment

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.

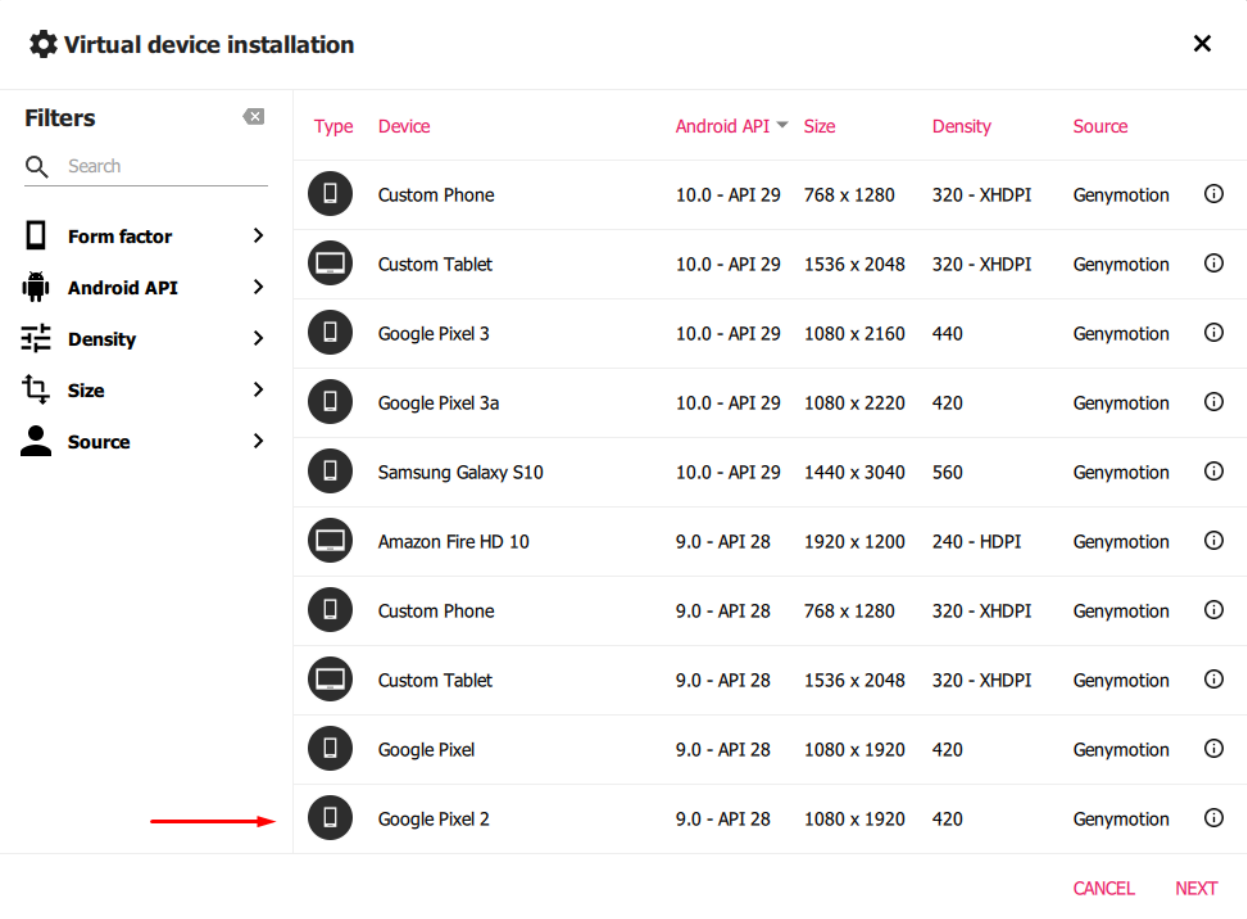


Creation of Android Virtual Machine (AVM)

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.**

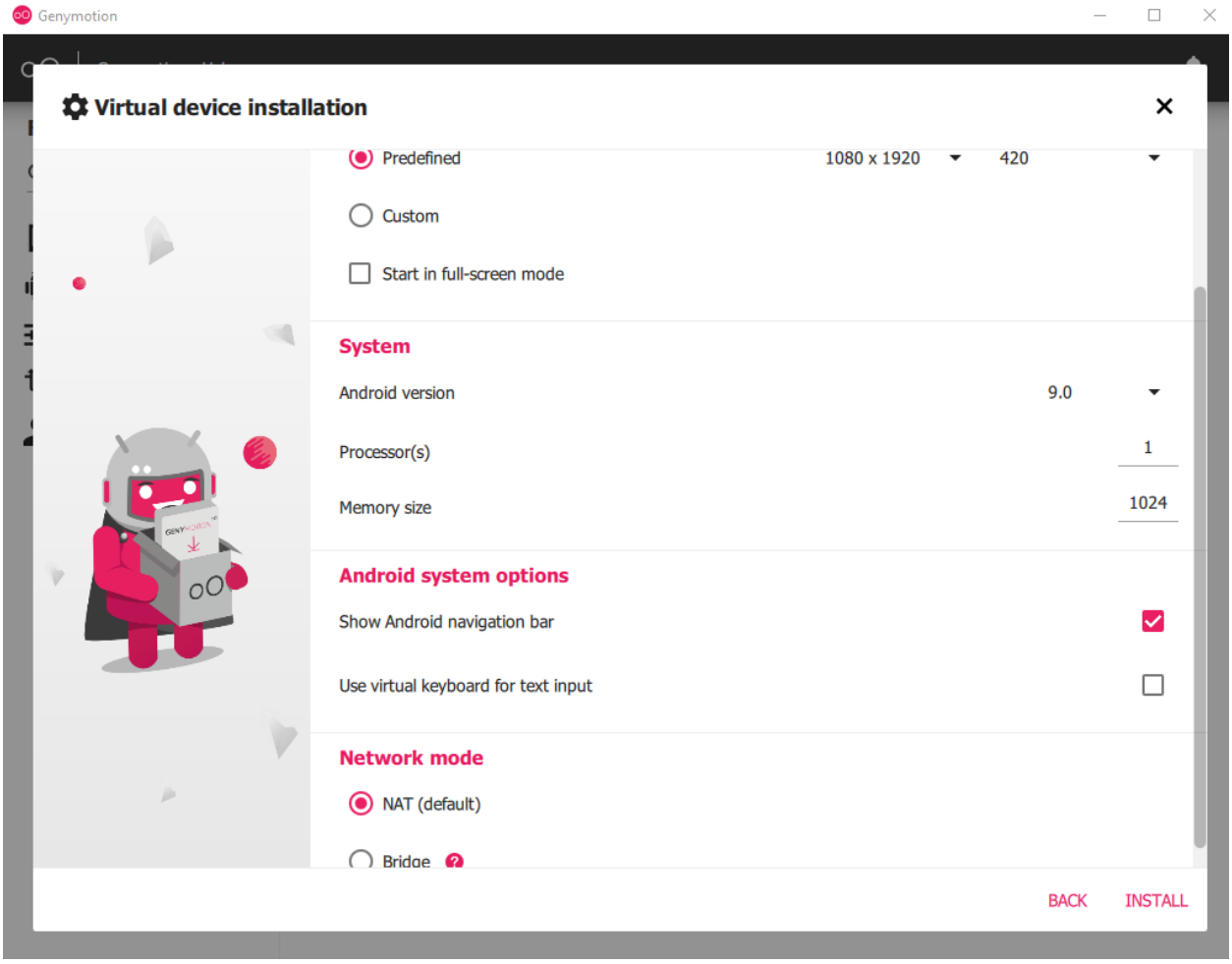


What is Android APIs? Android API refers to the collection of various software modules having different versions that make an Android SDK. In simpler words, A specific API version will correspond to a specific Android release as follows: –

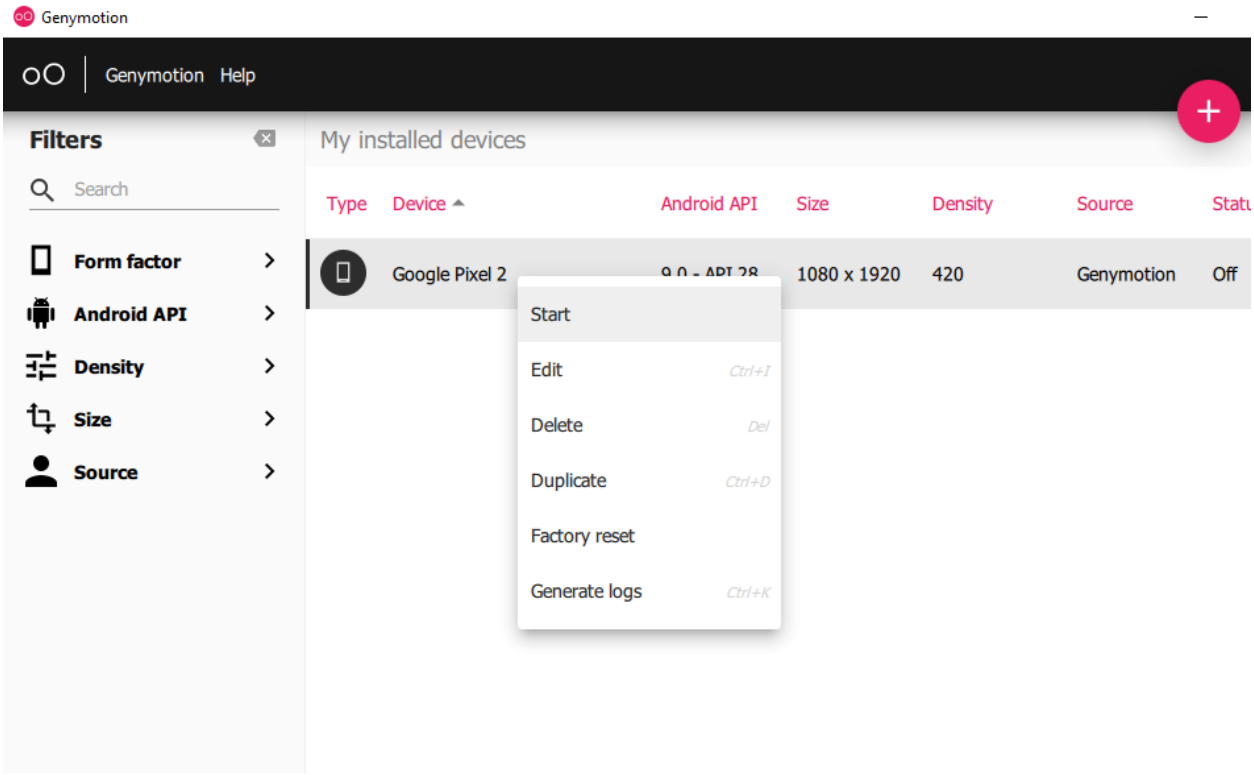
Platform Version	API Level	VERSION_CODE
Android 11	30	R
Android 10	29	Q
Android 9	28	Pie
Android 8.1	27	Oreo_MR1
Android 8.0	26	Oreo
Android 7.1,7.1.1	25	N_MR1
Android 7.0	24	N
Android 6.0	23	M
Android 5.1	22	LOLLIPOP_MR1
Android 5.0	21	LOLLIPOP
Android 4.4W	20	KITKAT_WATCH
Android 4.4	19	KITKAT
Android 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.

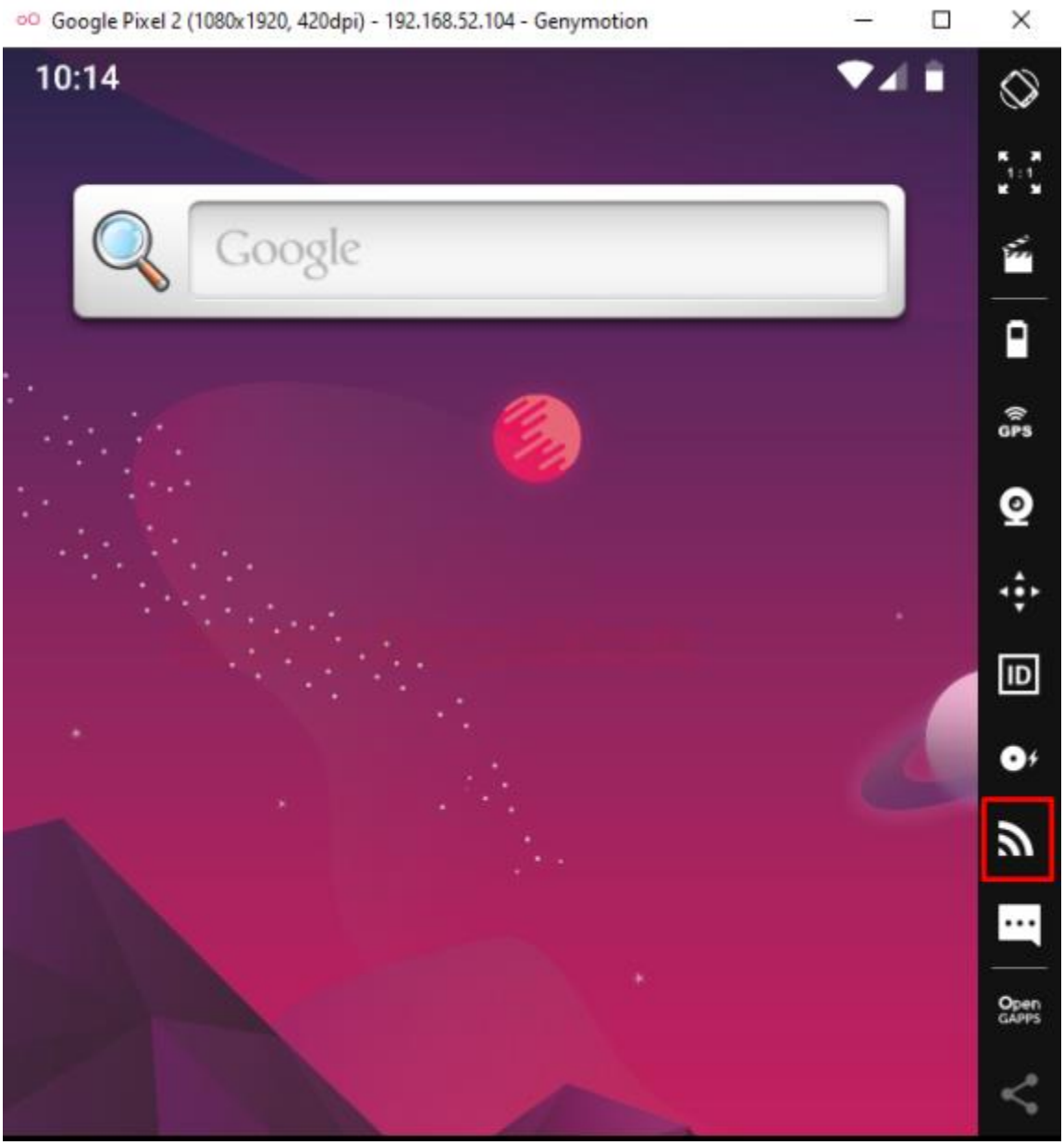
Next step is to configure cores, RAM and networking mode. I'll be setting up 1 core for processing, 1GB of RAM and NAT mode as follows



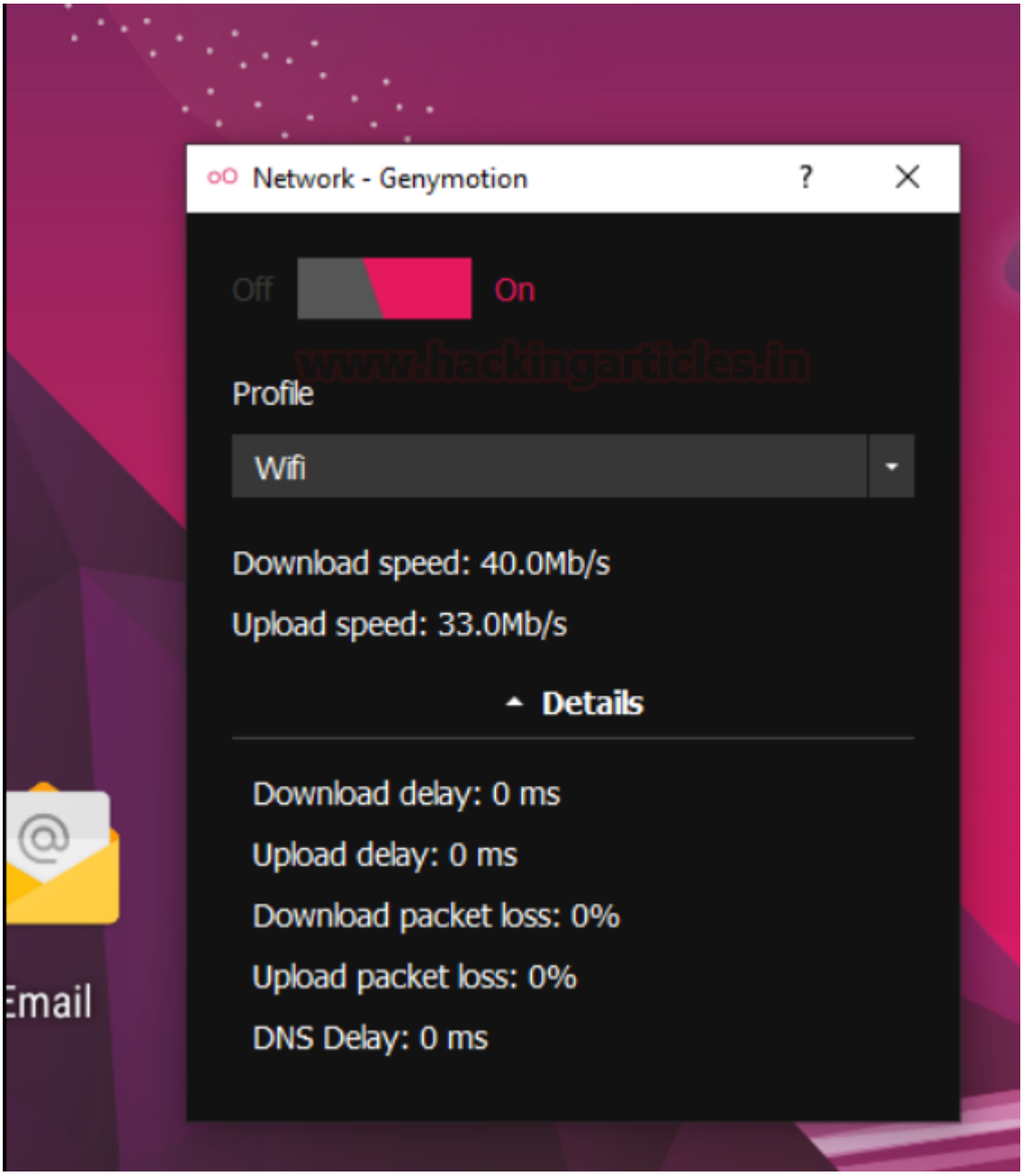
Once the install button is clicked, the respective API will first be downloaded if not already available in the SDK and then the machine will be ready to launch like following. Right click on it and press start



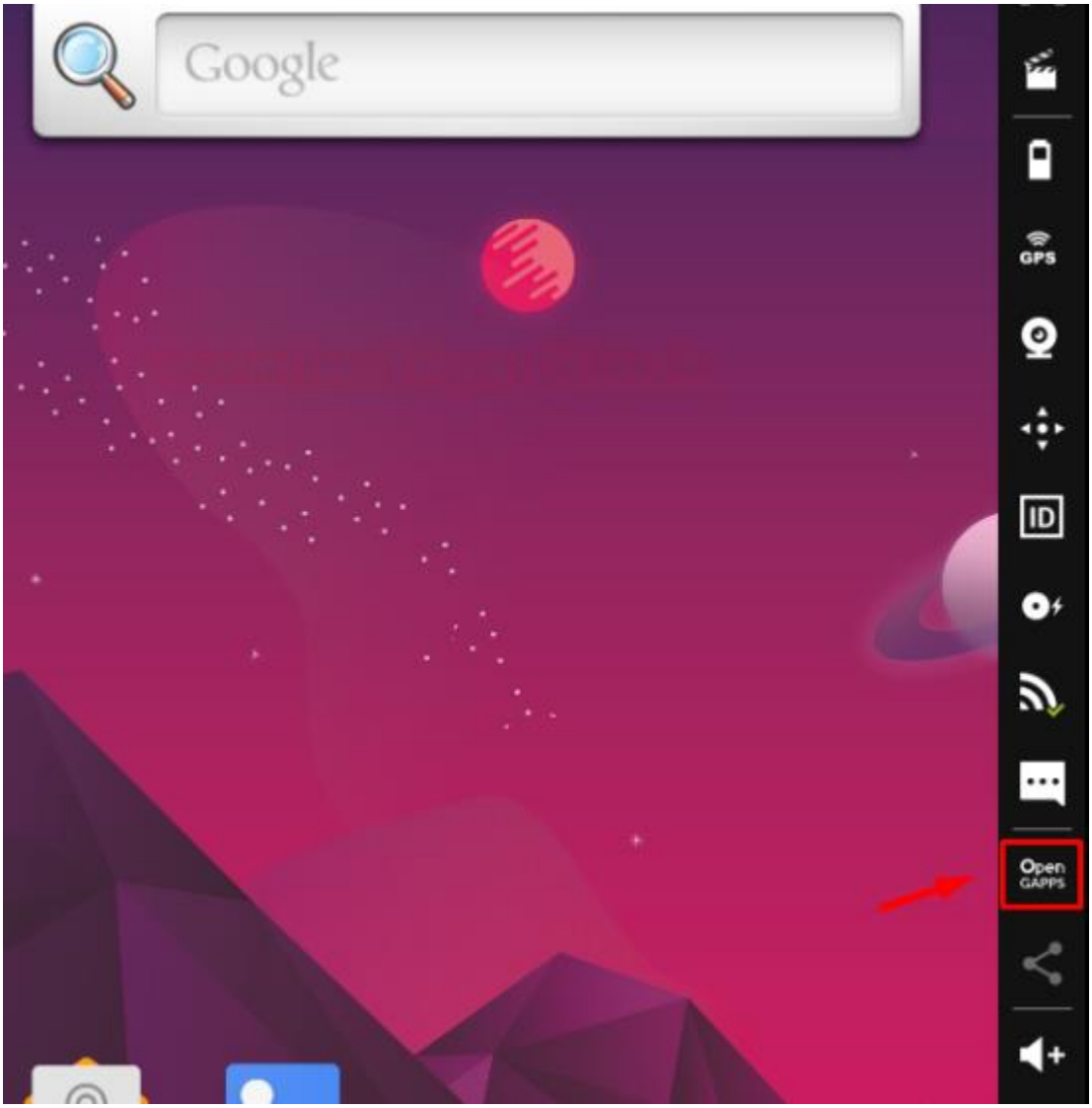
This will initiate a virtual box instance under genymotion's window and the first thing that we'll do is start networking on the virtual machine by clicking the signal icon.



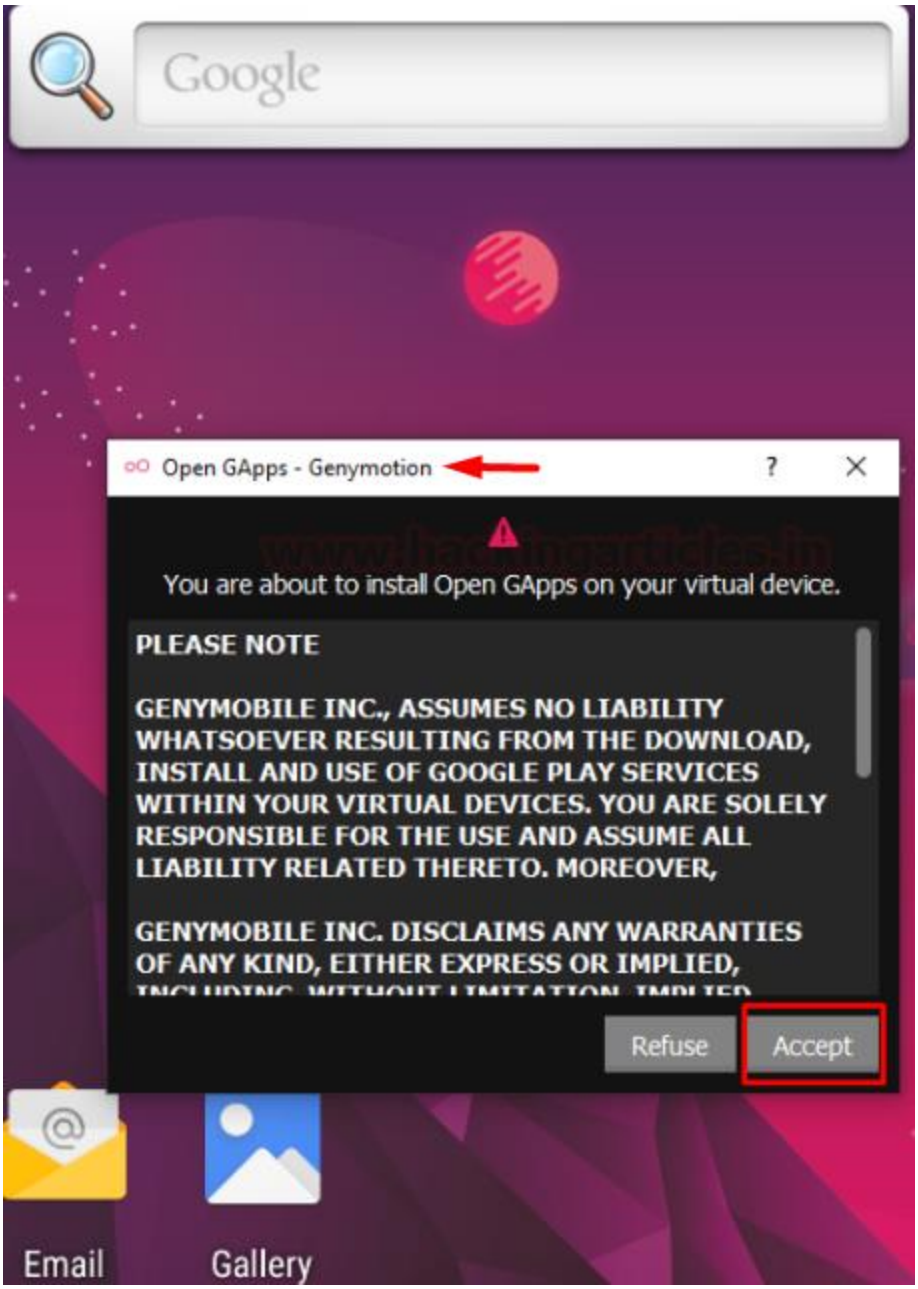
next is to turn the switch to ON and choose the respective networking adapter you want to allocate. In my case a Wi-Fi connection is active so I'll be allocating Wi-Fi to the machine.



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.

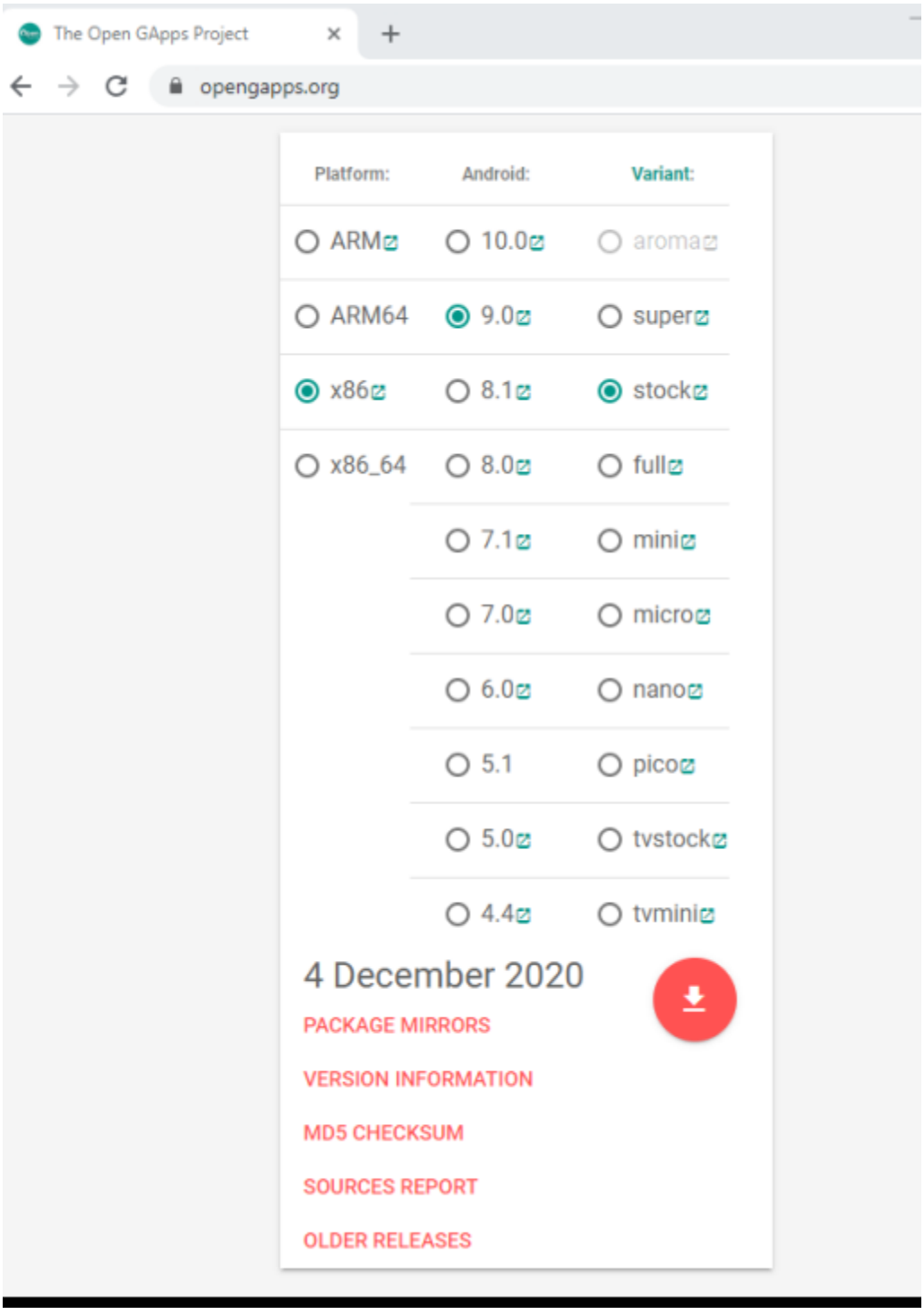


Simply accept the agreement and we're good to go.

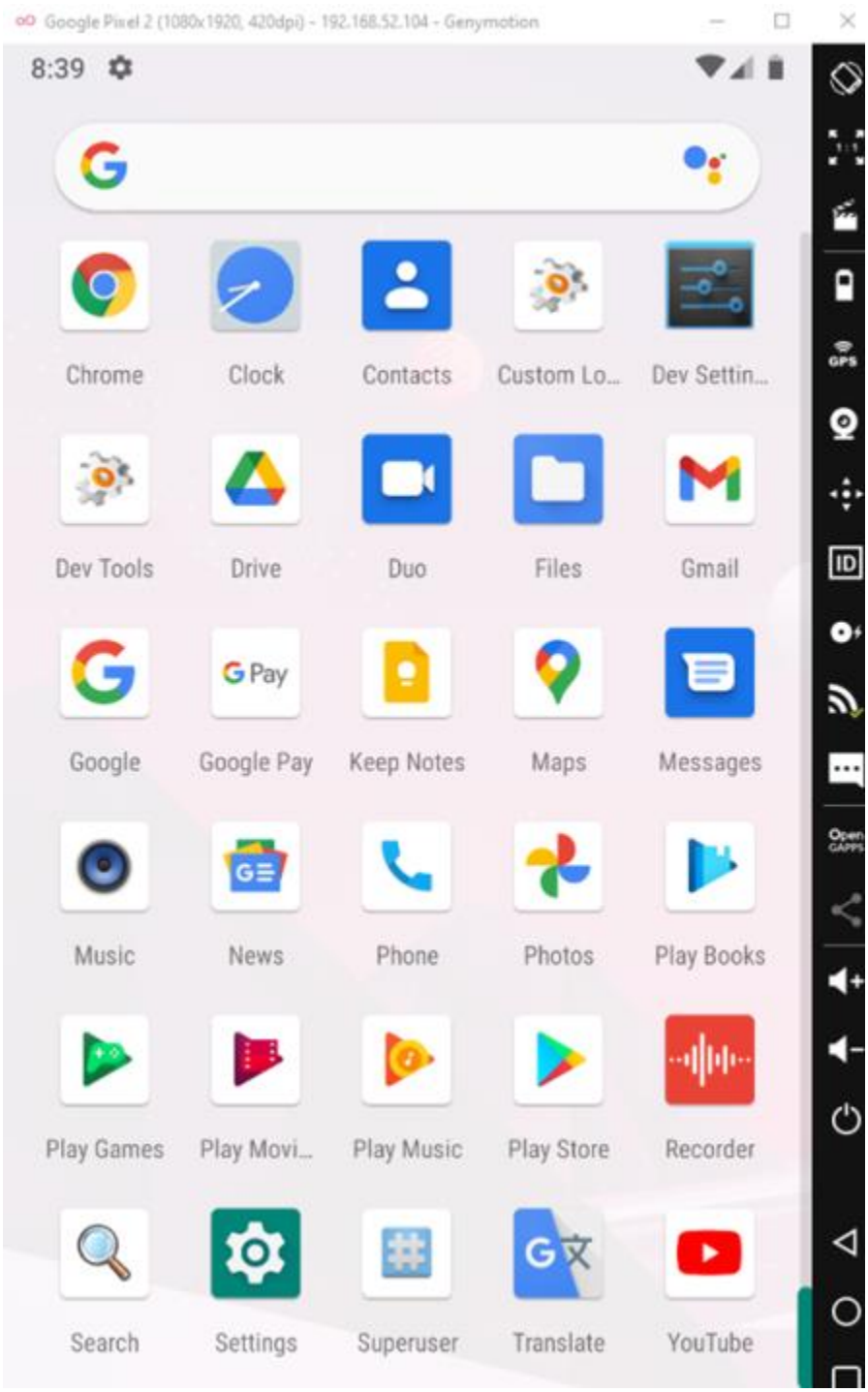


IMPORTANT: In many scenarios, this type of installation may fail giving an "**archive error**". In such cases, it is recommended to download Open GApps archive from their website here.

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.


Google Pixel 2 (1080x1920, 420dpi) - 192.168.52.104 - Genymotion
8:39
Chrome
Clock
Contacts
Custom Lo...
Dev Settin...
Dev Tools
Drive
Duo
Files
Gmail
Google
Google Pay
Keep Notes
Maps
Messages
Music
News
Phone
Photos
Play Books
Play Games
Play Movi...
Play Music
Play Store
Recorder
Search
Settings
Superuser
Translate
YouTube

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

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.

```
┌──(root💀kali)-[/home/kali]
└─# adb connect 192.168.52.104
connected to 192.168.52.104:5555
┌──(root💀kali)-[/home/kali]
└─# adb shell
vbox86p:/ # getprop | grep abi
[ro.product.cpu.abi]: [x86]
[ro.product.cpu.abilist]: [x86]
[ro.product.cpu.abilist32]: [x86]
[ro.product.cpu.abilist64]: []
[ro.vendor.product.cpu.abilist]: [x86]
[ro.vendor.product.cpu.abilist32]: [x86]
[ro.vendor.product.cpu.abilist64]: []
vbox86p:/ # uname -a
Linux localhost 4.4.157-genymotion-gcb750d1 #1 SMP PREEMPT Wed Jan 29 14:54:22 UTC 20
20 i686
vbox86p:/ #
```

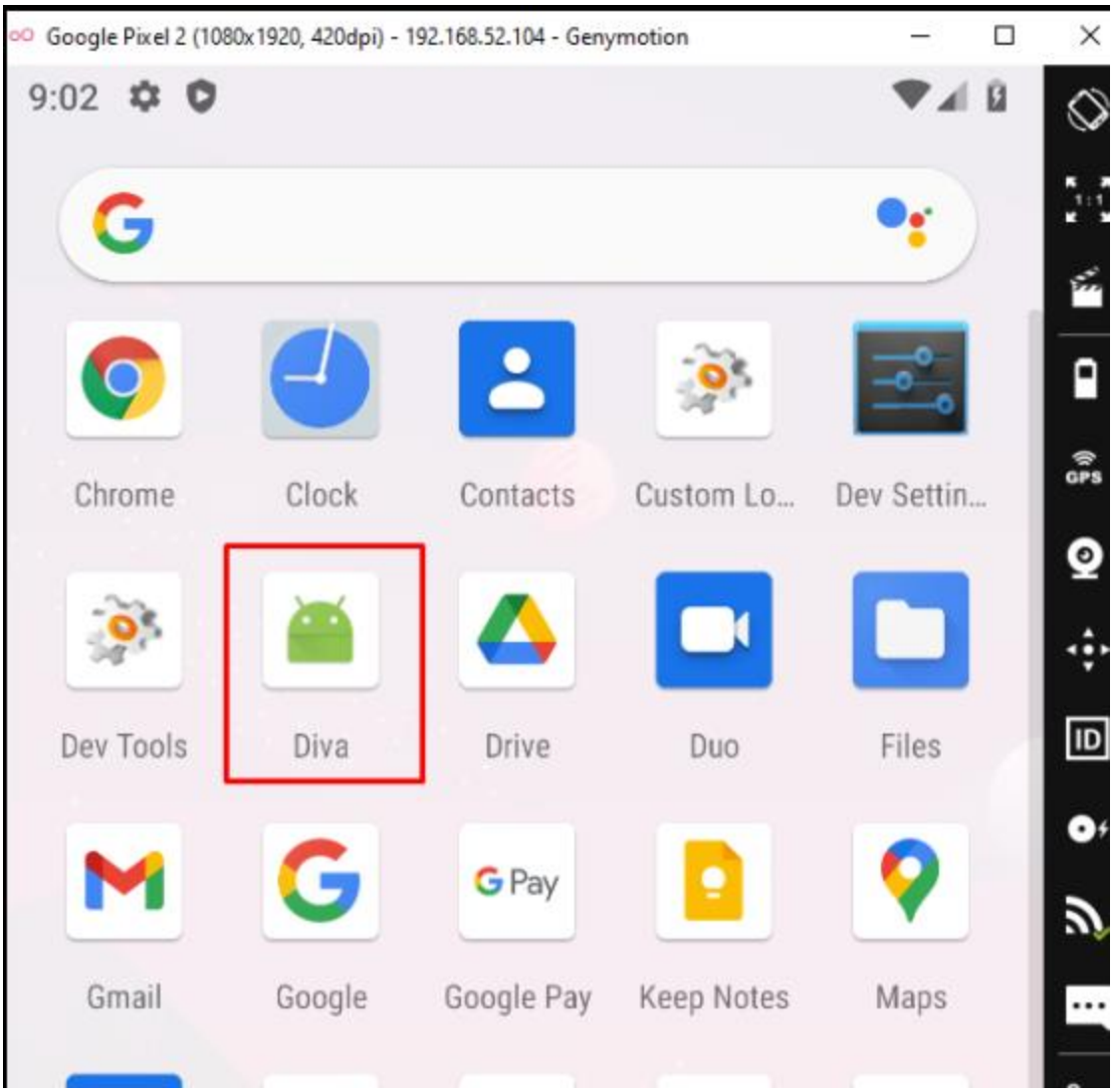
2. **Installation of APK in device**

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
```

```
┌──(root💀kali)-[/home/kali/Downloads]
└─# adb shell
vbox86p:/ # cd data/data/
vbox86p:/data/data # ls | tail -10
com.google.android.tts
com.google.android.videos
com.google.android.webview
com.google.android.youtube
com.opengapps.defaultdialeroverlay
com.opengapps.phoneoverlay
com.opengapps.pixellauncheroverlay
com.opengapps.telecomoverlay
com.opengapps.wellbeingoverlay
jakhar.aseem.diva
vbox86p:/data/data #
```

4. **Starting and Stopping adb service**

This is simply done by the following two commands:

adb start-server
adb kill-server

```
┌──(root💀kali)-[/home/kali]
└─# adb start-server
┌──(root💀kali)-[/home/kali]
└─# adb connect 192.168.52.104
connected to 192.168.52.104:5555
┌──(root💀kali)-[/home/kali]
└─# adb kill-server
┌──(root💀kali)-[/home/kali]
└─# adb devices -l
List of devices attached
* daemon not running; starting now at tcp:5037
* daemon started successfully

┌──(root💀kali)-[/home/kali]
└─#
```

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

```
┌──(root💀kali)-[/home/kali]
└─# adb connect 192.168.52.104
connected to 192.168.52.104:5555
┌──(root💀kali)-[/home/kali]
└─# adb unroot
restarting adbd as non root
```

Logs

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

adb logcat

```
┌──(root💀kali)-[/home/kali/Downloads]
└─# 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.

```
ability permissive=1
11-26 11:03:42.380   572   572 I wifi@1.0-servic: type=1400 audit(0.0:1405): avc: denied { write } for sco
tcontext=u:r:hal_wifi_default:s0 tclass=netlink_route_socket permissive=1
11-26 11:03:42.380   572   572 I wifi@1.0-servic: type=1400 audit(0.0:1406): avc: denied { nlmsg_read } fo
t:s0 tcontext=u:r:hal_wifi_default:s0 tclass=netlink_route_socket permissive=1
11-26 11:03:45.990  7765  7819 I Finsky  : [643] noh.run(3): Stats for Executor: BlockingExecutor nqn@4581
tive threads = 0, queued tasks = 0, completed tasks = 12]
11-26 11:03:45.990  7765  7819 I Finsky  : [643] noh.run(3): Stats for Executor: LightweightExecutor nqn@e
 active threads = 0, queued tasks = 0, completed tasks = 73]
11-26 11:03:46.427  8235  8235 E diva-log: Error while processing transaction with credit card: 0000
11-26 11:03:46.497  7765  7819 I Finsky  : [643] noh.run(3): Stats for Executor: bgExecutor nqn@604c846[Ru
hreads = 0, queued tasks = 0, completed tasks = 63]
11-26 11:03:48.032   193   193 I redis   : type=1400 audit(0.0:1408): avc: denied { accept } for lport=637
xt=u:r:redis:s0 tclass=tcp_socket permissive=1
11-26 11:03:48.032   193   193 I redis   : type=1400 audit(0.0:1409): avc: denied { write } for name="fwma
ntext=u:r:redis:s0 tcontext=u:object_r:fwmarkd_socket:s0 tclass=sock_file permissive=1
11-26 11:03:48.032   193   193 I redis   : type=1400 audit(0.0:1410): avc: denied { connectto } for path="
u:r:redis:s0 tcontext=u:r:netd:s0 tclass=unix_stream_socket permissive=1
11-26 11:03:48.036   193   193 I redis   : type=1400 audit(0.0:1416): avc: denied { read } for path="socke
604 scontext=u:r:redis:s0 tcontext=u:r:redis:s0 tclass=tcp_socket permissive=1
11-26 11:03:48.454   494  2805 W NotificationService: Toast already killed. pkg=jakhar.aseem.diva callback
ation$Stub$Proxy@9d58185
```

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/
```

```
┌──(root💀kali)-[/home/kali/Downloads]
└─# echo "Say my name" > file.txt
┌──(root💀kali)-[/home/kali/Downloads]
└─# adb push file.txt /sdcard/
file.txt: 1 file pushed. 0.0 MB/s (12 bytes in 0.014s)
┌──(root💀kali)-[/home/kali/Downloads]
└─#
```

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

```
┌──(root💀kali)-[/home/kali/Downloads]
└─# rm file.txt
┌──(root💀kali)-[/home/kali/Downloads]
└─# adb pull /sdcard/file.txt /home/kali/Downloads/new_file.txt
/sdcard/file.txt: 1 file pulled. 0.0 MB/s (12 bytes in 0.005s)
┌──(root💀kali)-[/home/kali/Downloads]
└─# cat new_file.txt
Say my name
┌──(root💀kali)-[/home/kali/Downloads]
└─#
```

pm tool

1. **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

```
┌──(root💀kali)-[/home/kali/Downloads]
└─# adb shell pm list packages | tail -10
package:com.opengapps.phoneoverlay
package:com.google.android.apps.magazines
package:com.android.bluetooth
package:com.android.development
package:com.android.providers.contacts
package:com.android.captiveportallogin
package:com.google.android.inputmethod.latin
package:jakhar.aseem.diva
package:com.google.android.storagemanager
package:com.google.android.apps.restore
┌──(root💀kali)-[/home/kali/Downloads]
└─#
```

2. **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

```
┌──(root💀kali)-[/home/kali/Downloads]
└─# adb shell pm list packages -s
package:com.google.android.carriersetup
package:com.android.cts.priv.ctsshim
package:com.google.android.youtube
package:com.android.internal.display.cutout.emulation.corner
package:com.google.android.ext.services
package:com.example.android.livecubes
package:com.android.internal.display.cutout.emulation.double
package:com.android.providers.telephony
package:com.google.android.googlequicksearchbox
package:com.android.providers.calendar
package:com.android.providers.media
package:com.google.android.onetimeinitializer
package:com.google.android.ext.shared
package:com.android.wallpapercropper
package:com.android.documentsui
package:com.android.externalstorage
package:com.android.htmlviewer
package:com.android.companiondevicemanager
package:com.android.quicksearchbox
package:com.android.mms.service
package:com.android.providers.downloads
```

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

adb shell pm list packages –3

```
┌──(root💀kali)-[/home/kali/Downloads]
└─# adb shell pm list packages -3
package:jakhar.aseem.diva
┌──(root💀kali)-[/home/kali/Downloads]
└─#
```

4. **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

```
┌──(root💀kali)-[/home/kali/Downloads]
└─# adb shell pm clear jakhar.aseem.diva
Success
┌──(root💀kali)-[/home/kali/Downloads]
└─#
```

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

```
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.ap
ice.DrawerOverlayService}
    intent={act=com.android.launcher3.WINDOW_OVERLAY dat=app://com.google.android.apps
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/u0a8
    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.ap
7?v=9&cv=14 pkg=com.google.android.googlequicksearchbox}
      binder=null
      requested=true received=false hasBound=true doRebind=false
      * Client AppBindRecord{f7cf55c ProcessRecord{86a5c24 1387:com.google.android.app
}}
        Per-process Connections:
          ConnectionRecord{a56c04c u0 CR IMP com.google.android.googlequicksearchbox/c
s.gsa.nowoverlayservice.DrawerOverlayService:@223487f}
          ConnectionRecord{db6d59e u0 CR WPRI com.google.android.googlequicksearchbox/
ps.gsa.nowoverlayservice.DrawerOverlayService:@edff7d9}
    All Connections:
      ConnectionRecord{a56c04c u0 CR IMP com.google.android.googlequicksearchbox/com.g
a.nowoverlayservice.DrawerOverlayService:@223487f}
      ConnectionRecord{db6d59e u0 CR WPRI com.google.android.googlequicksearchbox/com.
sa.nowoverlayservice.DrawerOverlayService:@edff7d9}

  (nothing)
```

dumpsys tool

1. **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 pm path 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.

```
┌──(root💀kali)-[/home/kali/Downloads]
└─# adb shell pm path jakhar.aseem.diva
package:/data/app/jakhar.aseem.diva-GFSriuE4GUNh7WcrGufkhQ==/base.apk
┌──(root💀kali)-[/home/kali/Downloads]
└─# adb shell pm clear jakhar.aseem.diva
Success
┌──(root💀kali)-[/home/kali/Downloads]
└─#
```

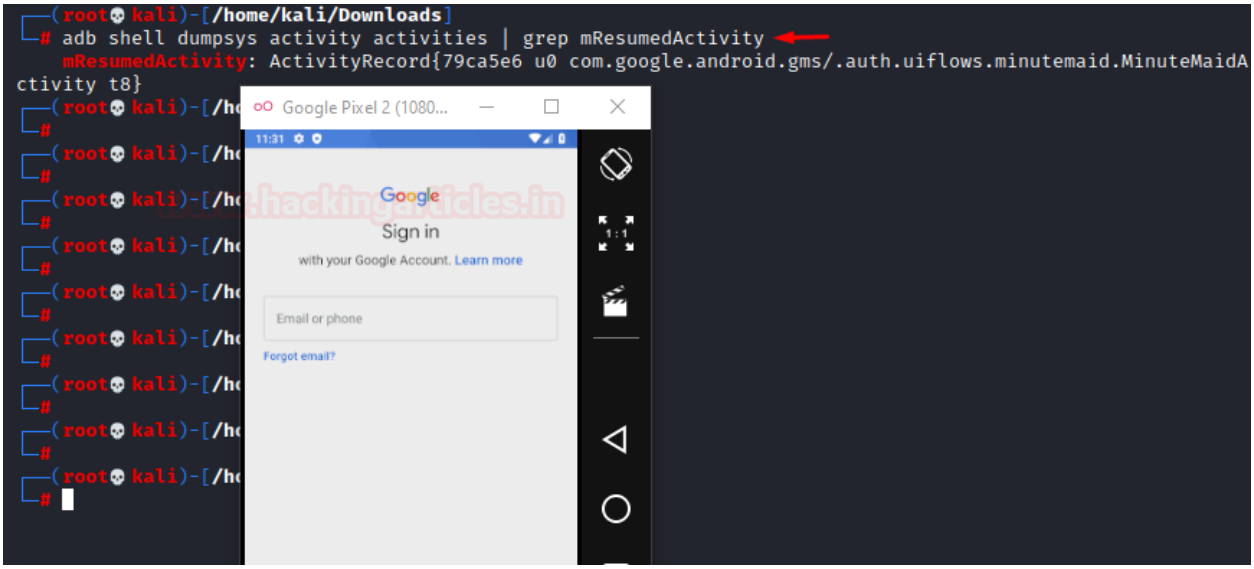
2. **Extracting information about a package**: A package will have components like actions, activities, content providers etc as mentioned in the previous article and to view this information about a specific package we use the following command:

adb shell dumpsys package jakhar.aseem.diva

```
┌──(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}

Key Set Manager:
  [jakhar.aseem.diva]
      Signing KeySets: 30

Packages:
  Package [jakhar.aseem.diva] (4a91d69):
    userId=10126
    pkg=Package{b2db3ee jakhar.aseem.diva}
    codePath=/data/app/jakhar.aseem.diva-GFSriuE4GUNh7WcrGufkhQ==
    resourcePath=/data/app/jakhar.aseem.diva-GFSriuE4GUNh7WcrGufkhQ==
    legacyNativeLibraryDir=/data/app/jakhar.aseem.diva-GFSriuE4GUNh7WcrGufkhQ==/lib
    primaryCpuAbi=x86
```

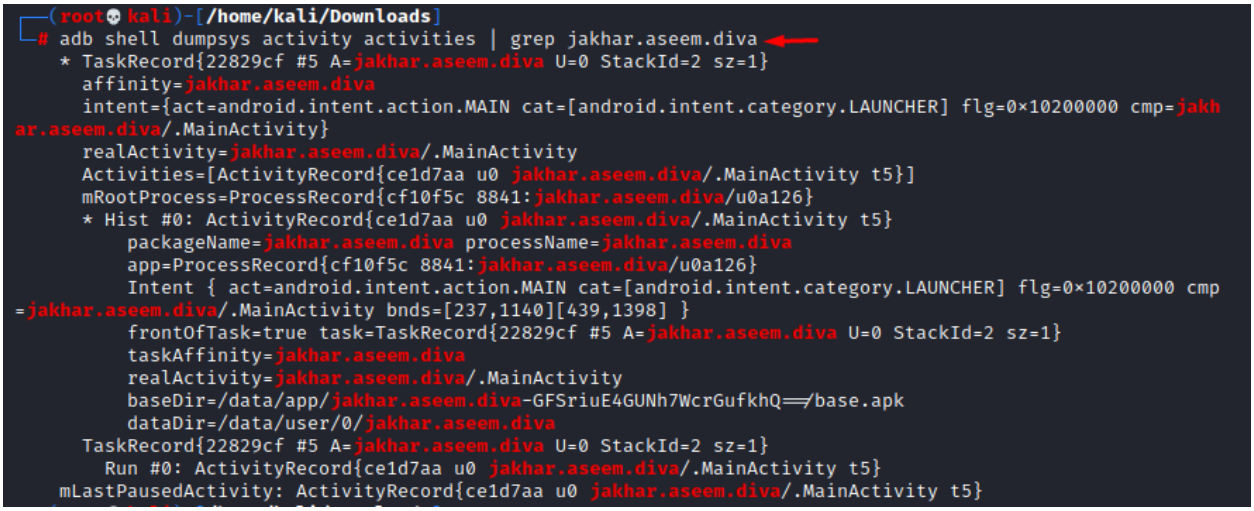
3. **View foreground activity:** The activity that is currently on the main screen can be displayed with the following command:

adb shell dumpsys activity activities | grep mResumedActivity



4. **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



5. **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
      requested=true received=false hasBound=true doRebind=false
      * Client AppBindRecord{f7cf55c ProcessRecord{86a5c24 1387:com.google.android.apps.n
exuslauncher/u0a87}}
        Per-process Connections:
          ConnectionRecord{a56c04c u0 CR IMP com.google.android.googlequicksearchbox/com.
google.android.apps.gsa.nowoverlayservice.DrawerOverlayService:@223487f}
          ConnectionRecord{db6d59e u0 CR WPRI com.google.android.googlequicksearchbox/com
.google.android.apps.gsa.nowoverlayservice.DrawerOverlayService:@edff7d9}
    All Connections:
      ConnectionRecord{a56c04c u0 CR IMP com.google.android.googlequicksearchbox/com.goog
le.android.apps.gsa.nowoverlayservice.DrawerOverlayService:@223487f}
      ConnectionRecord{db6d59e u0 CR WPRI com.google.android.googlequicksearchbox/com.goo
gle.android.apps.gsa.nowoverlayservice.DrawerOverlayService:@edff7d9}

  (nothing)
```

6. **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}

Key Set Manager:
  [jakhar.aseem.diva]
      Signing KeySets: 30

Packages:
  Package [jakhar.aseem.diva] (4a91d69):
    userId=10126
    pkg=Package{b2db3ee jakhar.aseem.diva}
    codePath=/data/app/jakhar.aseem.diva-GFSriuE4GUNh7WcrGufkhQ==
    resourcePath=/data/app/jakhar.aseem.diva-GFSriuE4GUNh7WcrGufkhQ==
    legacyNativeLibraryDir=/data/app/jakhar.aseem.diva-GFSriuE4GUNh7WcrGufkhQ==/lib
    primaryCpuAbi=x86
    secondaryCpuAbi=null
    versionCode=1 minSdk=15 targetSdk=23
    versionName=1.0
    splits=[base]
    apkSigningVersion=1
    applicationInfo=ApplicationInfo{10a3f0 jakhar.aseem.diva}
    flags=[ DEBUGGABLE HAS_CODE ALLOW_CLEAR_USER_DATA ALLOW_BACKUP ]
    dataDir=/data/user/0/jakhar.aseem.diva
    supportsScreens=[small, medium, large, xlarge, resizeable, anyDensity]
    usesLibraries:
```

am tool

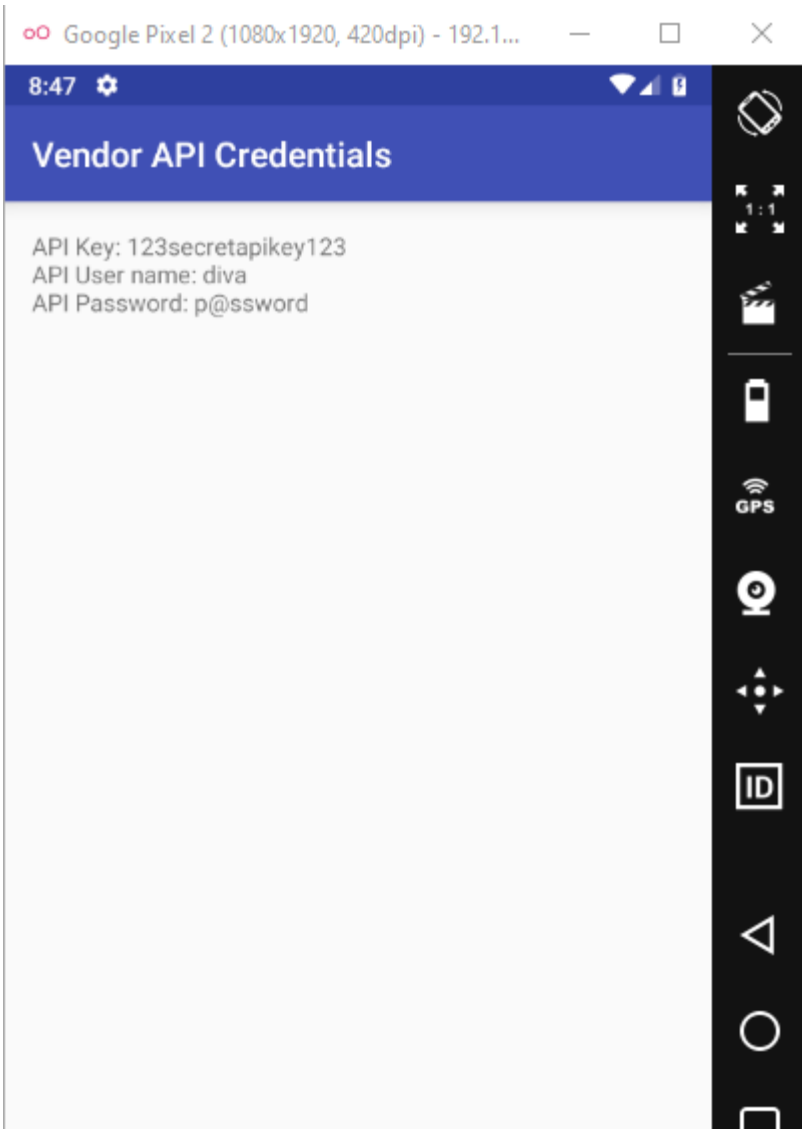
Activities are any page coded in java that performs a specific task. What's interesting is that these activities can also be played around with adb. Many of these functions are performed using another tool called **am** that is installed in android to give information about activities.

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

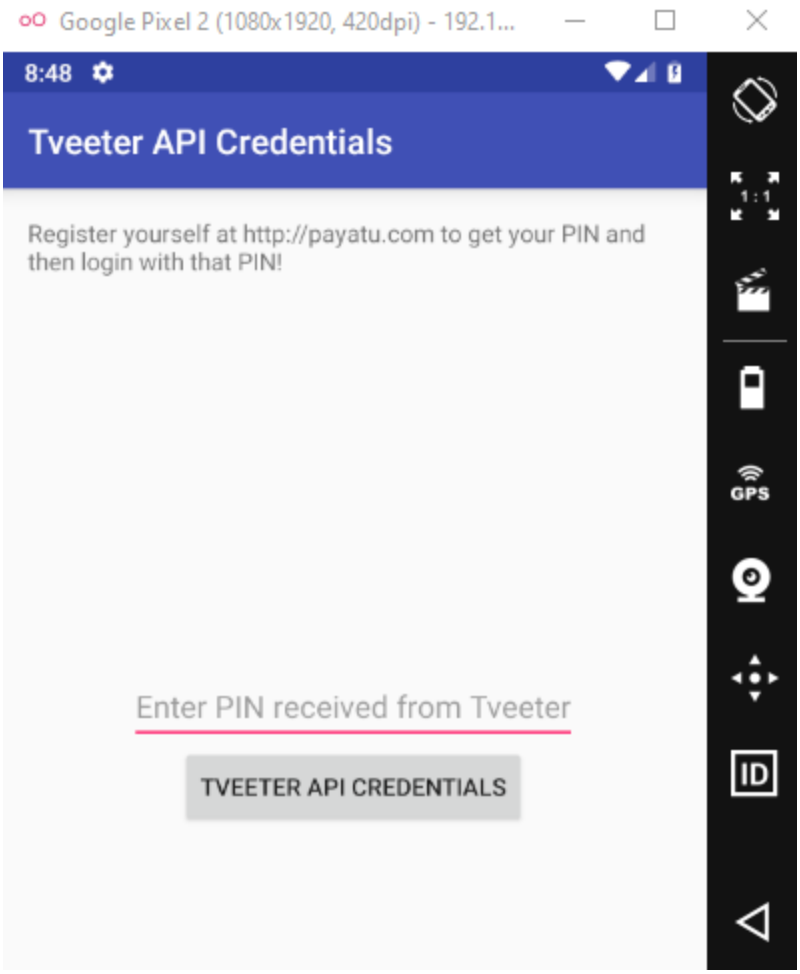
adb shell am start -n jakhar.aseem.diva/.APICredsActivity
adb shell am start -n jakhar.aseem.diva/.APICreds2Activity

```
┌──(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:



2. **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 }
```

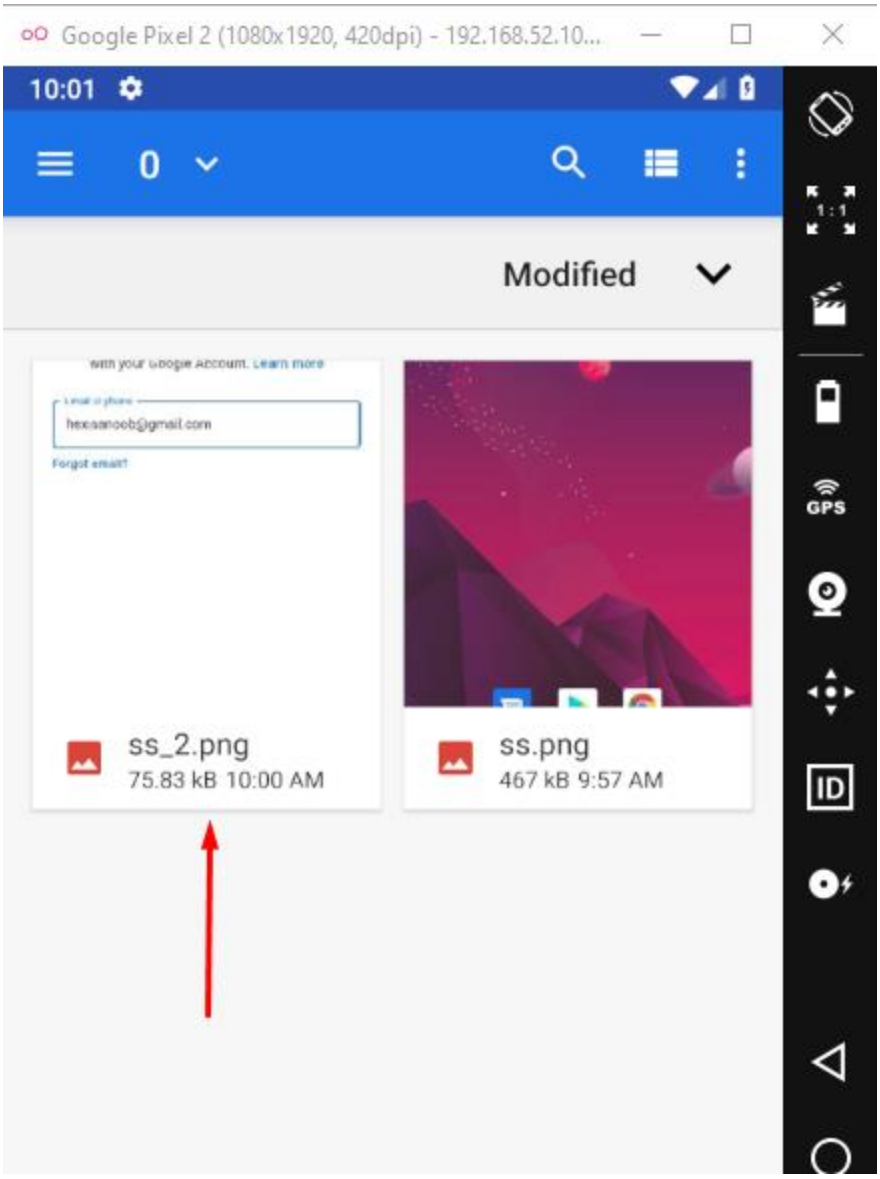
Miscellaneous

1. **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

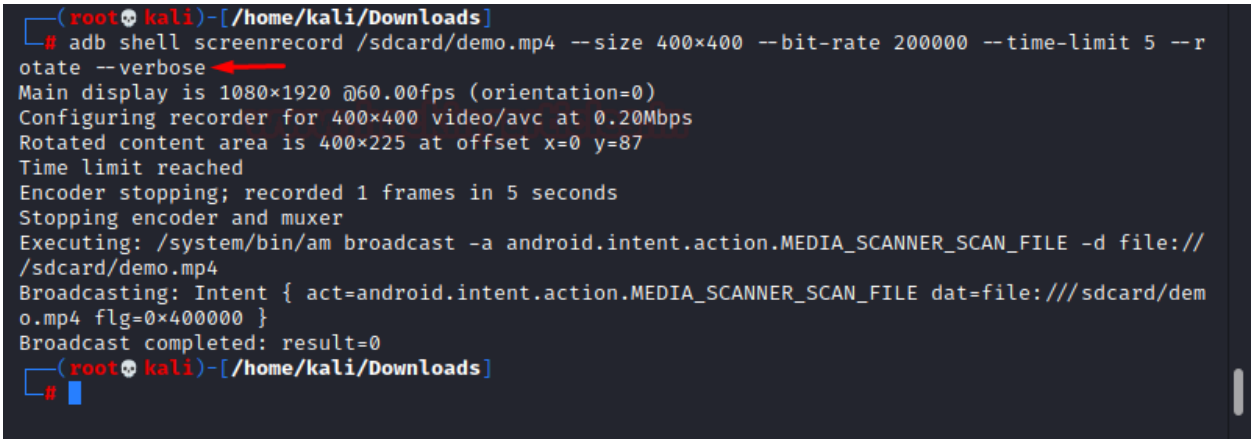
```
┌──(root💀kali)-[/home/kali/Downloads]
└─# adb shell screencap /sdcard/ss_2.png
┌──(root💀kali)-[/home/kali/Downloads]
└─#
```

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:



2. **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 --verbosenc.cmd



/sdcard/demo.mp4 – Path to store the recording

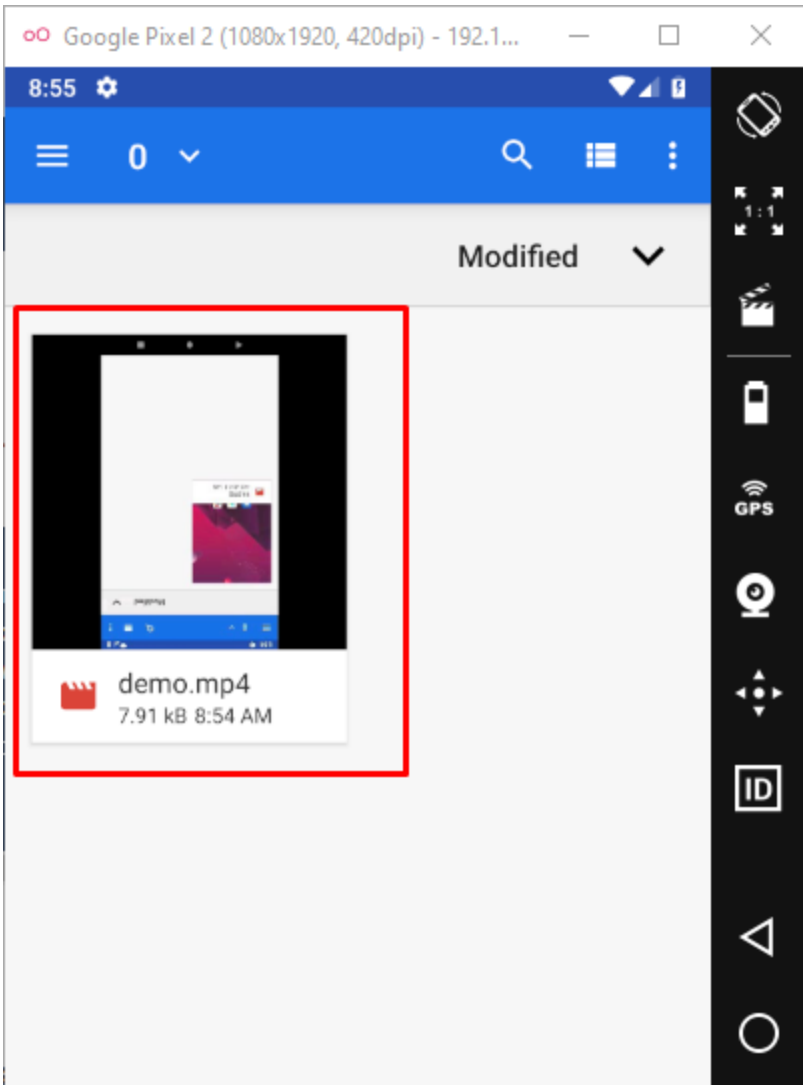
size – The size in pixelxpixel (heightxwidth) in which the video will be recorded

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:



3. **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
```

```
┌──(root💀kali)-[/home/kali/Downloads]
└─# adb shell pidof jakhar.aseem.diva
7399
┌──(root💀kali)-[/home/kali/Downloads]
└─# adb logcat | grep 7399
11-27 08:56:31.470   484   506 I ActivityManager: Start proc 7399:jakhar.aseem.diva/u0a126 for
activity jakhar.aseem.diva/.MainActivity
11-27 08:56:31.487  7399  7399 I Zygote  : seccomp disabled by setenforce 0
11-27 08:56:31.493  7399  7399 I khar.aseem.div: Late-enabling -Xcheck:jni
11-27 08:56:31.722  7399  7399 W khar.aseem.div: Unexpected CPU variant for X86 using defaults:
 x86
11-27 08:56:31.840  7399  7399 I khar.aseem.div: The ClassLoaderContext is a special shared lib
rary.
11-27 08:56:31.965  7399  7416 D libEGL  : Emulator has host GPU support, qemu.gles is set to 1
.
11-27 08:56:32.007  7399  7399 W khar.aseem.div: Accessing hidden method Landroid/view/View;→c
omputeFitSystemWindows(Landroid/graphics/Rect;Landroid/graphics/Rect;)Z (light greylist, reflec
tion)
11-27 08:56:32.008  7399  7399 W khar.aseem.div: Accessing hidden method Landroid/view/ViewGrou
p;→makeOptionalFitsSystemWindows()V (light greylist, reflection)
11-27 08:56:32.033  7399  7399 I jakhar.aseem.diva: type=1400 audit(0.0:1565): avc: denied { wr
ite } for comm=45474C20496E6974 name="property_service" dev="tmpfs" ino=8239 scontext=u:r:untru
sted_app_25:s0:c512,c768 tcontext=u:object_r:property_socket:s0 tclass=sock_file permissive=1
11-27 08:56:32.037  7399  7399 I jakhar.aseem.diva: type=1400 audit(0.0:1566): avc: denied { co
nnectto } for comm=45474C20496E6974 path="/dev/socket/property_service" scontext=u:r:untrusted
```

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

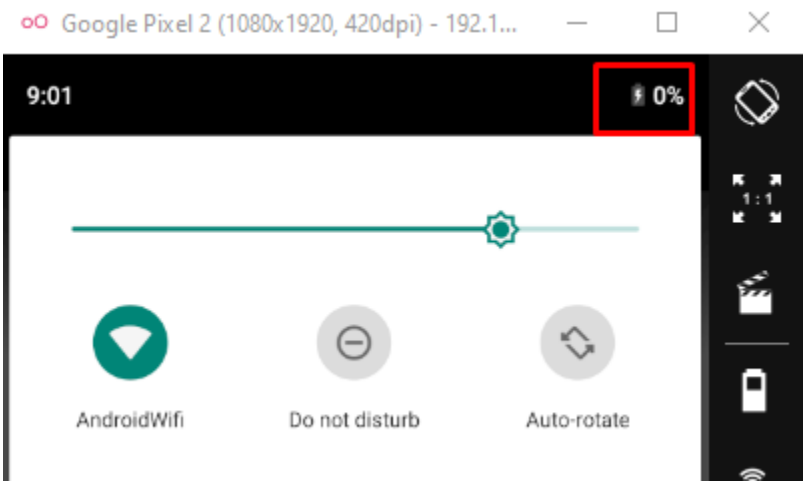
adb shell dumpsys battery

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]
└─#
```

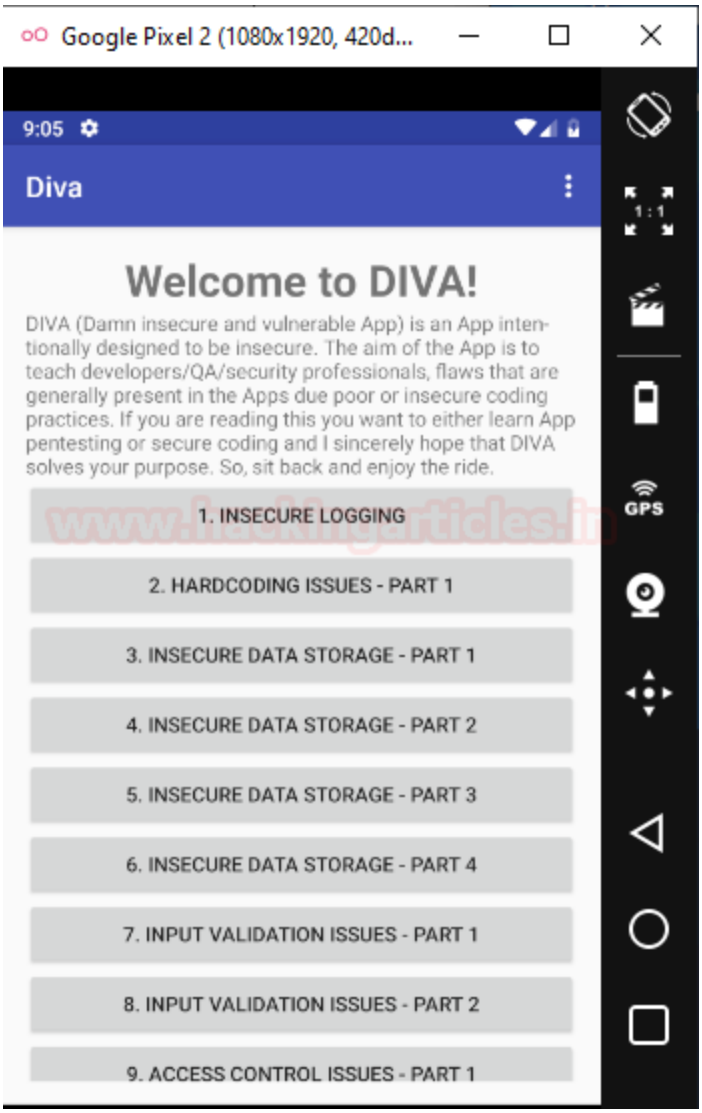
Sure enough, let's see how the battery now looks like:



5. **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:

6. **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]
└─#
```

7. **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
```

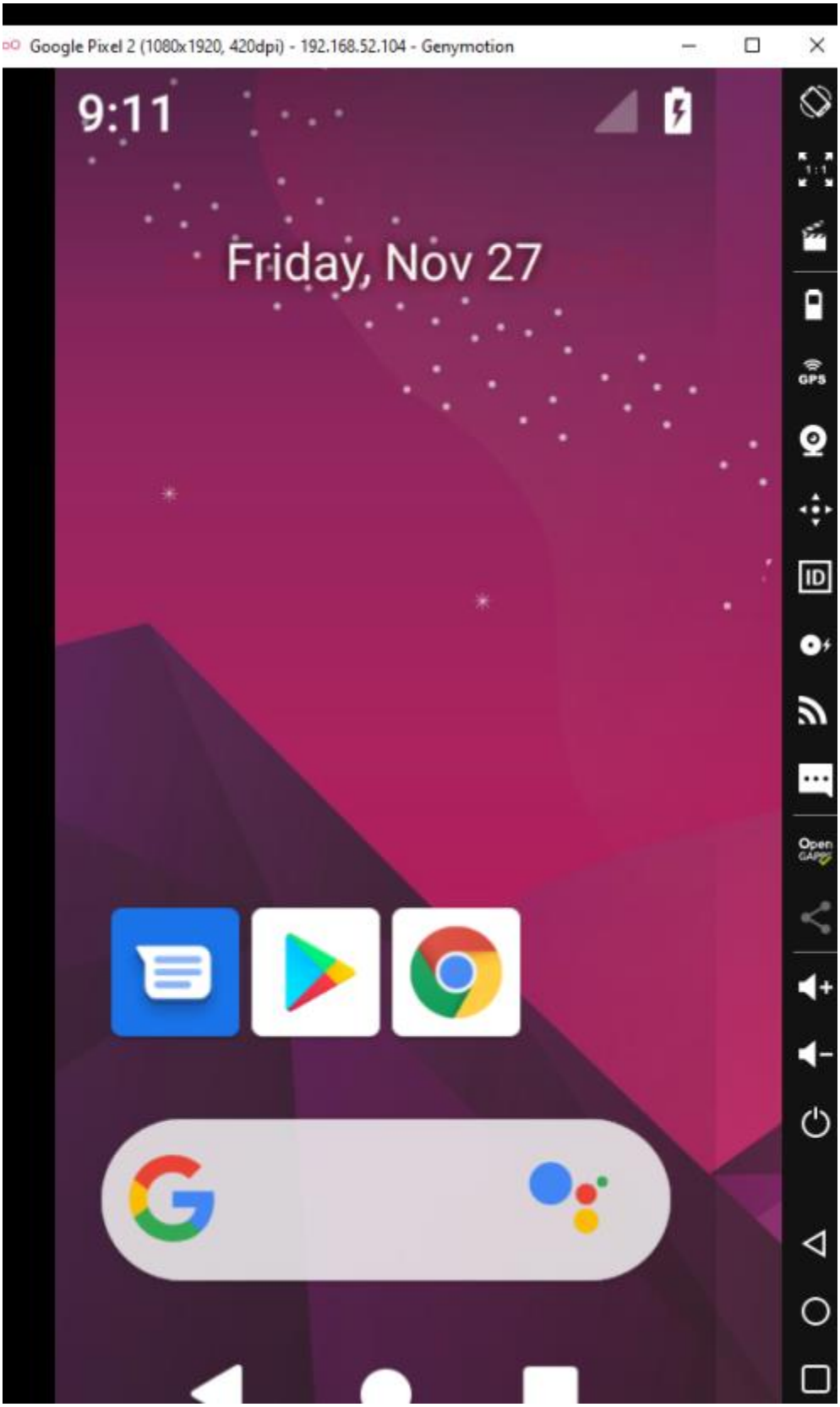
We can modify the screen resolution as well with wm tool using the following command:

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

where the size is in pixels.

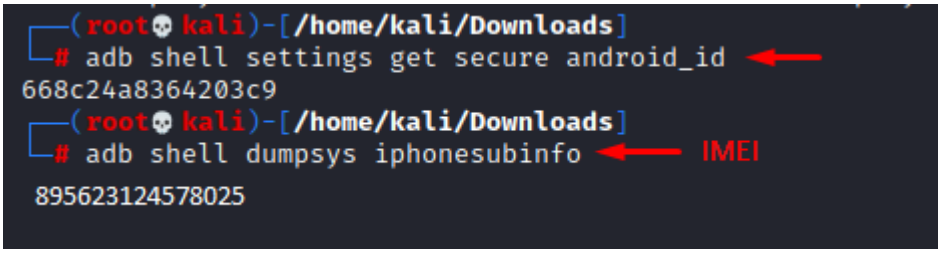
```
┌──(root💀 kali)-[/home/kali/Downloads]
└─# adb shell wm size 480×1024
┌──(root💀 kali)-[/home/kali/Downloads]
└─# adb reboot
┌──(root💀 kali)-[/home/kali/Downloads]
└─#
```

After reboot the window will look something like this:


Google Pixel 2 (1080x1920, 420dpi) - 192.168.52.104 - Genymotion
9:11
Friday, Nov 27
GPS
ID
Open GAPPS

- **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



Conclusion

In this article, we have looked how to set up a testing environment in our PC using Genymotion Android Emulator and performing various functions on it using ADB tool that we installed in Kali. In upcoming articles we'll have a look at various other tools, OWASP methodology for testing, understanding what are android vulnerabilities and of course, automated tool for android security testing. Thanks for reading.

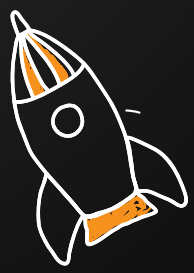
iGNITE Technologies

JOIN OUR TRAINING PROGRAMS

CLICK HERE

BEGINNER

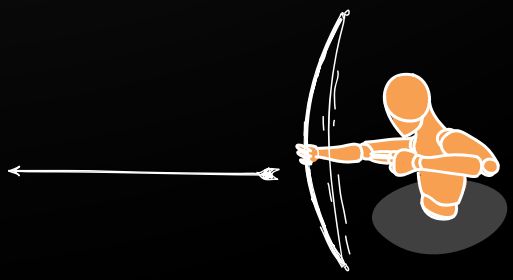
- Ethical Hacking
- Network Pentest
- Bug Bounty
- Wireless Pentest
- Network Security Essentials



ADVANCED

- Burp Suite Pro
- Android Pentest
- Web Services-API
- Advanced Metasploit
- Pro Infrastructure VAPT
- CTF
- Computer Forensics

EXPERT



- Red Team Operation
  - APT's - MITRE Attack Tactics
  - Active Directory Attack
  - MSSQL Security Assessment
- Privilege Escalation
  - Windows
  - Linux