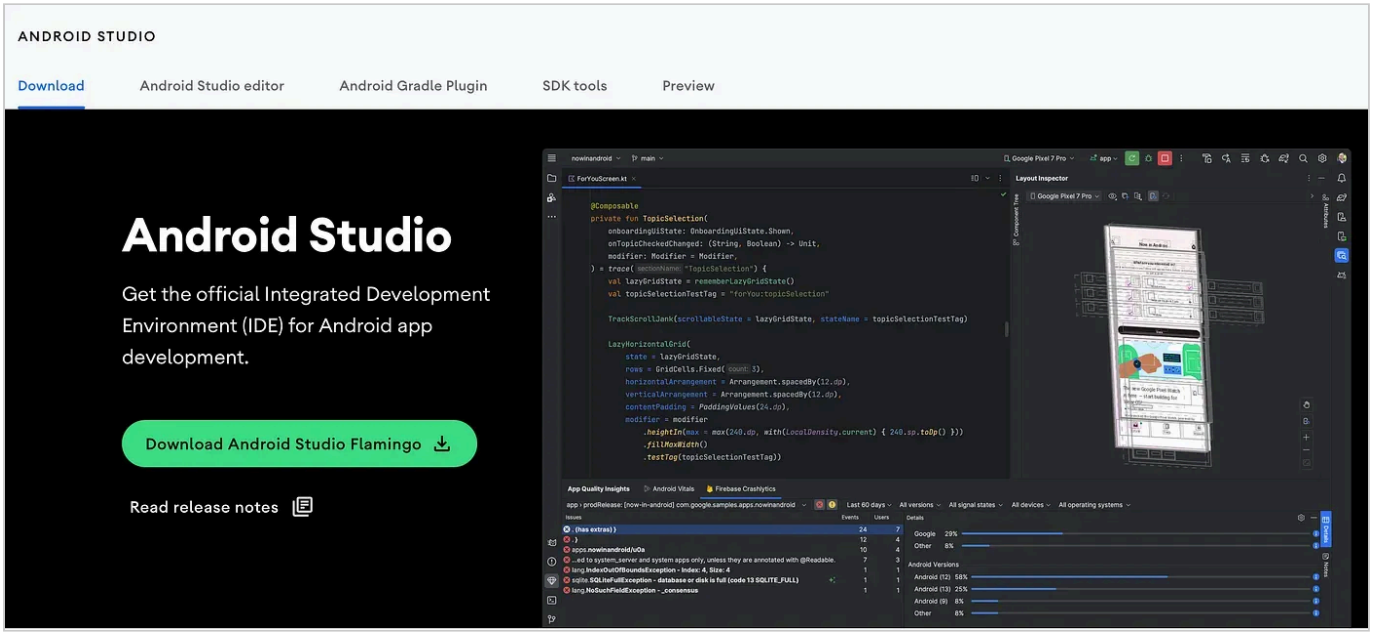
FIELD NOTE / ANDROID

Complete Beginner’s Guide to Setup Android Pentest Lab without Physical Device

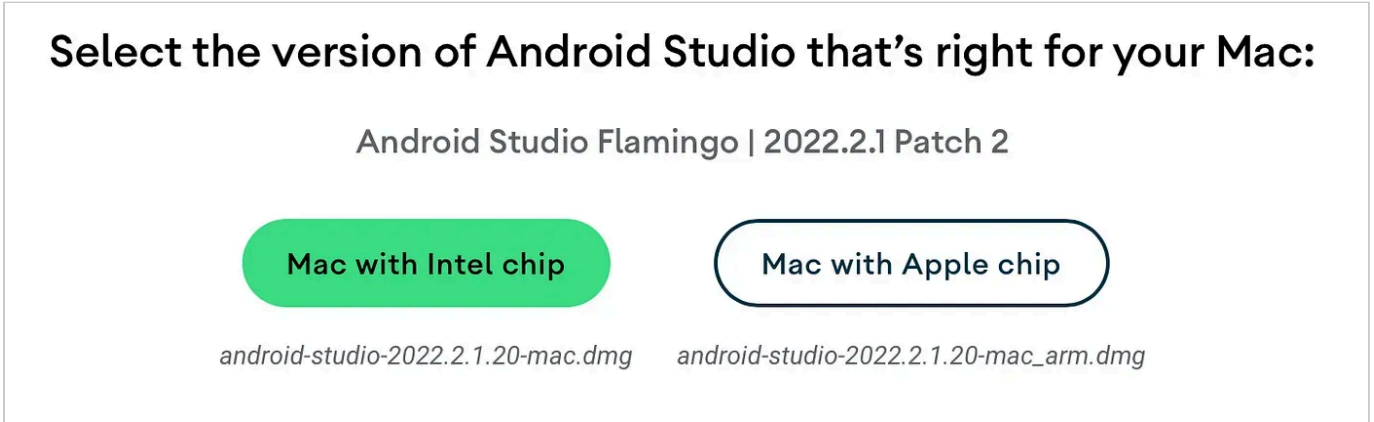
Petruknisme · 7 Mar, 2024 · 5 min to read

Are you interested in exploring the field of Android penetration testing but lacking on physical device? This guide will explain how to set up an Android pentesting lab using an emulator, eliminating the need for a physical device.

The method I’ll use for this guide relies on the Android Virtual Device (AVD) from Android Studio. AVD support Mac computers with ARM processors, making it a perfect choice for this specific approach. Download Android Studio from official website and choose mac apple chip if you are using ARM chip, like I did.



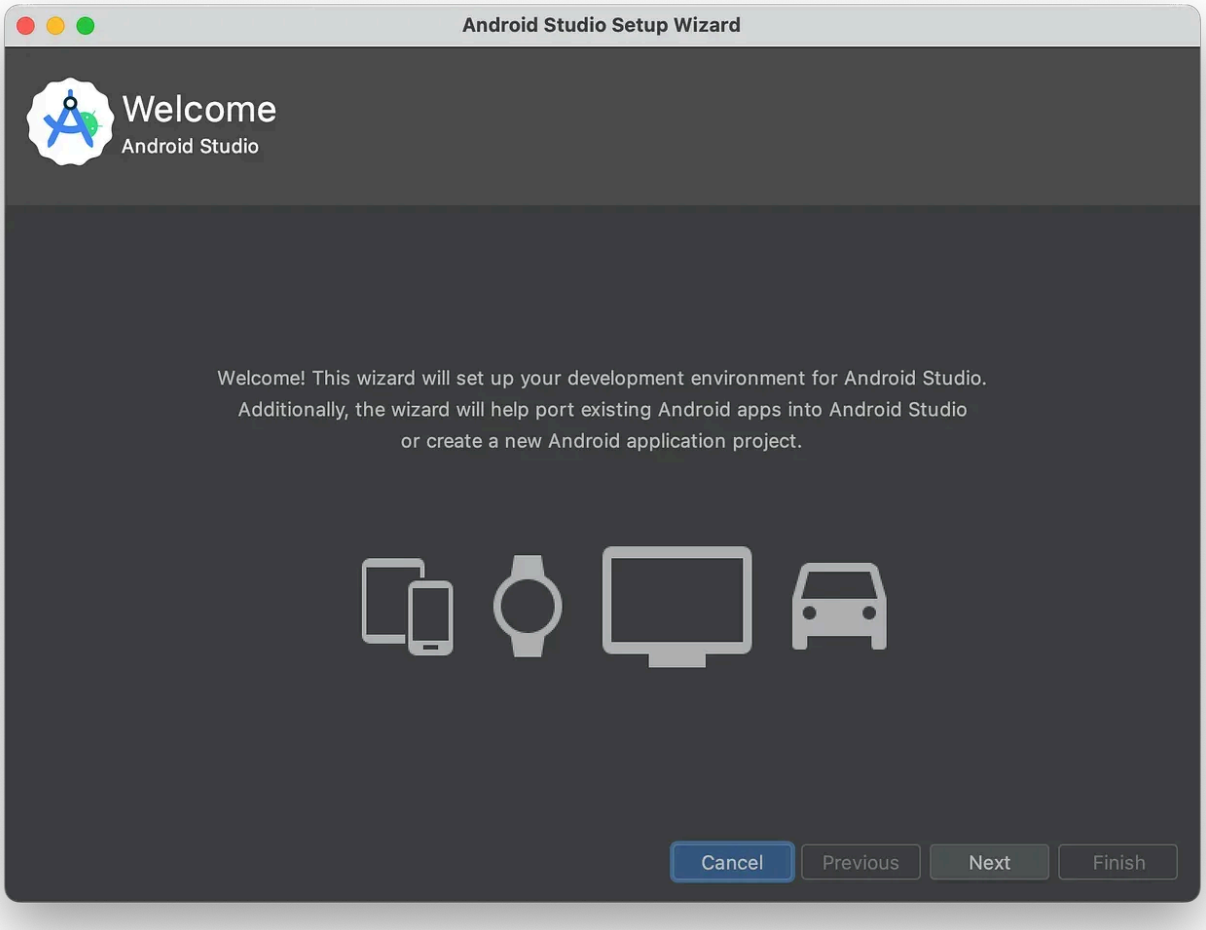
Download Android Studio & App Tools - Android Developers (https://developer.android.com/studio)



Licensed under CC BY-SA 4.0

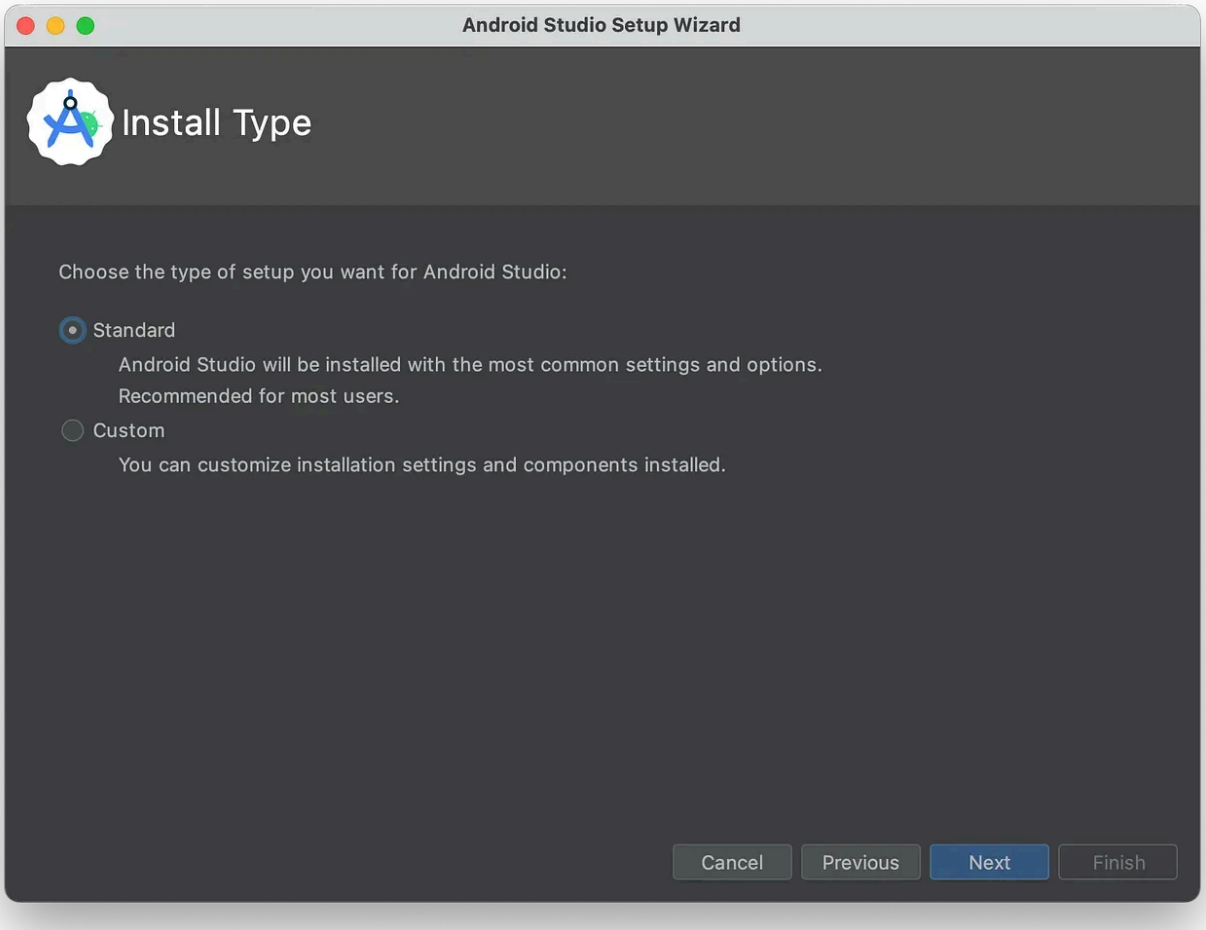
Install and setup Android Studio

Follow the wizard to install android studio

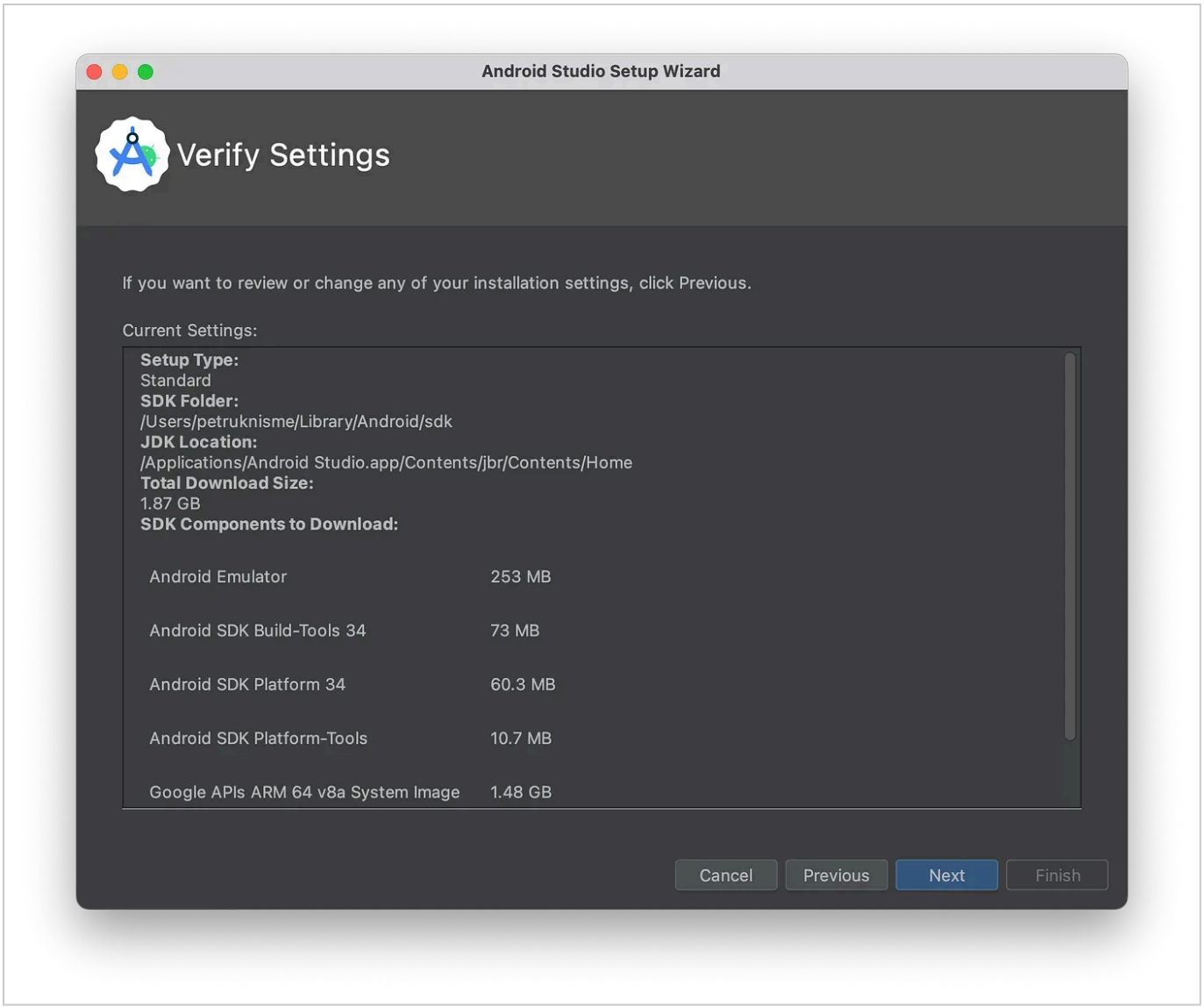


Choose standard install

Licensed under CC BY-SA 4.0

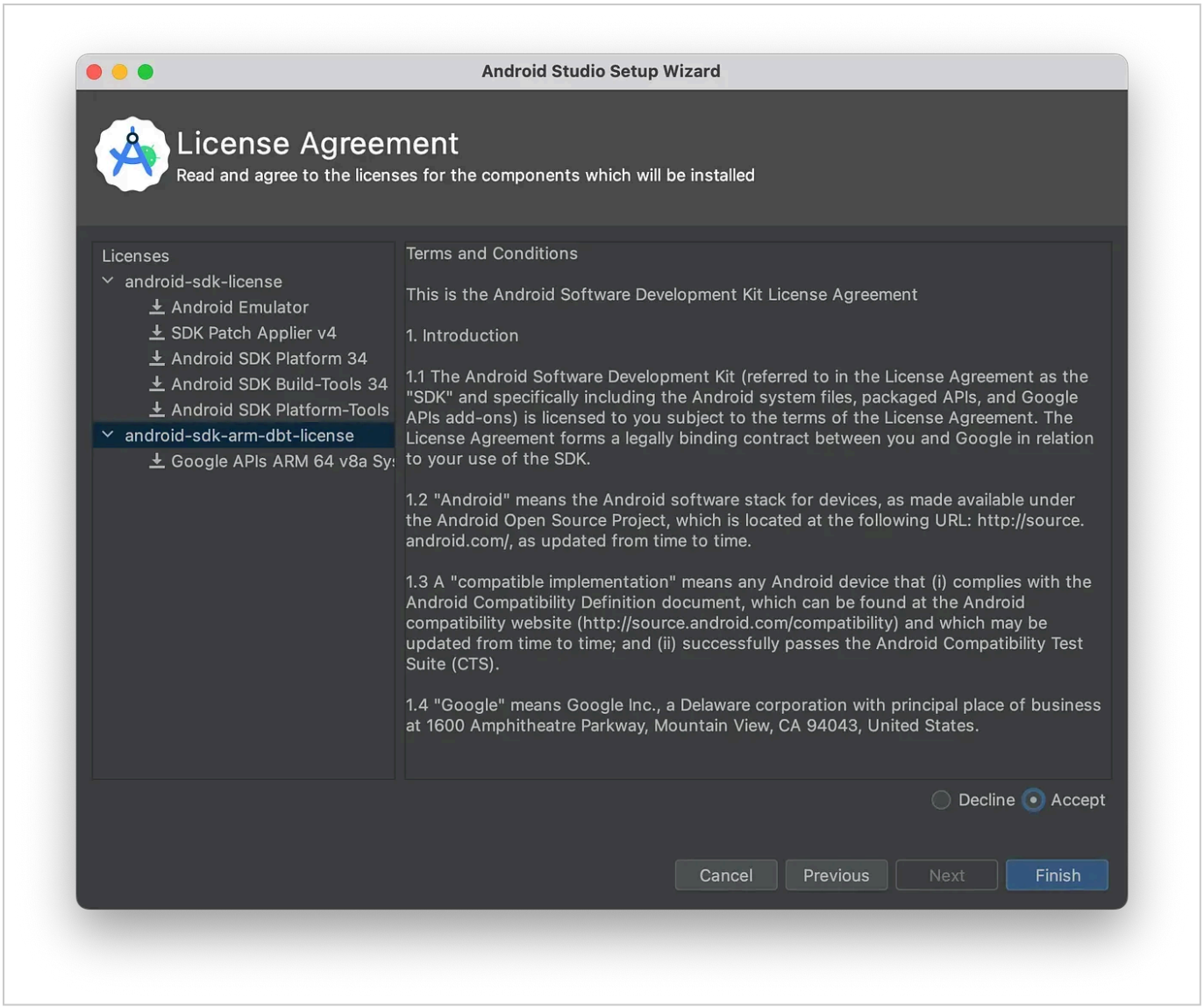


Licensed under CC BY-SA 4.0



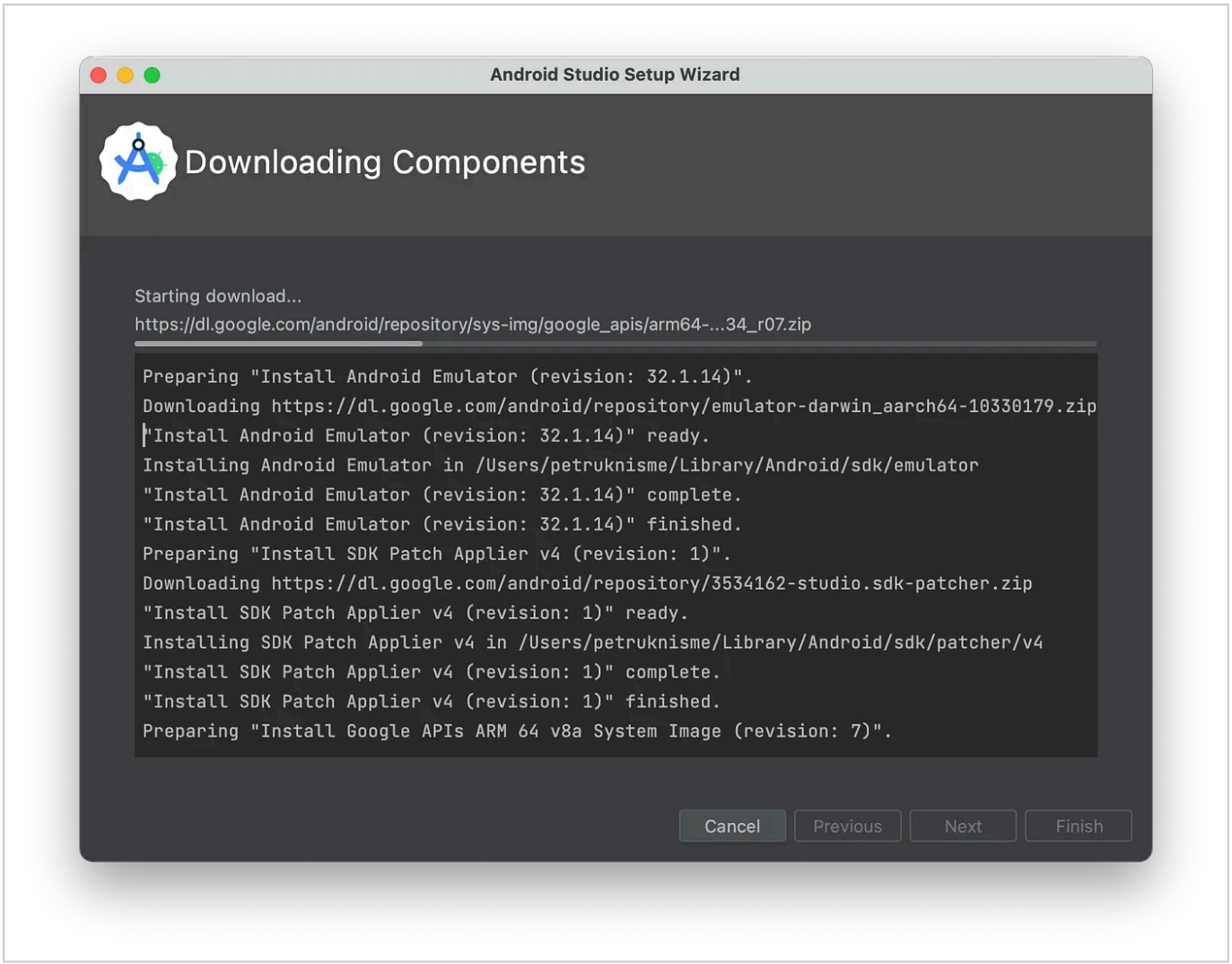
Current setting for android studio wizard installation

Licensed under CC BY-SA 4.0



Accepting User license

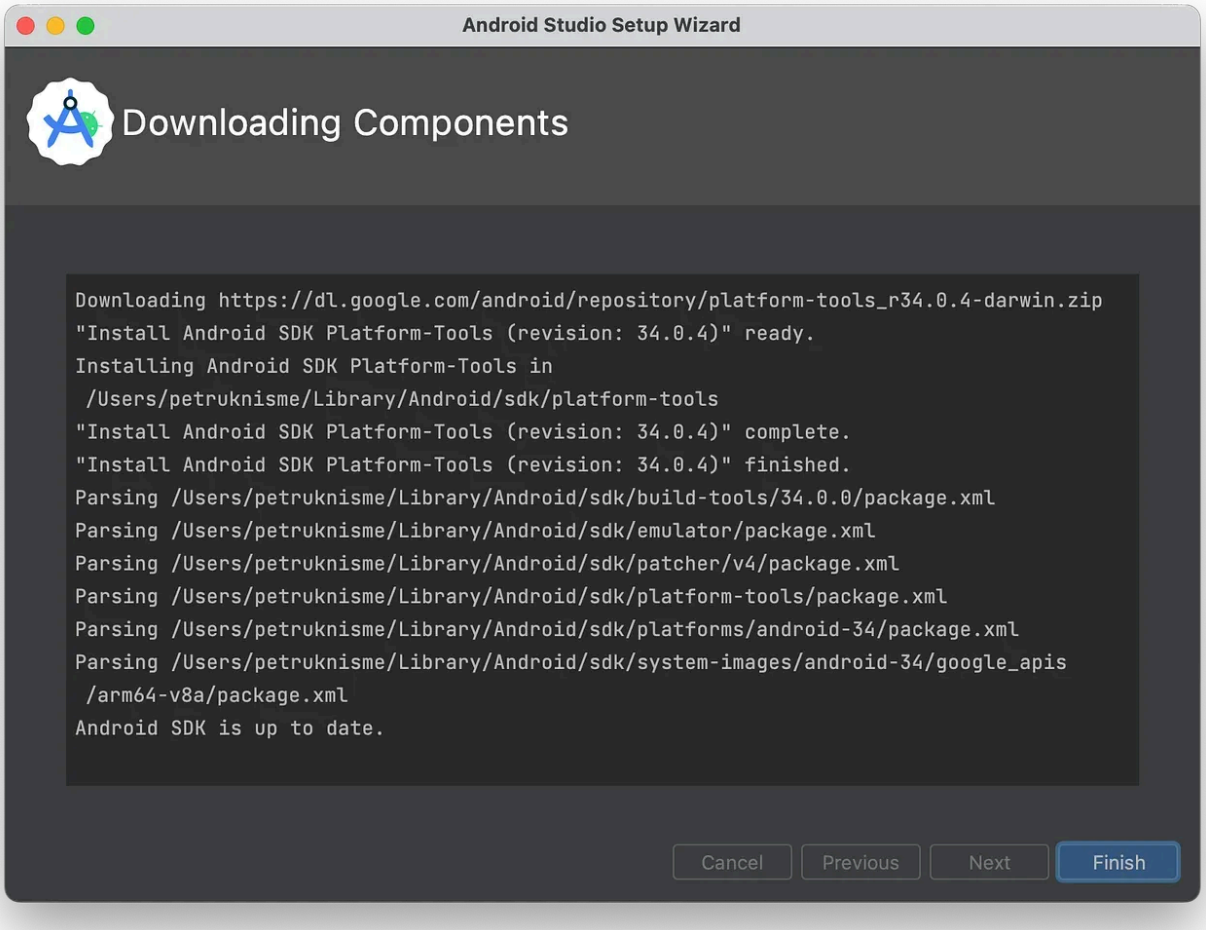
Licensed under CC BY-SA 4.0



Downloading android virtual device package

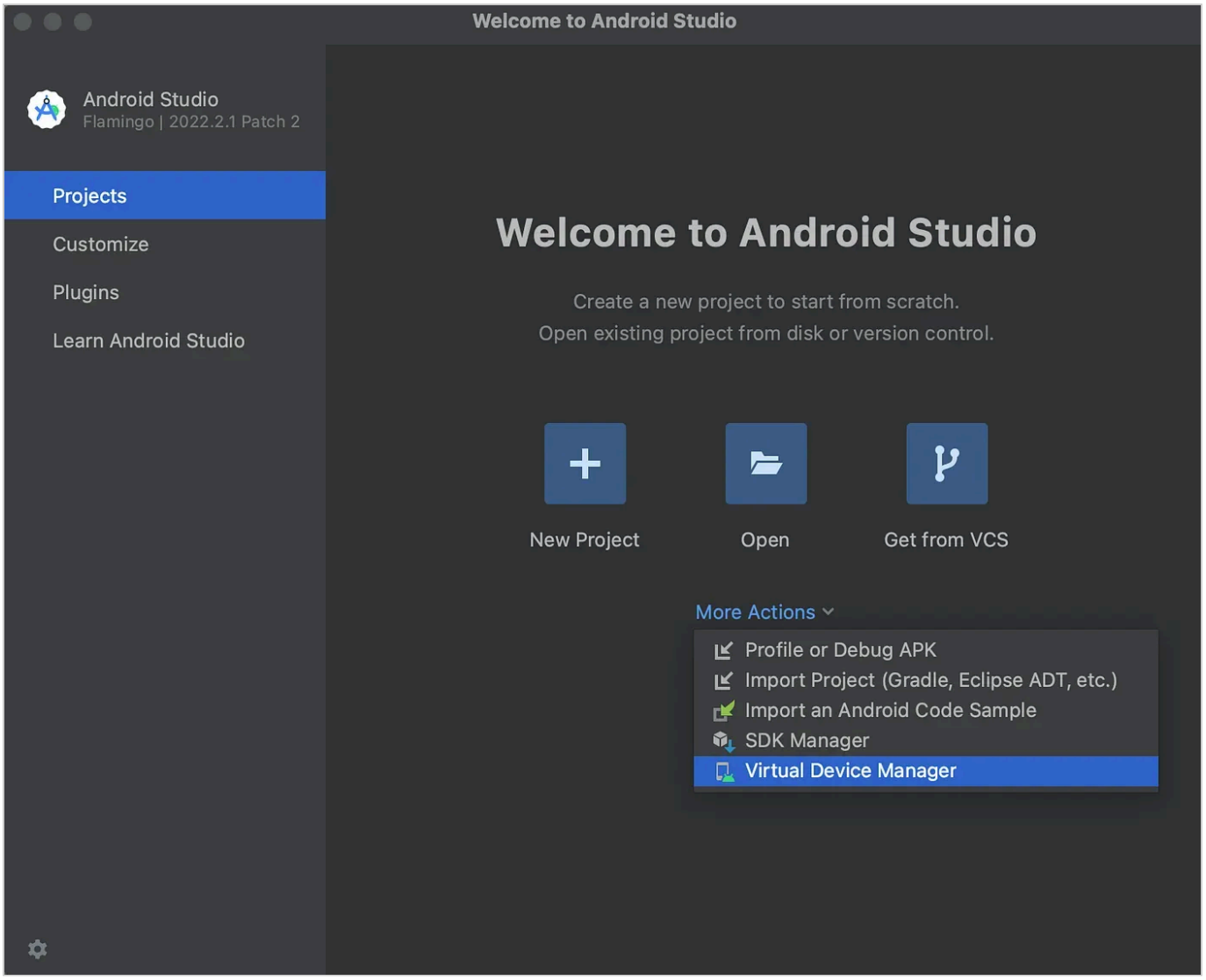
Wait until android studio setup wizard finished

Licensed under CC BY-SA 4.0



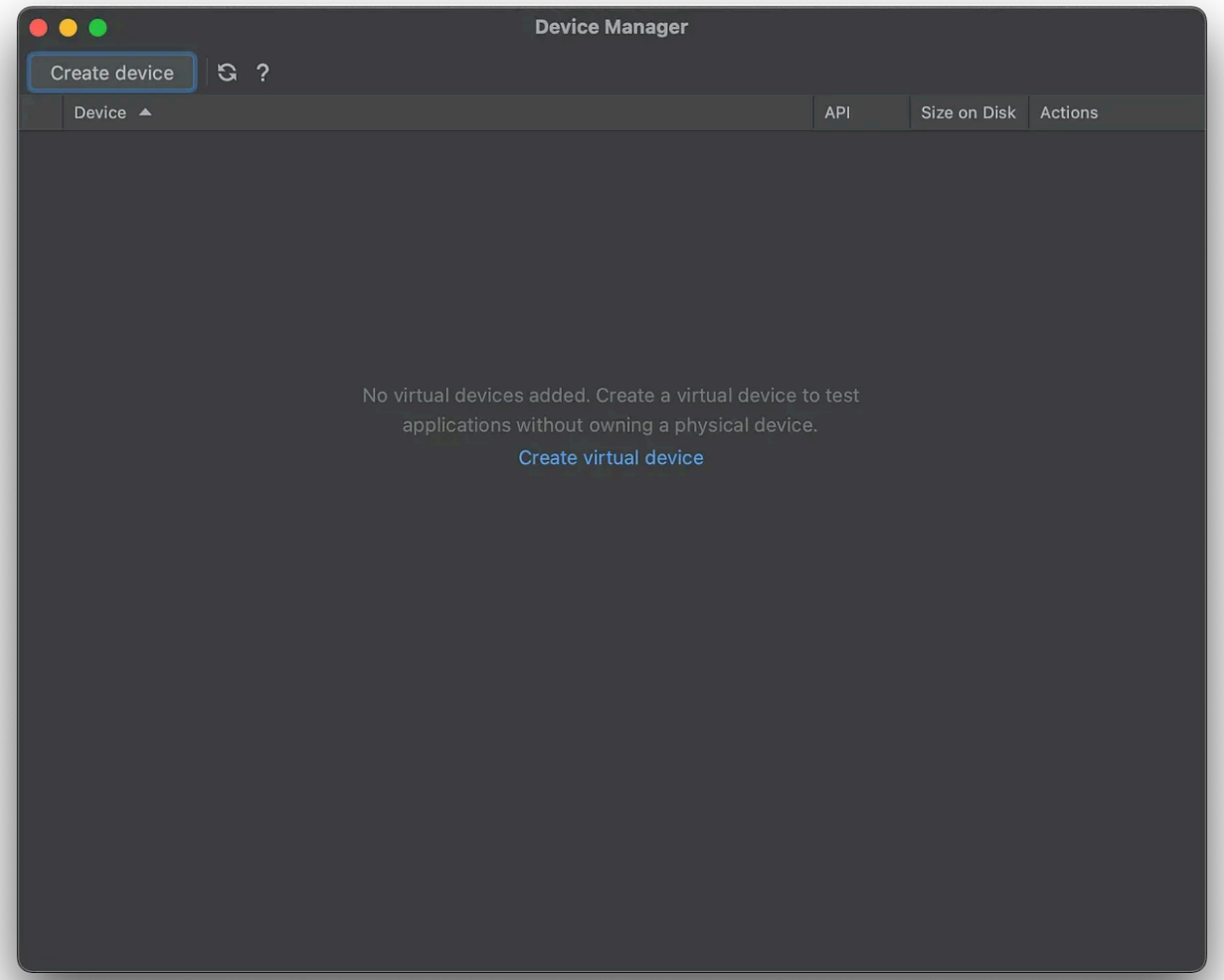
Click Virtual Device Manager to open up AVD manager

Licensed under CC BY-SA 4.0

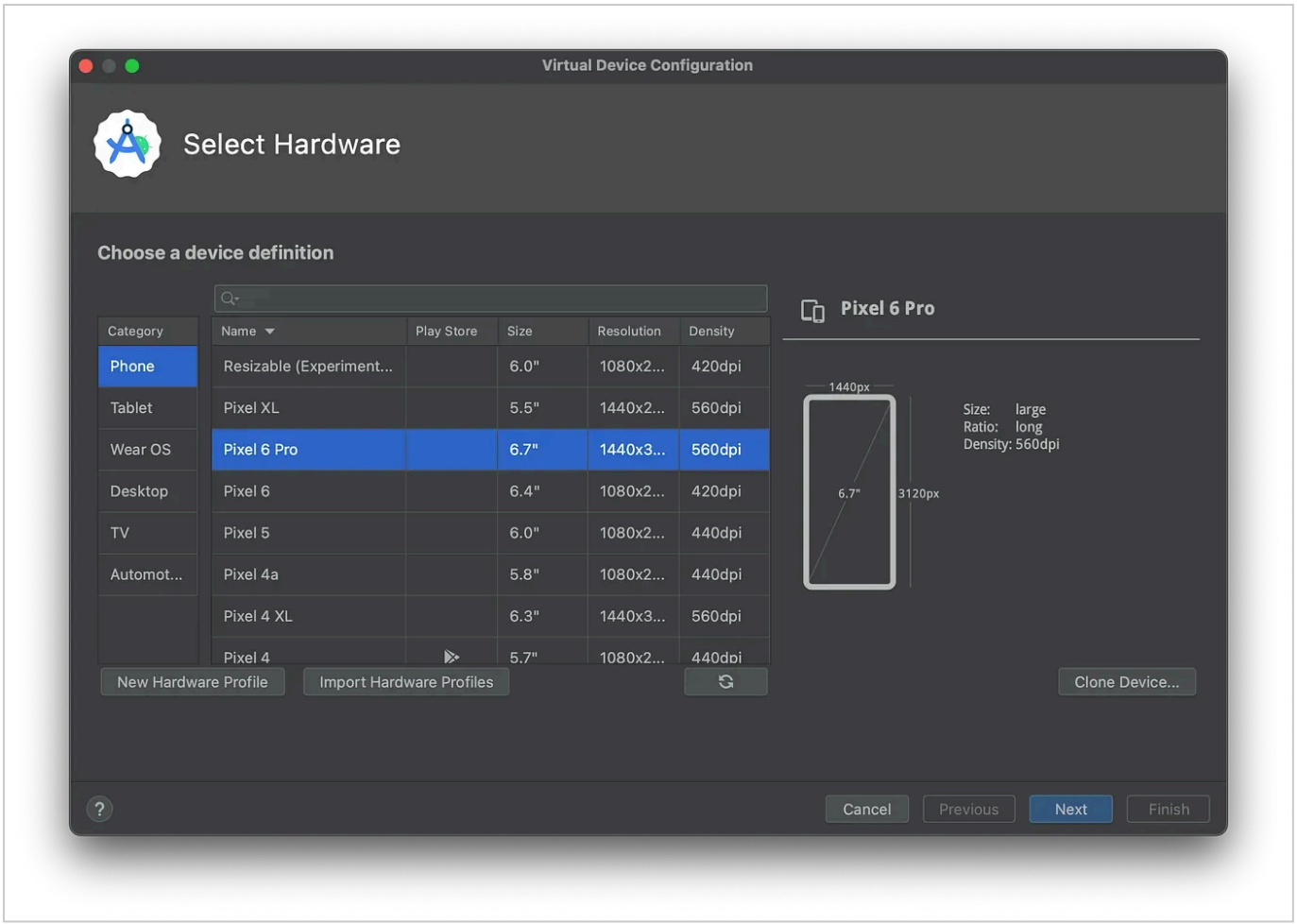


Click Create device and customized it as you need. If you need google play service, choose device with play store flag and system image that include Google Play API support.

Licensed under CC BY-SA 4.0

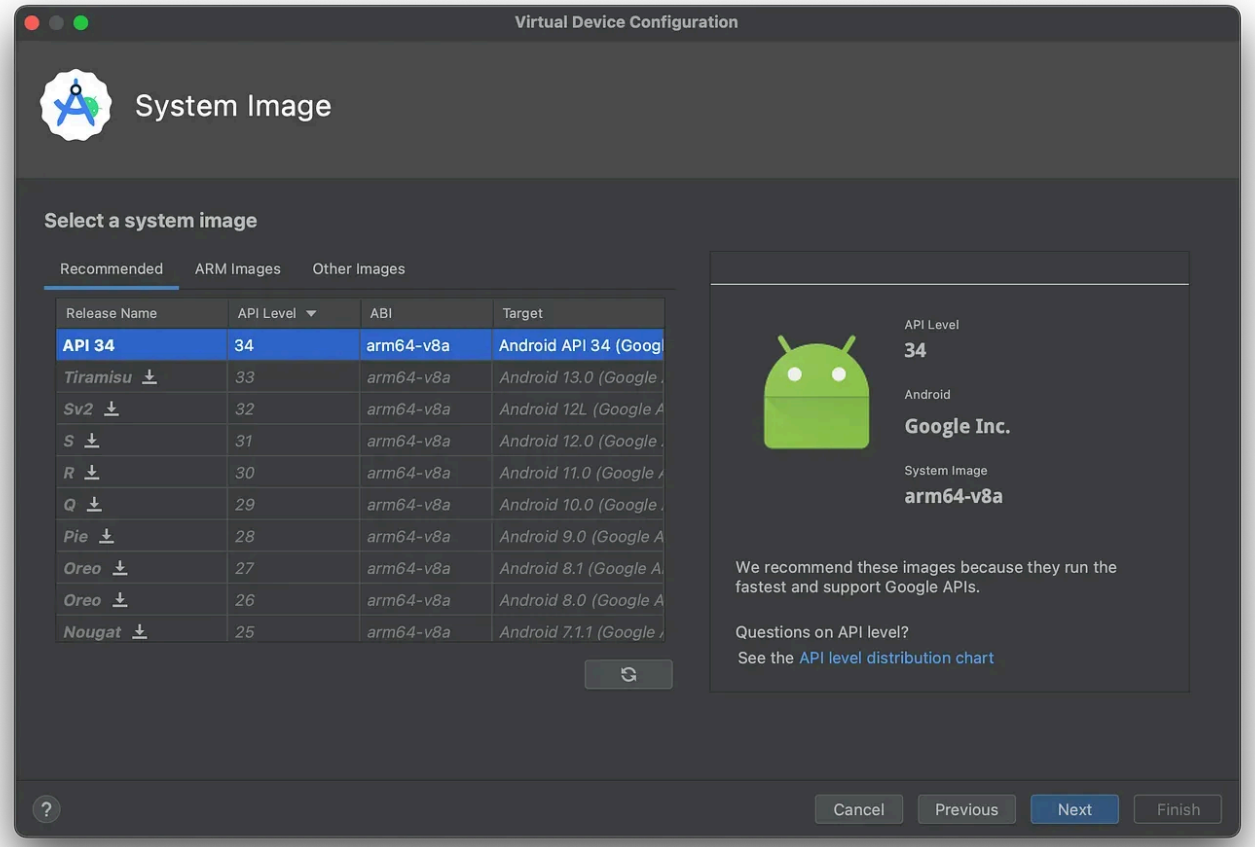


Licensed under CC BY-SA 4.0

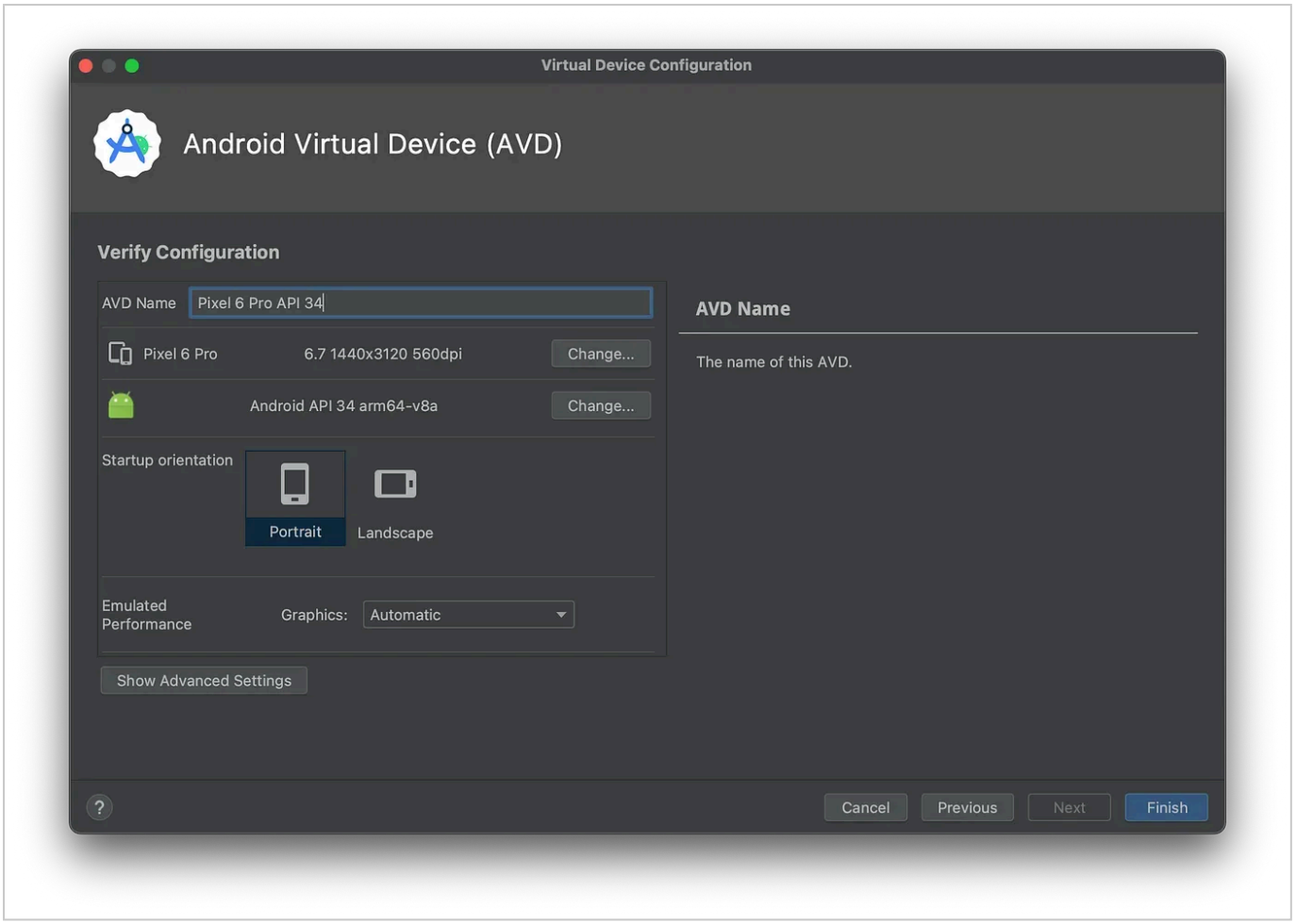


Choose system image that you want, in this case I used API 34(Android 14). Feel free to adjust as your need.

Licensed under CC BY-SA 4.0

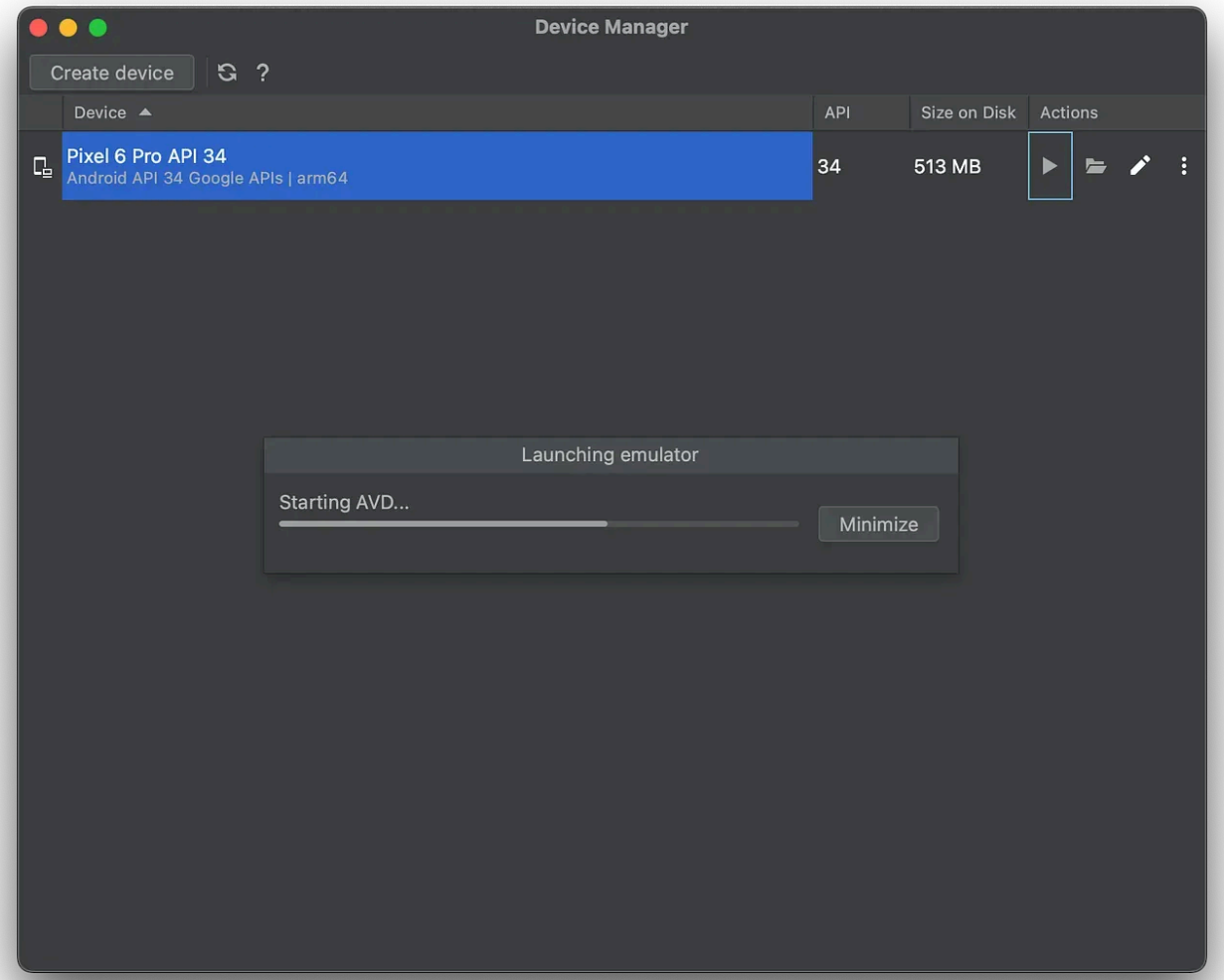


Licensed under CC BY-SA 4.0



If the previous process is done and succeed, start the AVD and wait until android is ready to use.

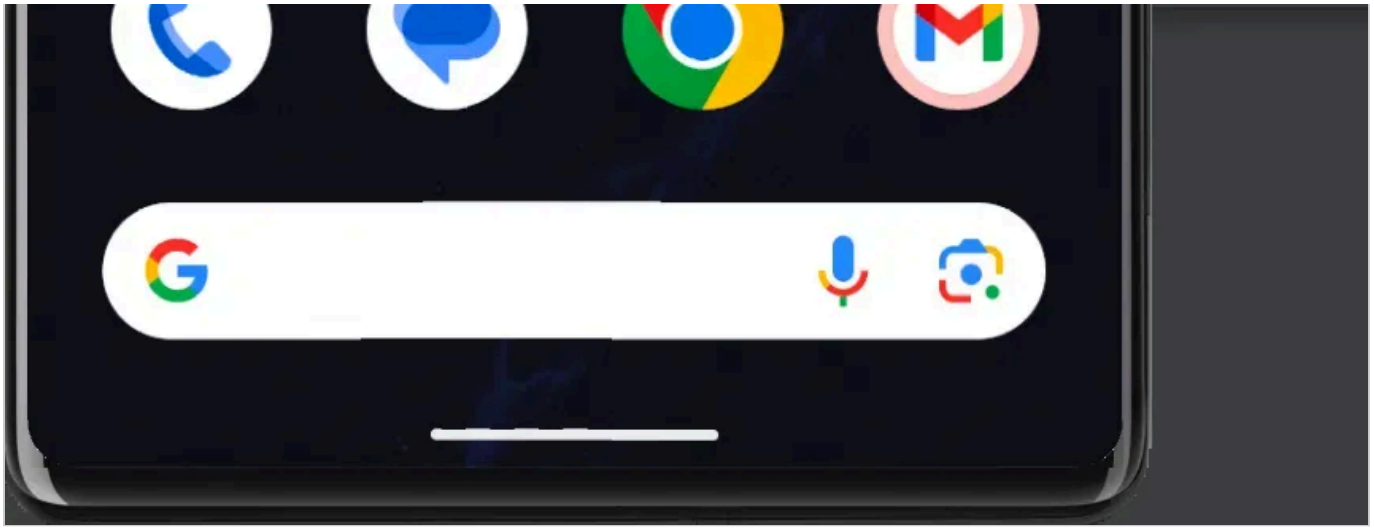
Licensed under CC BY-SA 4.0



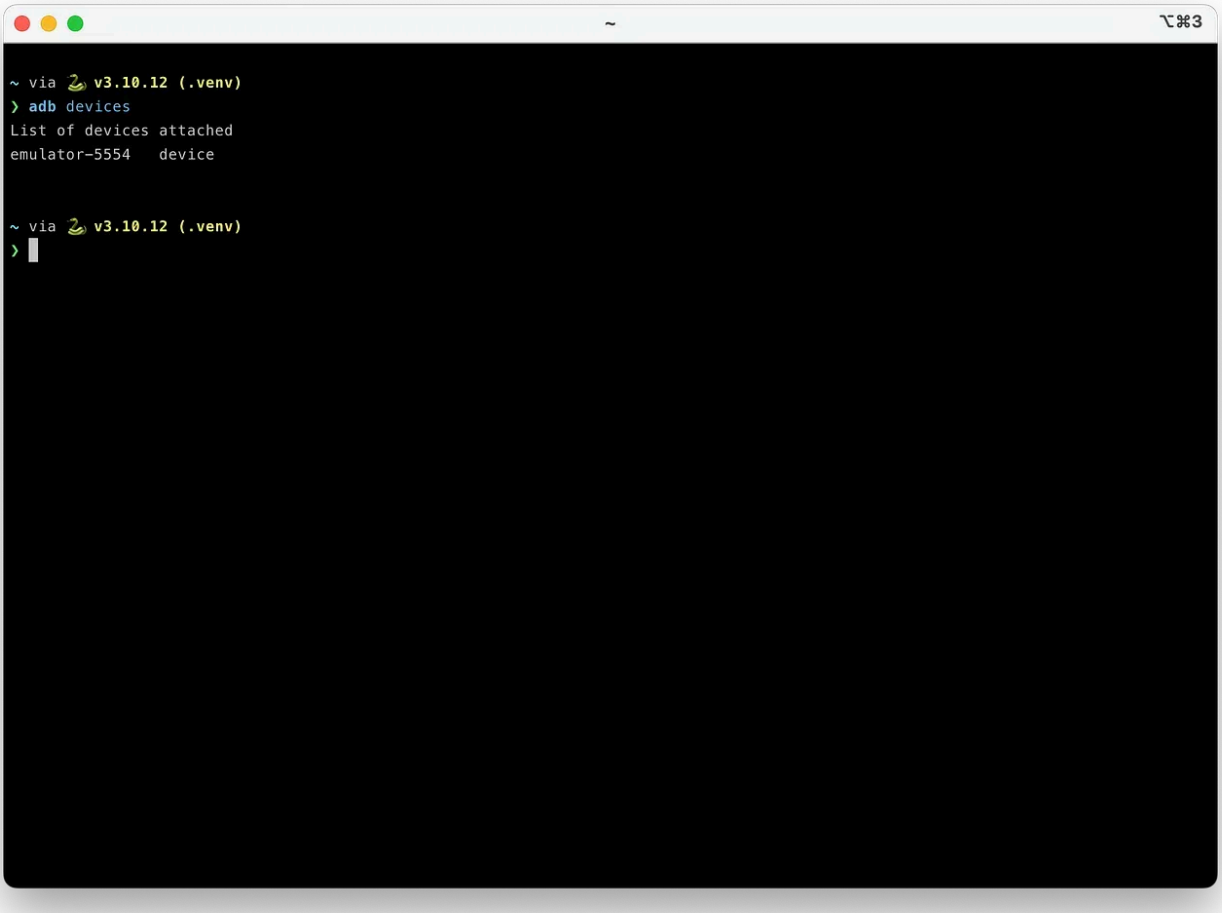
Licensed under CC BY-SA 4.0



Licensed under CC BY-SA 4.0

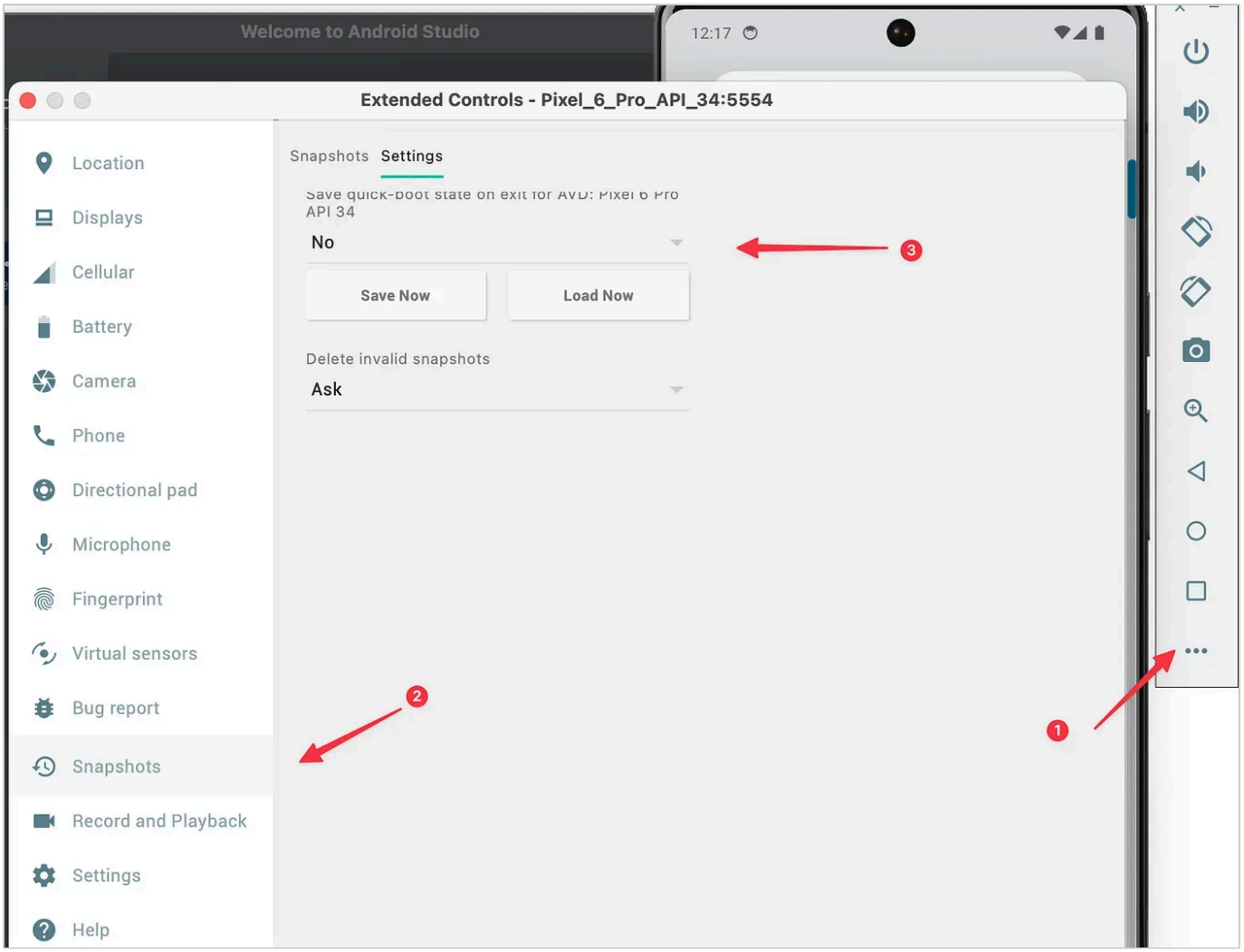


Check the `adb` connection between your mac and android vm to make sure that `adb` is connected/attached



WARNING: Before rooting the device, you need to disable snapshot for cold boot. If you forgot to disable the snapshot feature, the device will be going back to default state before rooting and all data after android rooted will gone.

Licensed under CC BY-SA 4.0



Disabling snapshot before rooting the device

For rooting our Android, we will use rootAVD from https://gitlab.com/newbit/rootAVD (https://gitlab.com/newbit/rootAVD). We need to git clone first.

```
❯ git clone https://gitlab.com/newbit/rootAVD && cd rootAVD
Cloning into 'rootAVD'...
remote: Enumerating objects: 501, done.
remote: Counting objects: 100% (175/175), done.
remote: Compressing objects: 100% (90/90), done.
remote: Total 501 (delta 126), reused 126 (delta 83), pack-reused 326
Receiving objects: 100% (501/501), 65.95 MiB | 7.21 MiB/s, done.
Resolving deltas: 100% (302/302), done.
```

Running the tool without argument will give us help message

```
❯ ./rootAVD.sh
rootAVD A Script to root AVD by NewBit XDA

Usage: rootAVD [DIR/ramdisk.img] [OPTIONS] | [EXTRA ARGUMENTS]
or: rootAVD [ARGUMENTS]

Arguments:
 ListAllAVDs   Lists Command Examples for ALL installed AVDs

 InstallApps   Just install all APKs placed in the Apps folder

Main operation mode:
 DIR    a path to an AVD system-image
     - must always be the 1st Argument after rootAVD

--------------------------------SNIP--------------------------------
```

For the first step, we need to run `rootAvd.sh ListAllAVDs` to know the available commands to run in command example section as shown below

```
❯ ./rootAVD.sh ListAllAVDs
rootAVD A Script to root AVD by NewBit XDA

Usage: rootAVD [DIR/ramdisk.img] [OPTIONS] | [EXTRA ARGUMENTS]
or: rootAVD [ARGUMENTS]

Arguments:
 ListAllAVDs   Lists Command Examples for ALL installed AVDs

 InstallApps   Just install all APKs placed in the Apps folder

----------------------------SNIP----------------------------

Extra Commands can be combined, there is no particular order.

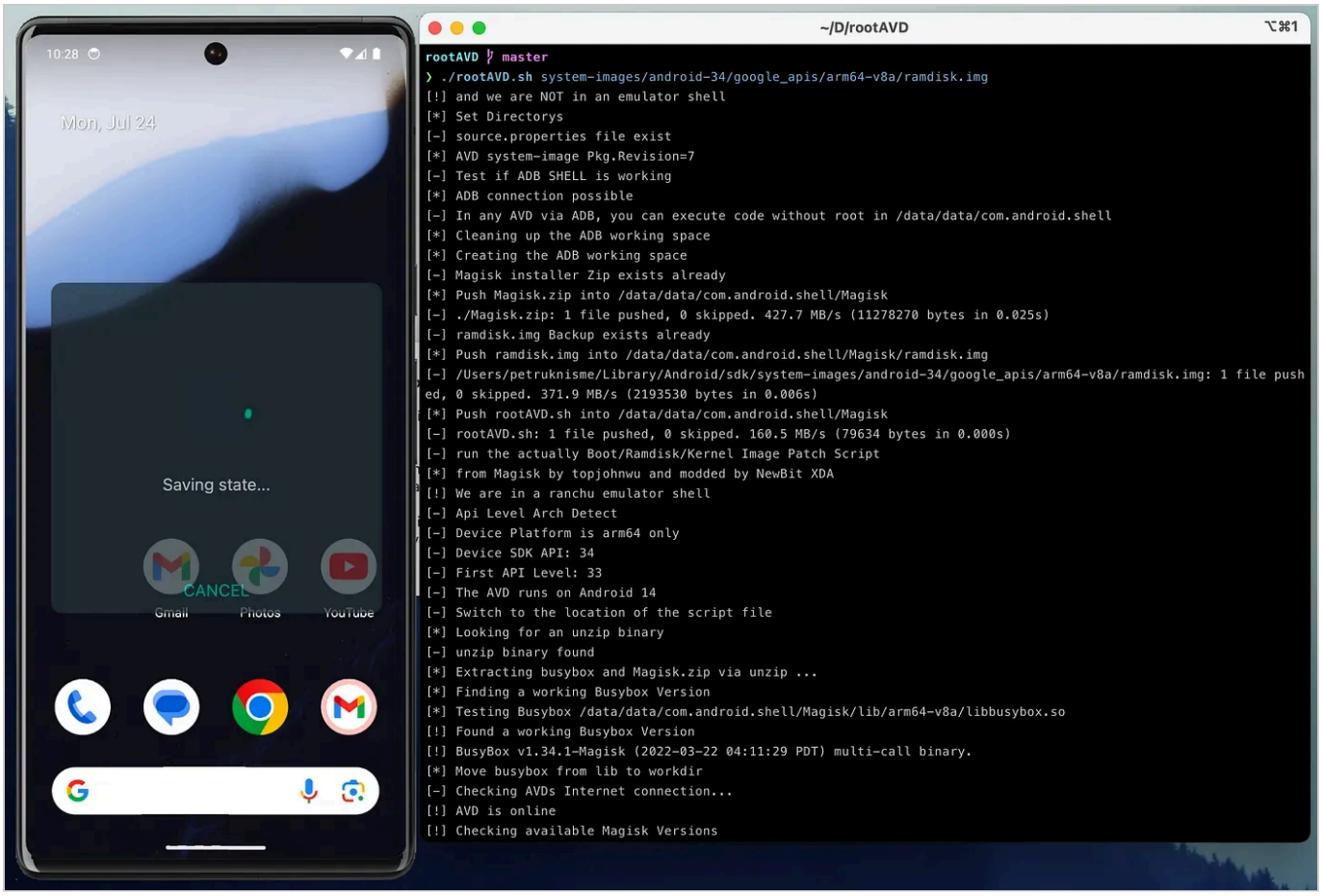
Notes: rootAVD will
- always create .backup files of ramdisk.img and kernel-ranchu
```

Licensed under CC BY-SA 4.0

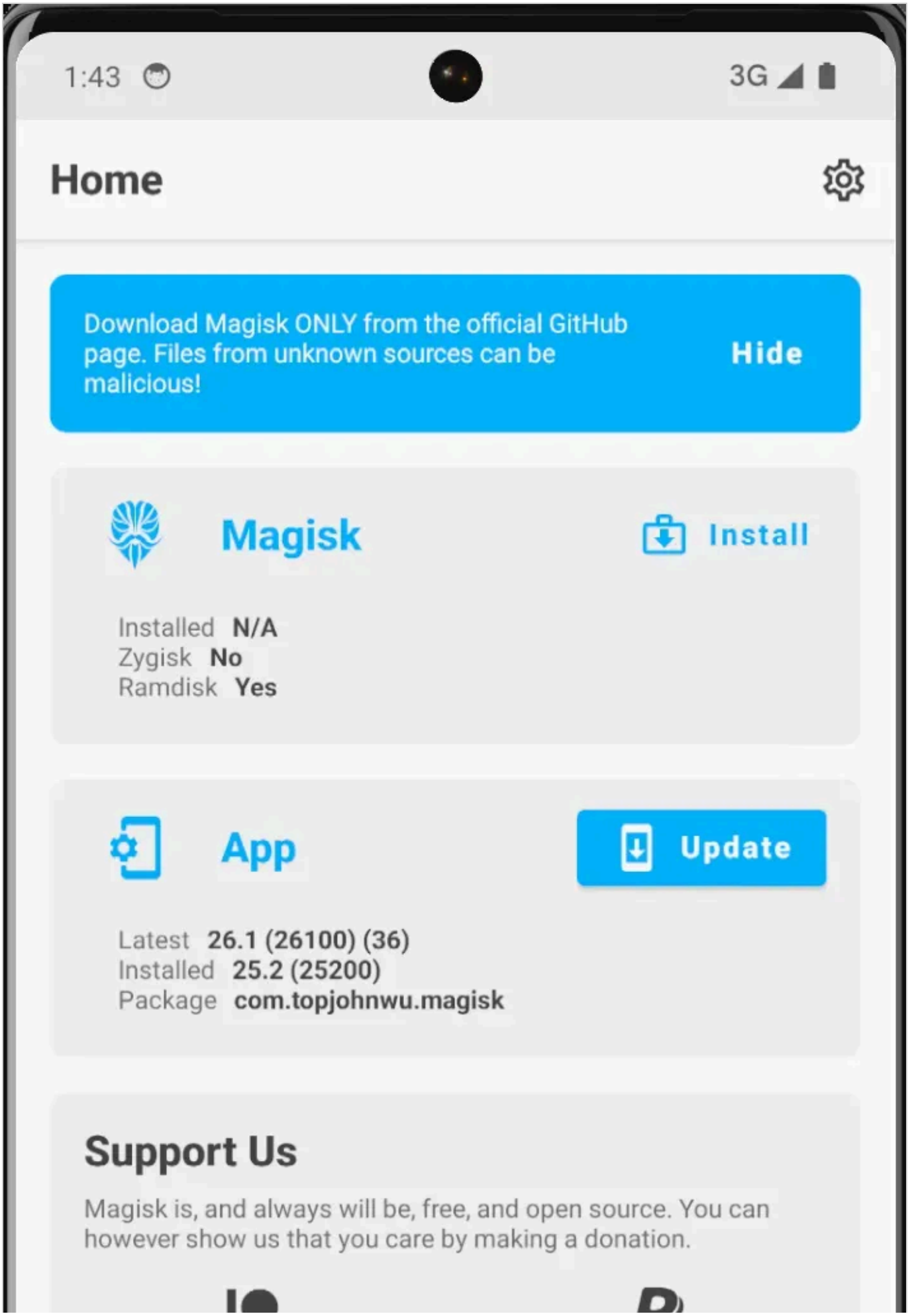
```
18  - replace both when done patching
19  - show a Menu, to choose the Magisk Version (Stable || Canary || Alpha), i
20  - make the choosen Magisk Version to its local
21  - install all APKs placed in the Apps folder
22  - use ~/Library/Android/sdk to search for AVD system images
23
24  Command Examples:
25  ./rootAVD.sh
26  ./rootAVD.sh ListAllAVDs
27  ./rootAVD.sh InstallApps
28
29  ./rootAVD.sh system-images/android-34/google_apis/arm64-v8a/ramdisk.img
30  ./rootAVD.sh system-images/android-34/google_apis/arm64-v8a/ramdisk.img FA
31  ./rootAVD.sh system-images/android-34/google_apis/arm64-v8a/ramdisk.img DE
32  ./rootAVD.sh system-images/android-34/google_apis/arm64-v8a/ramdisk.img re
33  ./rootAVD.sh system-images/android-34/google_apis/arm64-v8a/ramdisk.img In
34  ./rootAVD.sh system-images/android-34/google_apis/arm64-v8a/ramdisk.img In
35  ./rootAVD.sh system-images/android-34/google_apis/arm64-v8a/ramdisk.img In
36  ./rootAVD.sh system-images/android-34/google_apis/arm64-v8a/ramdisk.img Ad
```

As shown in the above output, there's available system-image at **`system-images/android-34/google_apis/arm64-v8a/ramdisk.img`**. For rooting the device, we can use this command **`./rootAVD.sh system-images/android-34/google_apis/arm64-v8a/ramdisk.img`**

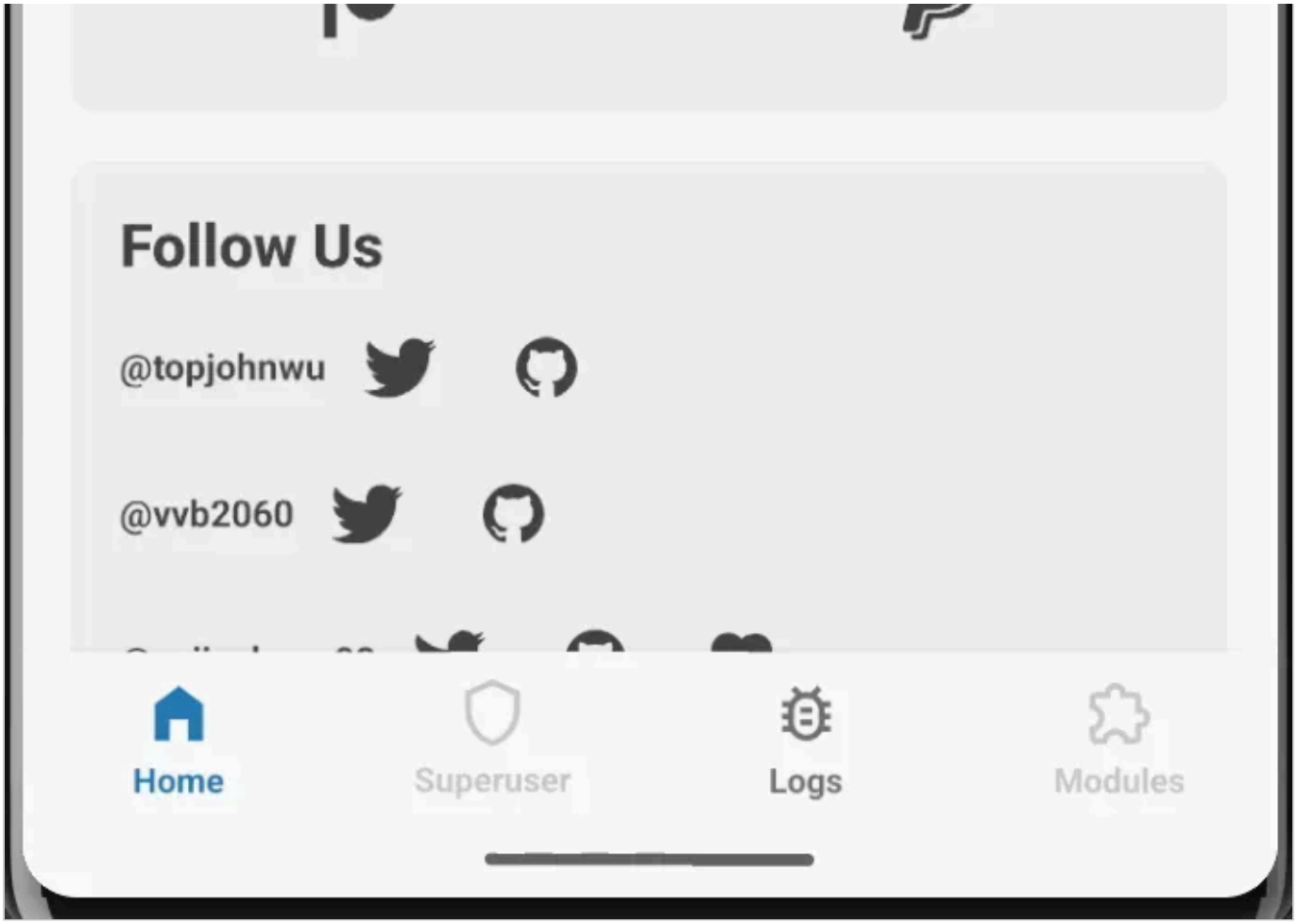
Licensed under CC BY-SA 4.0



Licensed under CC BY-SA 4.0



Licensed under CC BY-SA 4.0



Magisk Installed as N/A

As shown in the above picture that Installed status is N/A that means magisk is fail to installed. For the android 34, we unable to use the common method for rooting the device. But, instead we need to use `FAKEBOOTIMG` method.

```
❯ ./rootAVD.sh system-images/android-34/google_apis/arm64-v8a/ramdisk.img
[!] and we are NOT in an emulator shell
[*] Set Directorys
[-] source.properties file exist
[*] AVD system-image Pkg.Revision=7
[-] Test if ADB SHELL is working
[*] ADB connection possible
[-] In any AVD via ADB, you can execute code without root in /data/data/co
[*] Cleaning up the ADB working space
[*] Creating the ADB working space
[-] Magisk installer Zip exists already
[*] Push Magisk.zip into /data/data/com.android.shell/Magisk
[-] ./Magisk.zip: 1 file pushed, 0 skipped. 204.4 MB/s (11411692 bytes in
```

Licensed under CC BY-SA 4.0

```
[-] ramdisk.img Backup exists already
[*] Push ramdisk.img into /data/data/com.android.shell/Magisk/ramdisk.img
[-] /Users/user/Library/Android/sdk/system-images/android-34/google_apis/a

----------------------------------SNIP---------------------------
```

Script will ask you for the right magisk version to be installed, choose the version that suitable for you. For me, it's stable version.

```
[!] AVD is online
[!] Checking available Magisk Versions
[?] Choose a Magisk Version to install and make it local
[s] (s)how all available Magisk Versions
[1] local stable '26.1' (ENTER)
[2] stable 26.1
[3] canary d390ca2f(26104)
[4] alpha 46770db1-alpha(26103)
2 <--- type 2 and enter
[-] You choose Magisk stable Version 26.1
[*] Deleting local Magisk '26.1'
[*] Downloading Magisk stable 26.1
[!] Downloading Magisk stable 26.1 complete!
[-] Switch to the location of the script file
[*] Looking for an unzip binary
[-] unzip binary found
```

Then, the script will ask you to patch the generated img at `/sdcard/Download/fakeboot.img` manually

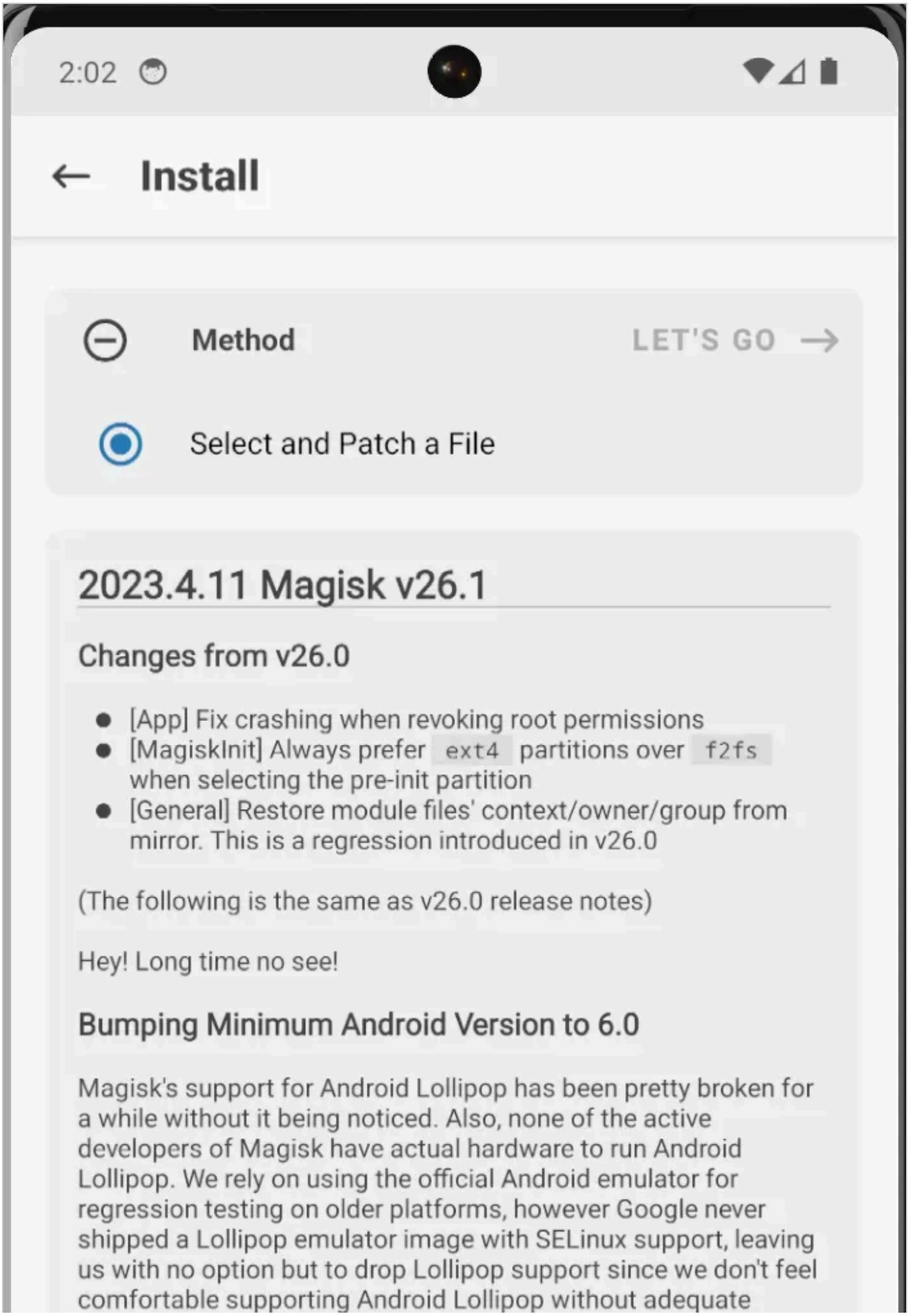
```
[*] After decompressing ramdisk.img, magiskboot will work
Detected format: [lz4_legacy]
[*] Processing fake Boot.img
[*] Creating a fake Boot.img
[-] removing old /sdcard/Download/fakeboot.img
[!] Only a minimal header is required for Magisk to repack the ramdisk
[*] repacking ramdisk.img into /sdcard/Download/fakeboot.img
[*] Searching for pre installed Magisk Apps
[!] Found a pre installed Magisk App, use it
[*] Detecting current user
```

Licensed under CC BY-SA 4.0

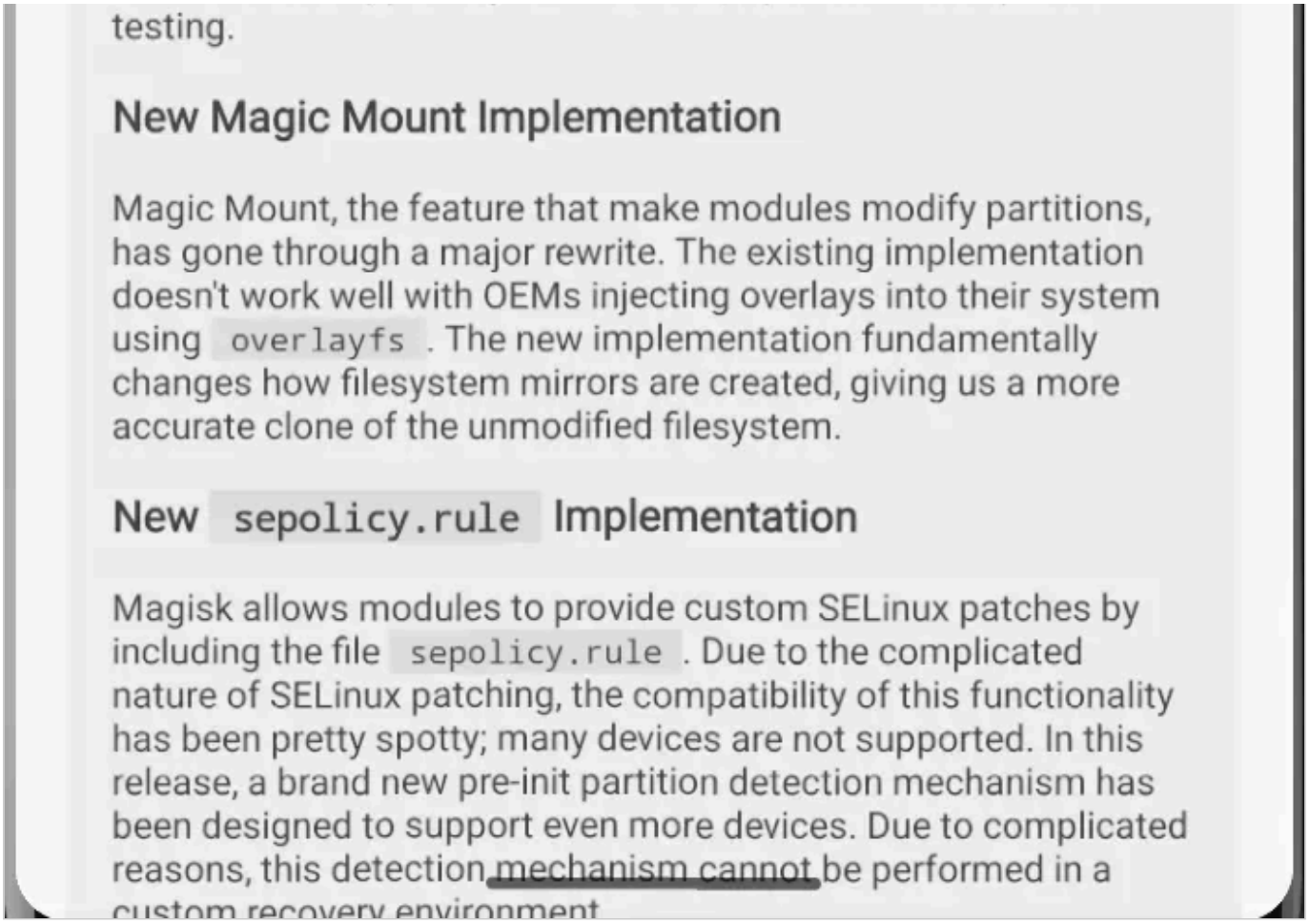
```
[-] Current user 0
[-] Starting Magisk
[*] Install/Patch /sdcard/Download/fakeboot.img and hit Enter when done(ma
```

Patch the **`img`** with this following step

Licensed under CC BY-SA 4.0

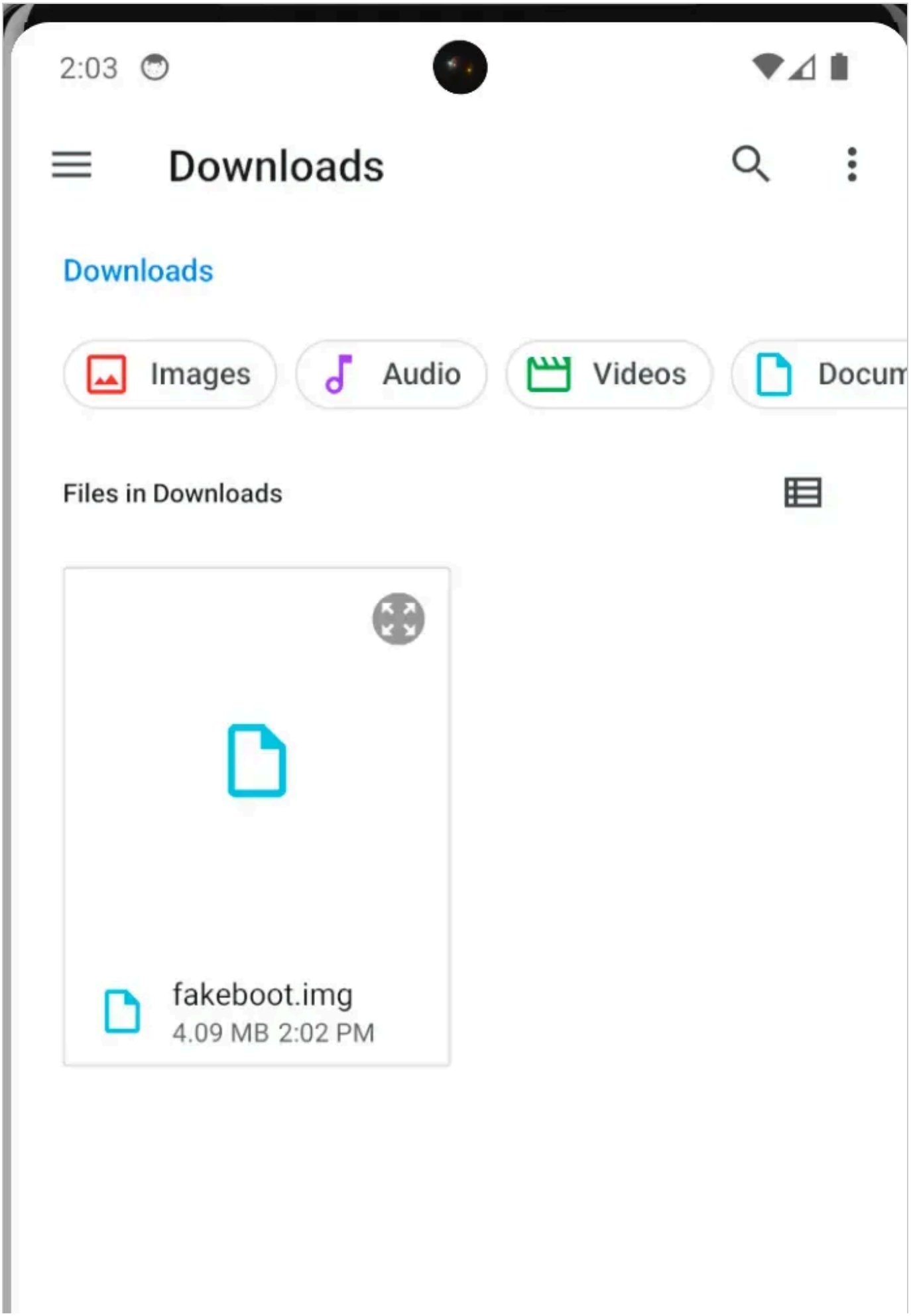


Licensed under CC BY-SA 4.0



Choose Select and Patch a File

Licensed under CC BY-SA 4.0



Licensed under CC BY-SA 4.0

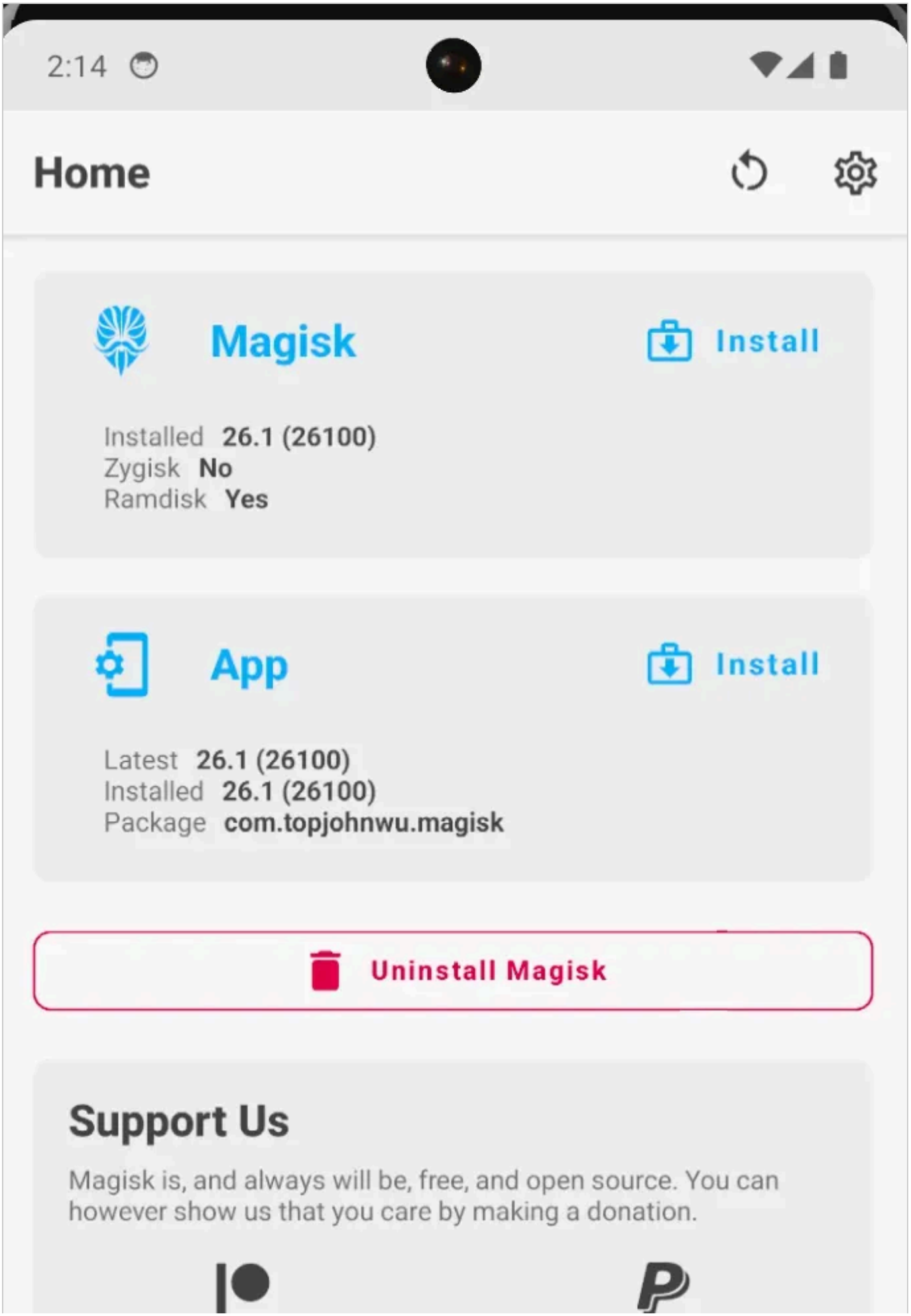


/sdcard/Download/fakeboot.img

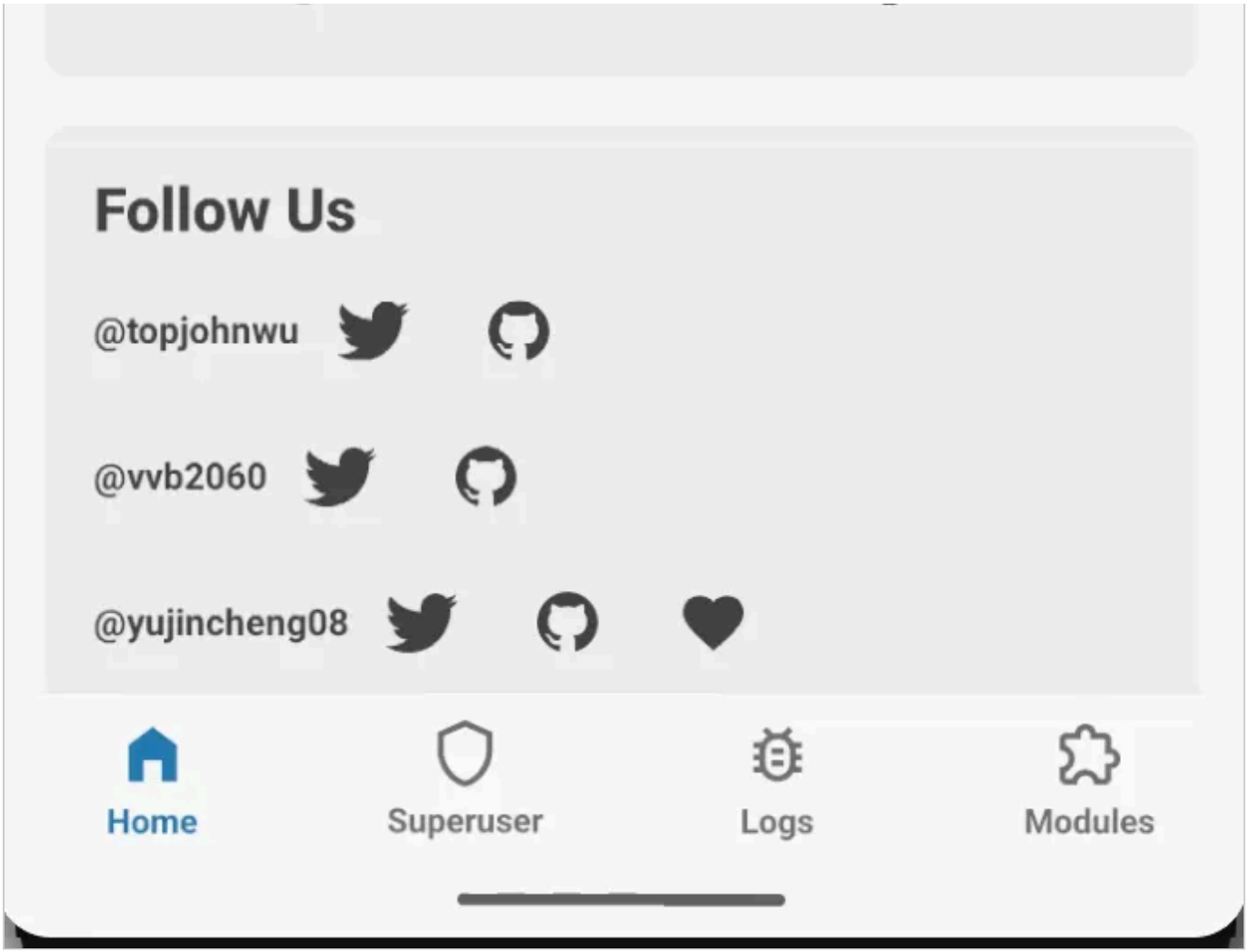
After patching the image, don’t forget to hit the Enter as mentioned in the script before. After that the script will automatically restart the device whenever the script is finished. As shown below, Magisk v26.1 is successfully installed.

Licensed under CC BY-SA 4.0

Licensed under CC BY-SA 4.0



Licensed under CC BY-SA 4.0

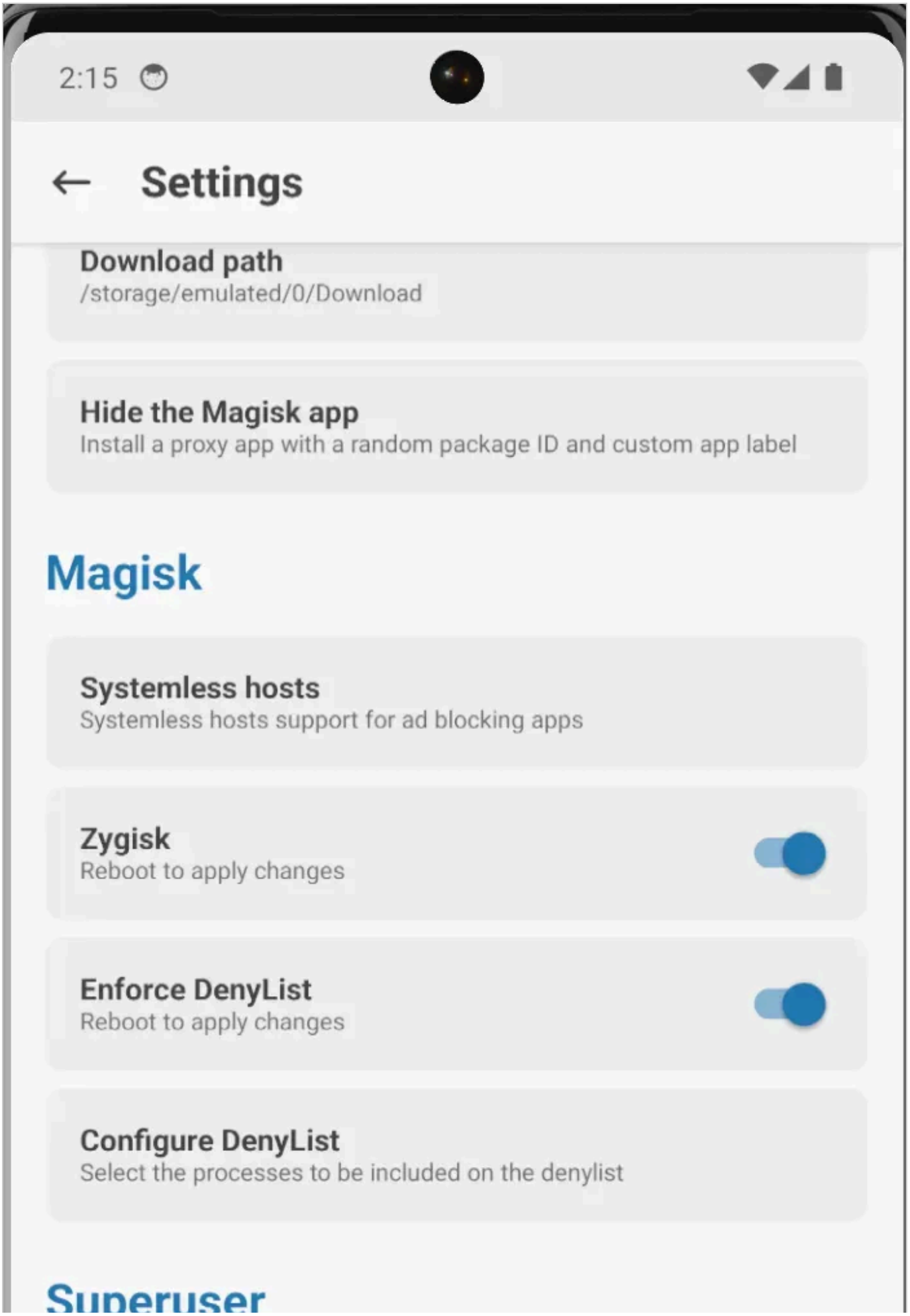


Magisk 26.1 Successfully installed

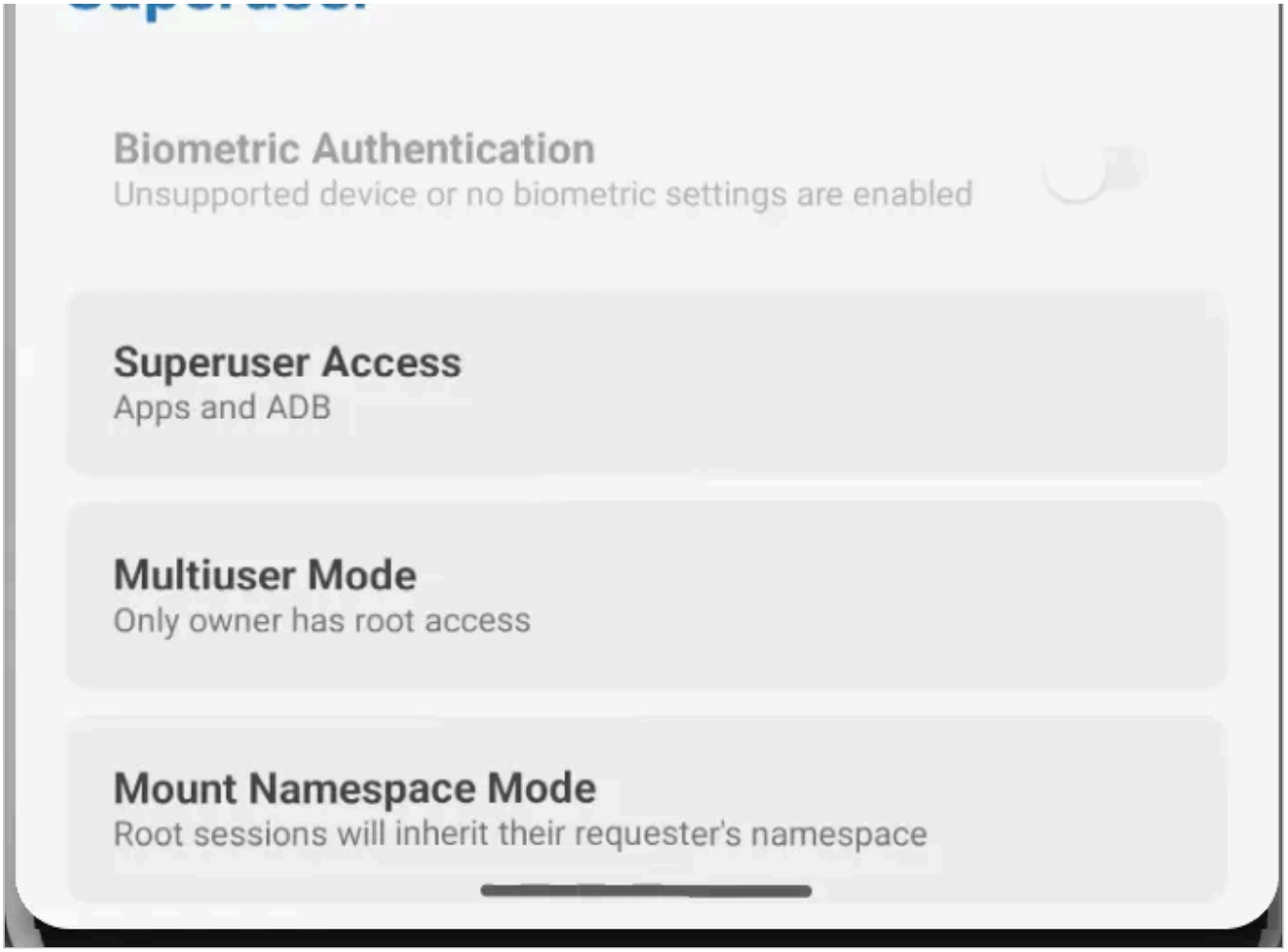
Don’t forget to enable Zygisk if you want to use root bypass feature in setting and then restart the device

Licensed under CC BY-SA 4.0

Licensed under CC BY-SA 4.0

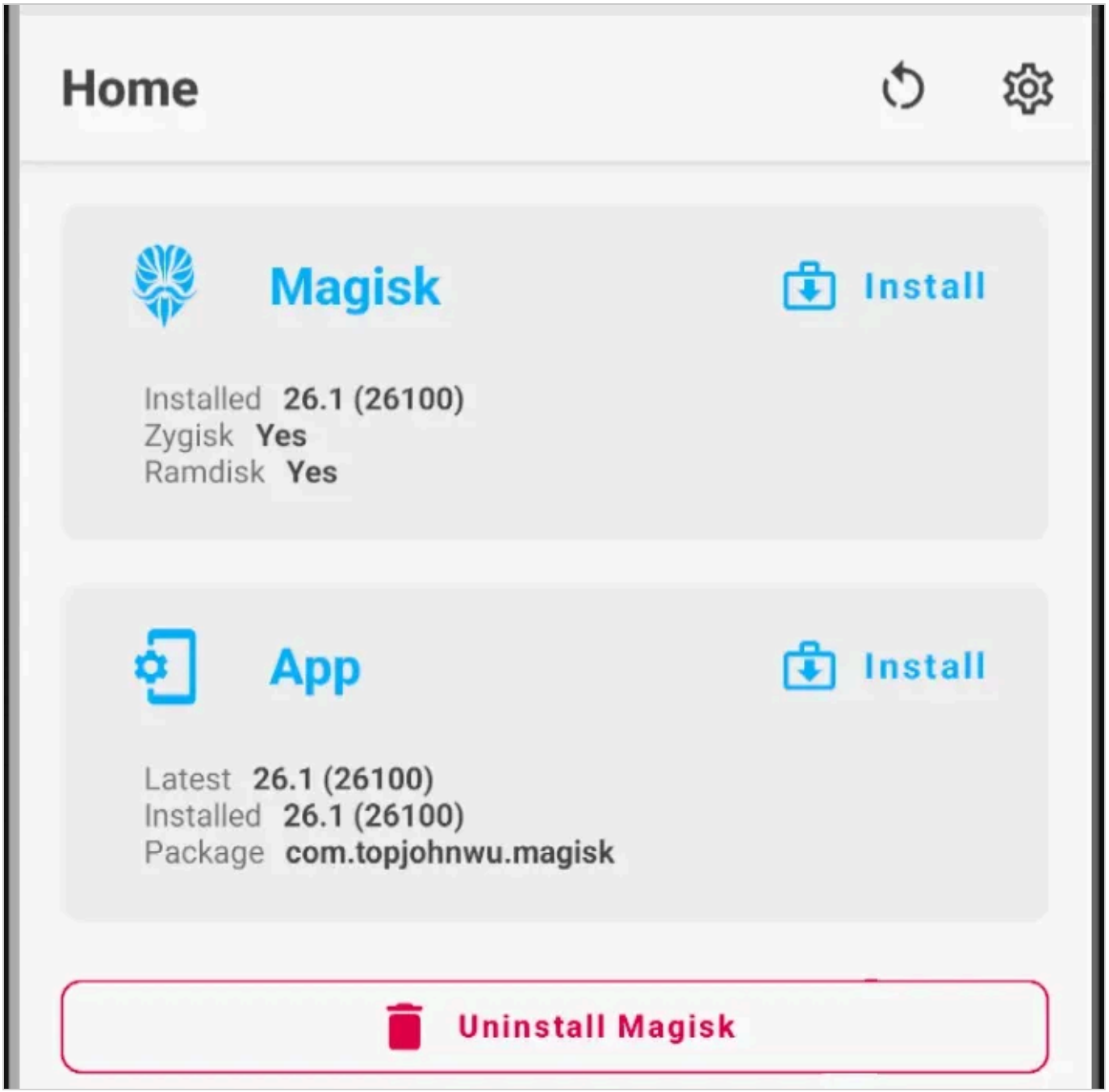


Licensed under CC BY-SA 4.0



Enabling Zygisk

Licensed under CC BY-SA 4.0



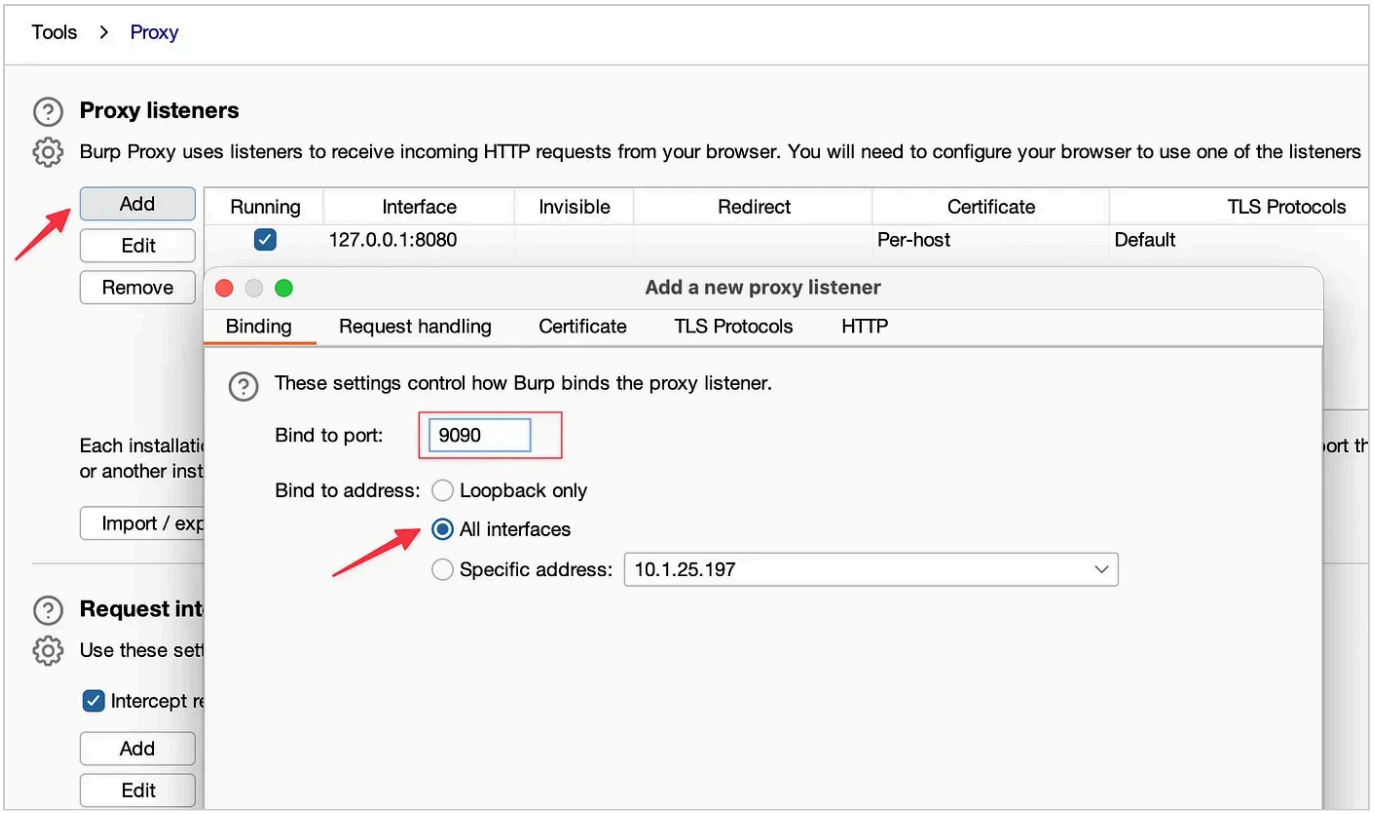
Zygisk Enabled

Now you are already rooted the virtual device with magisk and if you want to bypass root detection in the target app, just add the target app name in the **Configure Denylist**

Setting Burp Suite for Proxy Server

After we rooted the device, we will setup burp suite as proxy. So we able to modify the request before app send to the server.

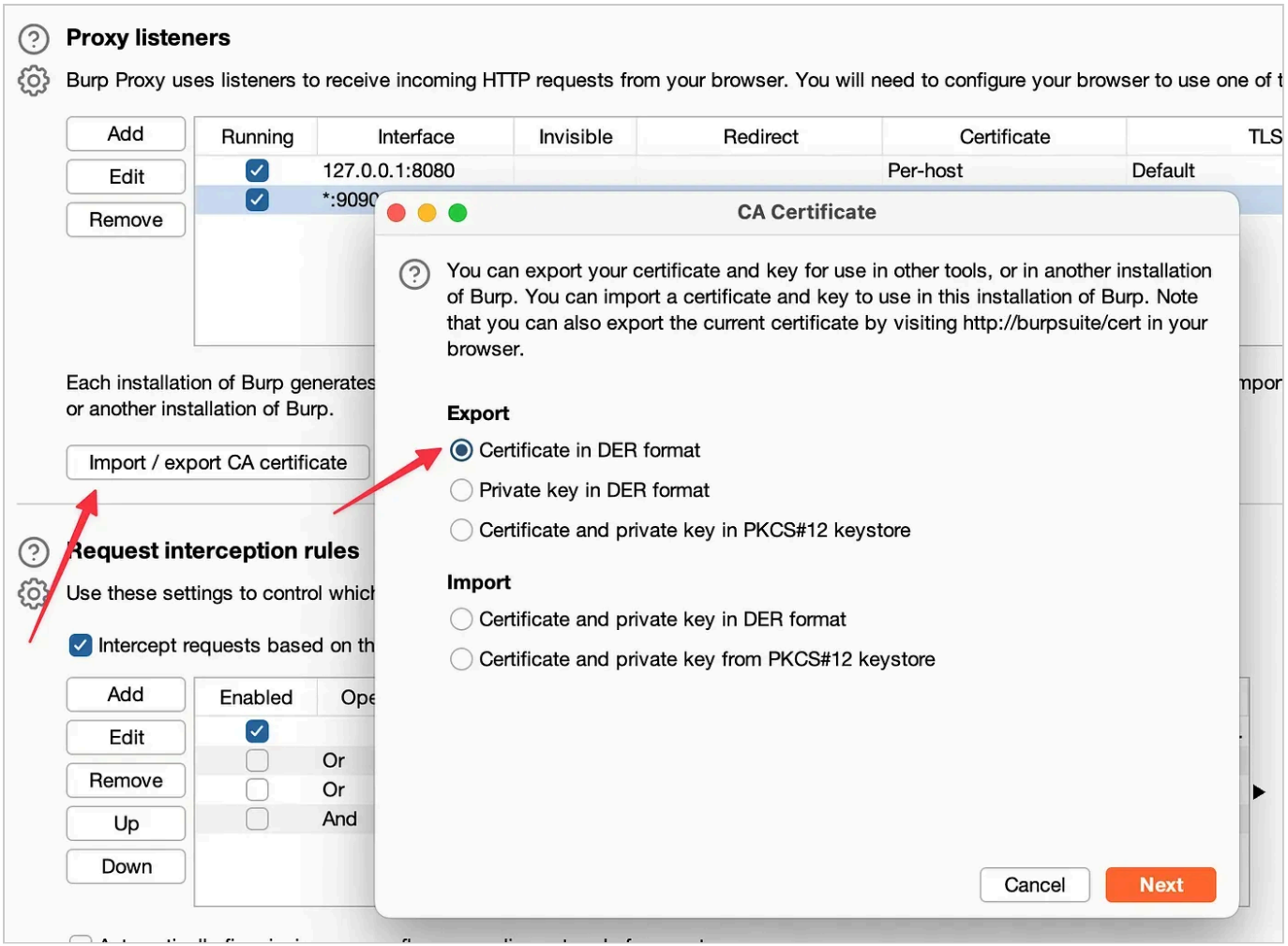
Licensed under CC BY-SA 4.0



Add new proxy listener

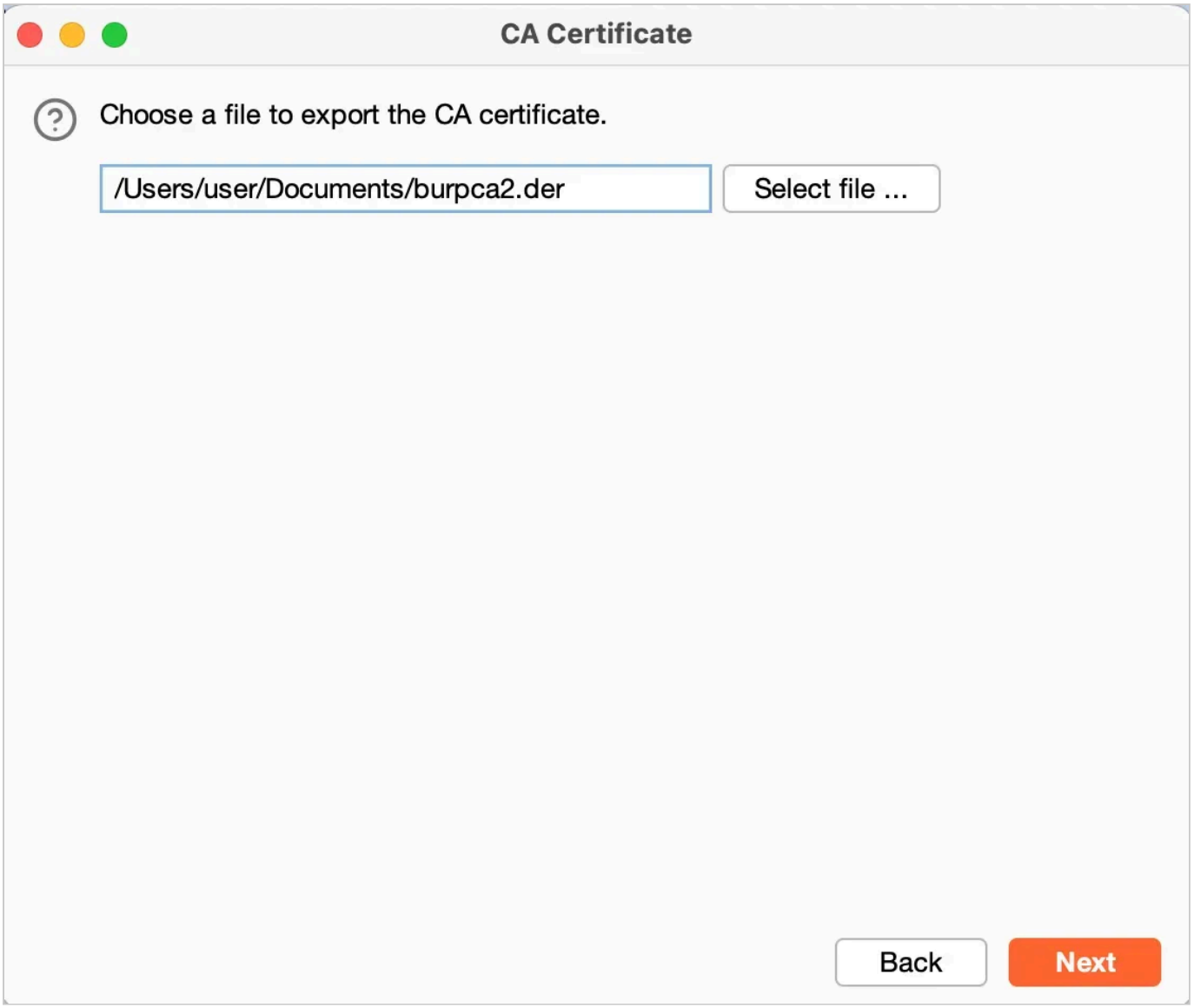
We need to export CA and Cert from Burpsuite, so android device able to trust the certificate when visiting website via our proxy.

Licensed under CC BY-SA 4.0

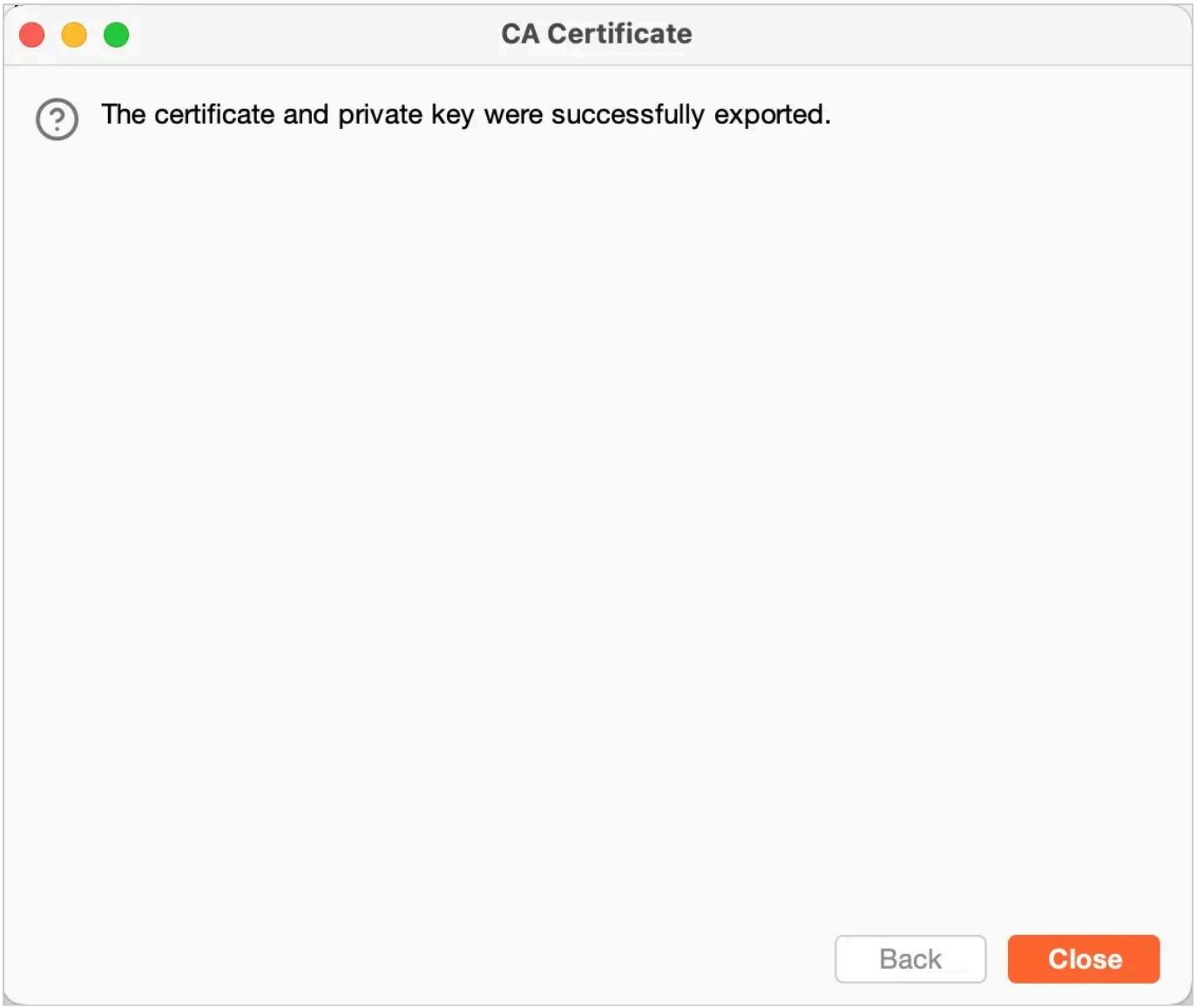


Export Burpsuite certificate

Licensed under CC BY-SA 4.0



Licensed under CC BY-SA 4.0



Success exporting burp CA certificate

After we success exporting burp CA Certificate, we need to transfer the file to the android device by using `adb` command

```
❯ adb push burpca.der /sdcard/Download/
burpca.der: 1 file pushed, 0 skipped. 1.7 MB/s (940 bytes in 0.001s)
```

Then we will need to install the CA certificate in this setting menu

Go to Android Settings → Security & Privacy → More Security & Privacy → Encryption & Credentials → Install a certificate → Choose CA Certificate.

Click Install Anyway and choose the previous pushed certificate in the `/sdcard/Download`.

Licensed under CC BY-SA 4.0

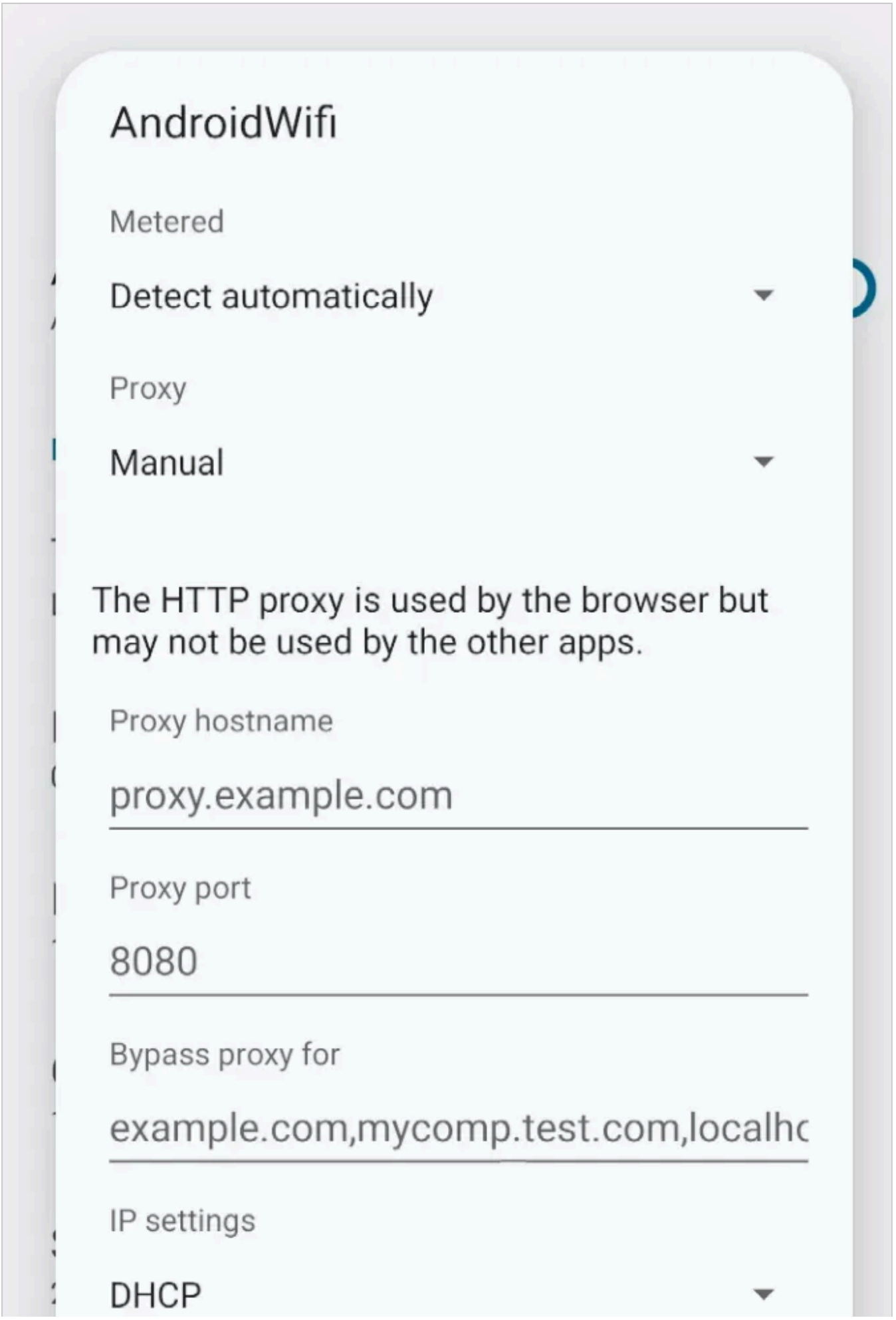
Setting Proxy in Android

There are many methods to modify the proxy in android:

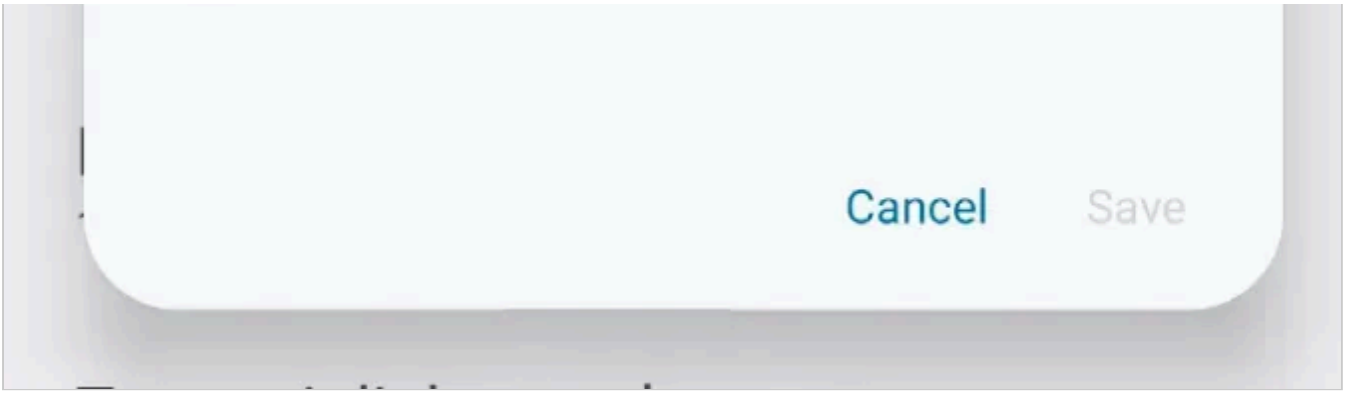
- Modify proxy via Android Wifi setting GUI

Go to Wifi Settings and then edit the wifi connection. Click Advance option and change the Proxy to manual and enter the burp ip and port

Licensed under CC BY-SA 4.0



Licensed under CC BY-SA 4.0



- Modify proxy via adb

To enable proxy using adb, we can use this command

```
❯ adb shell settings put global http_proxy IP_PROXY:PORT
```

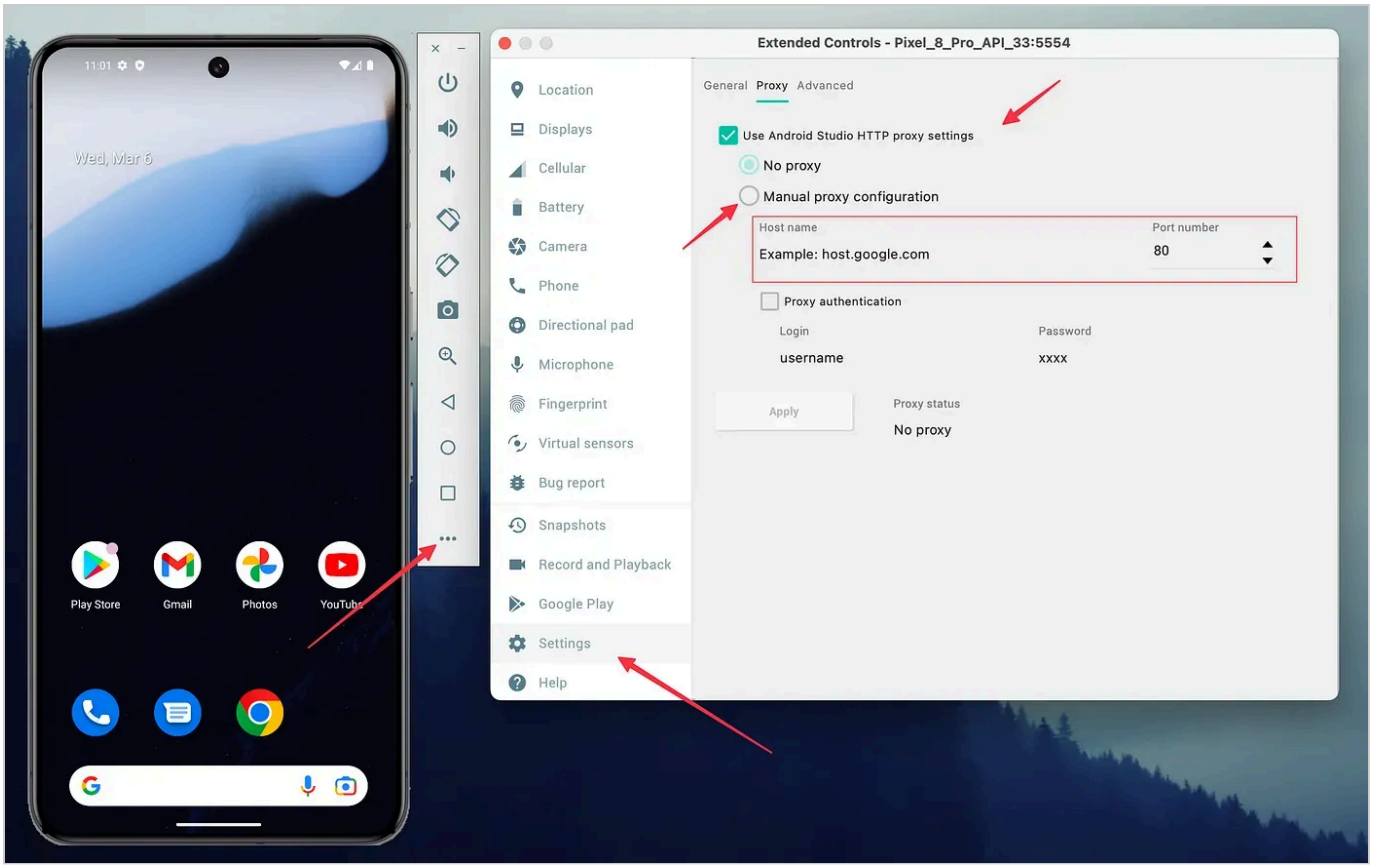
And to disable the proxy, we can use this command

```
❯ adb shell settings put global http_proxy :0
```

- Modify proxy via emulator settings

To setting the proxy in emulator, you can launch the emulator and then click Settings → Proxy. Then uncheck “Use Android Studio HTTP Proxy Settings” like the image below

Licensed under CC BY-SA 4.0



Setting proxy in Emulator

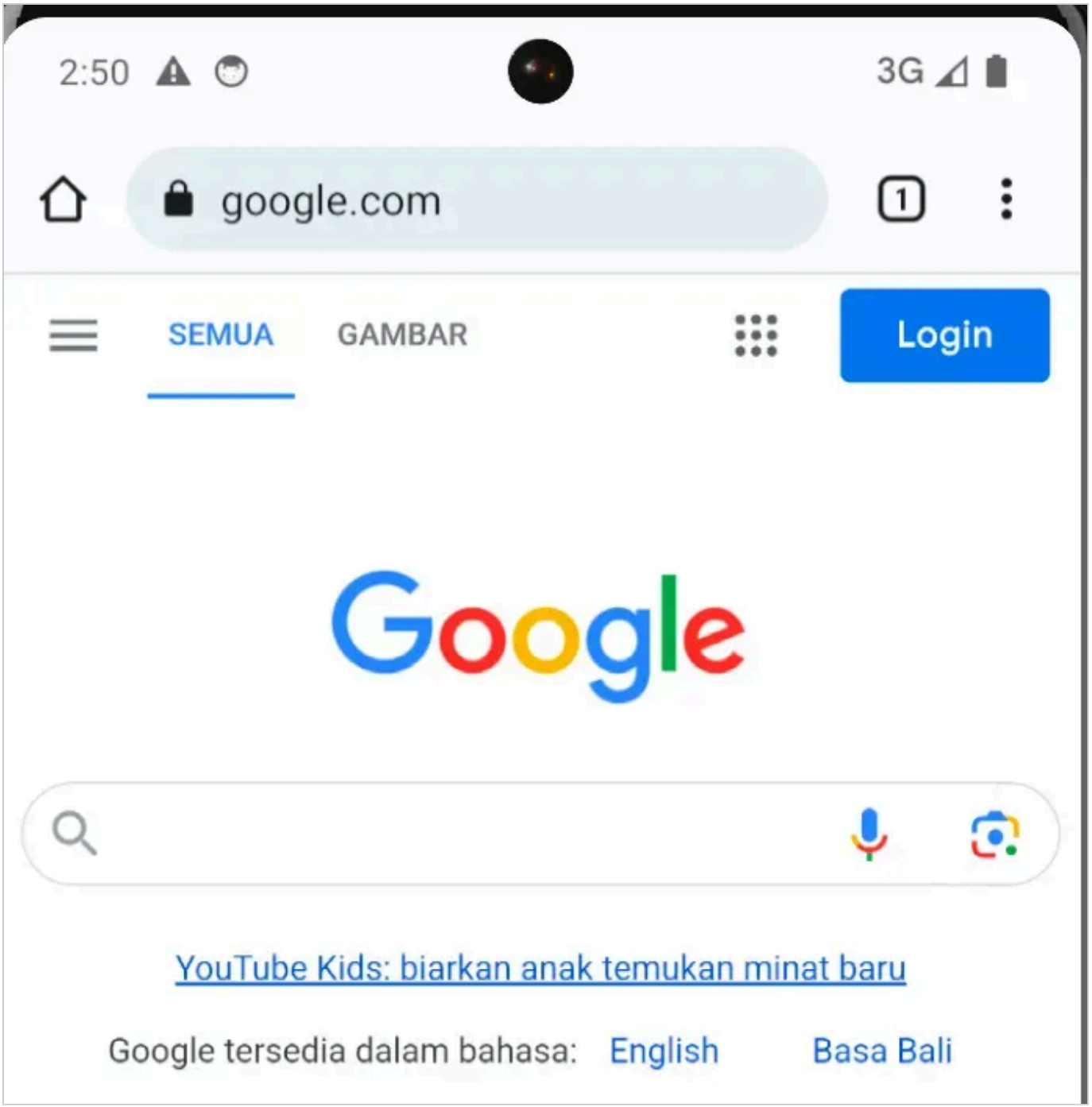
- Modify proxy via third party app

There are many third party app, such as:

- proxydroid(https://play.google.com/store/apps/details?id=org.proxydroid (https://play.google.com/store/apps/details?id=org.proxydroid&hl=id))
- Proxy-toggle https://github.com/theappbusiness/android-proxy-toggle (https://github.com/theappbusiness/android-proxy-toggle)

After that, try to visit google.com and it will show you that connection is secure with green lock in the browser.

Licensed under CC BY-SA 4.0



Installing frida

For more information about Frida, you can read at ***https://frida.re*** (https://frida.re/) /

I'm using virtualenv in MacOS, feel free to adjust it with your environment. Install `frida-tools` from `pip`

```
❯ pip -V
pip 23.0.1 from /nix/store/v0a077jw3gvdysh8m9l07a9khxnzgmll-python3-3.10.1
```

Licensed under CC BY-SA 4.0

```
❯ python -V
Python 3.10.12
❯ pip install frida frida-tools
Collecting frida
  Obtaining dependency information for frida from https://files.pythonhost
  Downloading frida-16.1.3-cp37-abi3-macosx_11_0_arm64.whl.metadata (2.0 k
Collecting frida-tools
  Using cached frida_tools-12.2.1-py3-none-any.whl
Requirement already satisfied: typing-extensions in ./.venv/lib/python3.10
Requirement already satisfied: colorama<1.0.0,>=0.2.7 in ./.venv/lib/pytho
Requirement already satisfied: prompt-toolkit<4.0.0,>=2.0.0 in ./.venv/lib
Requirement already satisfied: pygments<3.0.0,>=2.0.2 in ./.venv/lib/pytho
Requirement already satisfied: wcwidth in ./.venv/lib/python3.10/site-pack
Using cached frida-16.1.3-cp37-abi3-macosx_11_0_arm64.whl (30.5 MB)
Installing collected packages: frida, frida-tools
Successfully installed frida-16.1.3 frida-tools-12.2.1
```

After we've successfully installed frida- **16.1.3** and frida-tools-12.2.0, for frida to work in our android/ios we need to setup the frida-server.

Setup frida-server in Android

Go to https://github.com/frida/frida/releases (https://github.com/frida/frida/releases) and choose the frida-server-version-android-arm.xz (https://github.com/frida/frida/releases/download/15.2.2/frida-server-15.2.2-android-arm.xz), change the version with your installed frida version like I bold above. Frida client and server must have the **same version**.

Download the frida to host machine

```
❯ wget https://github.com/frida/frida/releases/download/16.1.3/frida-serve
```

```
Length: 6824388 (6.5M) [application/octet-stream]
Saving to: 'frida-server-16.1.3-android-arm64.xz'frida-server-15.2.2-andr
```

Extract xz file

```
```

```
❯ xz -d frida-server-16.1.3-android-arm64.xz
❯ ls | grep frida
frida-server-16.1.3-android-arm64
❯ mv frida-server-16.1.3-android-arm64 frida-server
```

After downloading, rename the file to `frida-server`, the next step is to copy the file to android. In my case, I copy the file to `/data/local/tmp/`

```
❯ adb push frida-server /data/local/tmp
frida-server: 1 file pushed, 0 skipped. 23.1 MB/s (19972988 bytes in 0.824
```

Run the `frida-server`

```
❯ adb shell
emu64a:/ $ su
emu64a:/ # cd /data/local/tmp/
emu64a:/data/local/tmp # chmod +x frida-server
emu64a:/data/local/tmp # ./frida-server &
```

To check if `frida-server` is sucessfully running, we can use command `ps` and `grep`

```
1|emu64a:/data/local/tmp # ps | grep frida
root          23299 23280   39280  19640 poll_schedule_timeout f10486e0 S f
```

We've done with setuping frida in android.

For now, we are already reaching the end section of thisarticle. Good luck on pentesting android \m/

Licensed under CC BY-SA 4.0

Originally published on **Medium** (https://medium.com/@petruknisme/complete-beginners-guide-to-setup-android-pentest-lab-without-physical-device-4f7b9e945d11).

Android # Security # Tutorial

Licensed under CC BY-SA 4.0