# NSSII Exercise 2

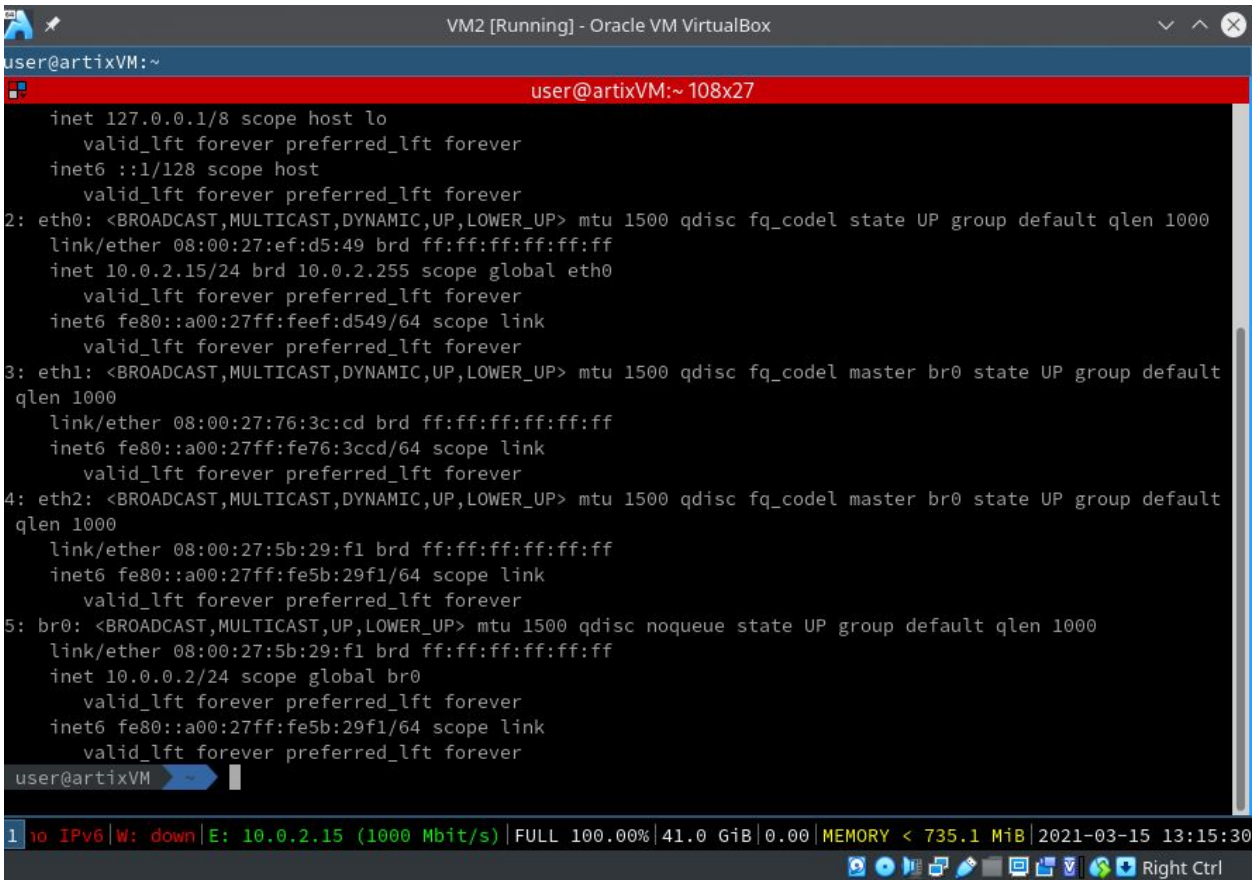
Manavjeet Singh, 2018295

| VMs | IP address | Comments |
|---|---|---|
| VM1 | 10.0.0.1/24 | Client in Task 2 and 3 |
| VM2 | 10.0.0.2/24 | Server in Task 2 and 3 |
| VM3 | 10.0.0.3/24 | - |
| VM4 | 10.0.0.4/24 | - |
| VM5 | 10.0.0.5/24 | - |

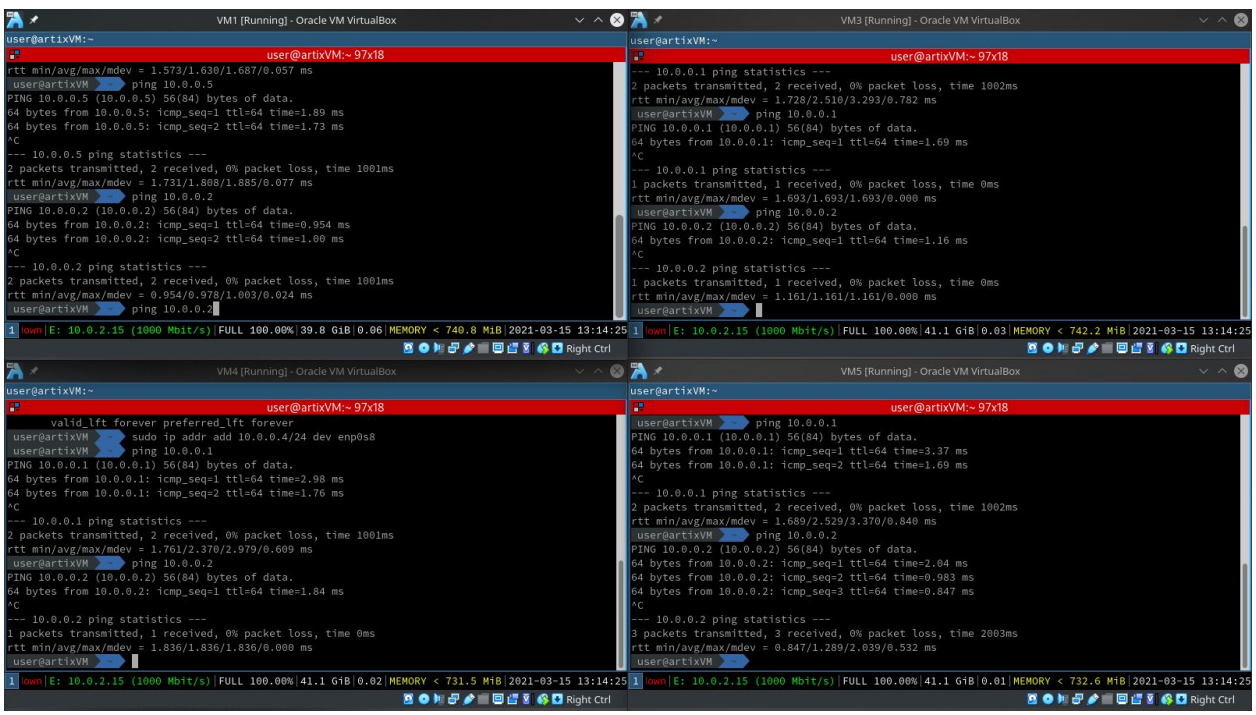
# Part 1

Creating Bridge:

- `sudo ifconfig eth1 0.0.0.0`
- `sudo ifconfig eth2 0.0.0.0`
- `sudo brctl addbr br0`
- `sudo brctl addif br0 eth1 eth2`
- `sudo ip addr add 10.0.0.2/24 dev br0`
- `sudo ip link set dev br0 up`



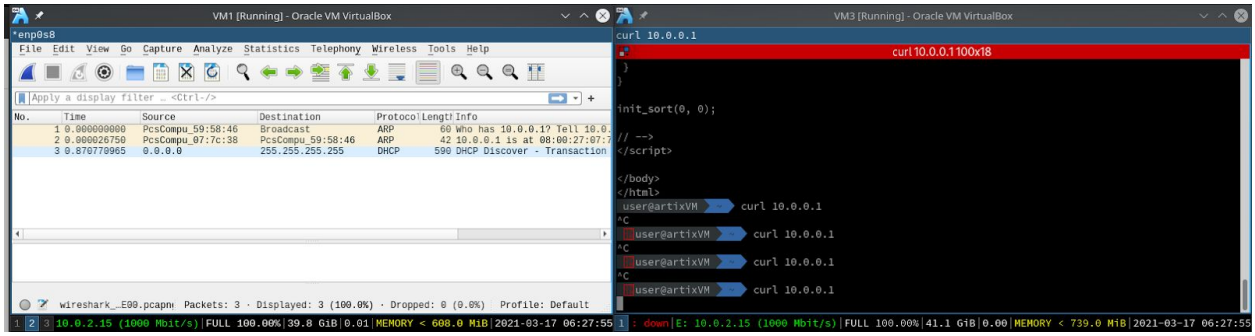

Bridge settings



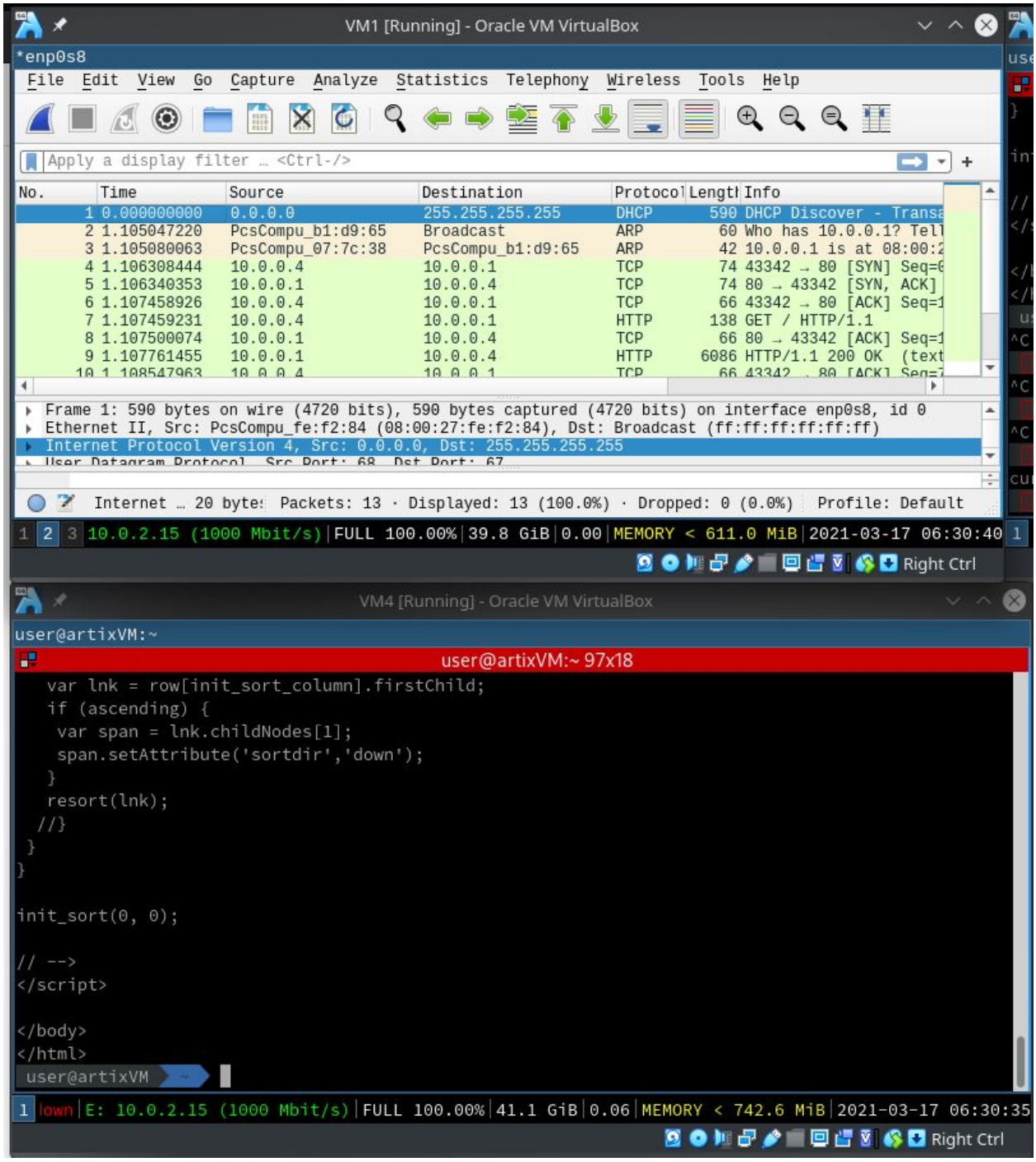

Machines are able to ping each other

1. To block VM1 from VM3: `sudo ebtables -I FORWARD -s 08:00:27:59:58:46 -d 08:00:27:07:7c:38 -j DROP`

   When accessing server on VM1 from VM3:

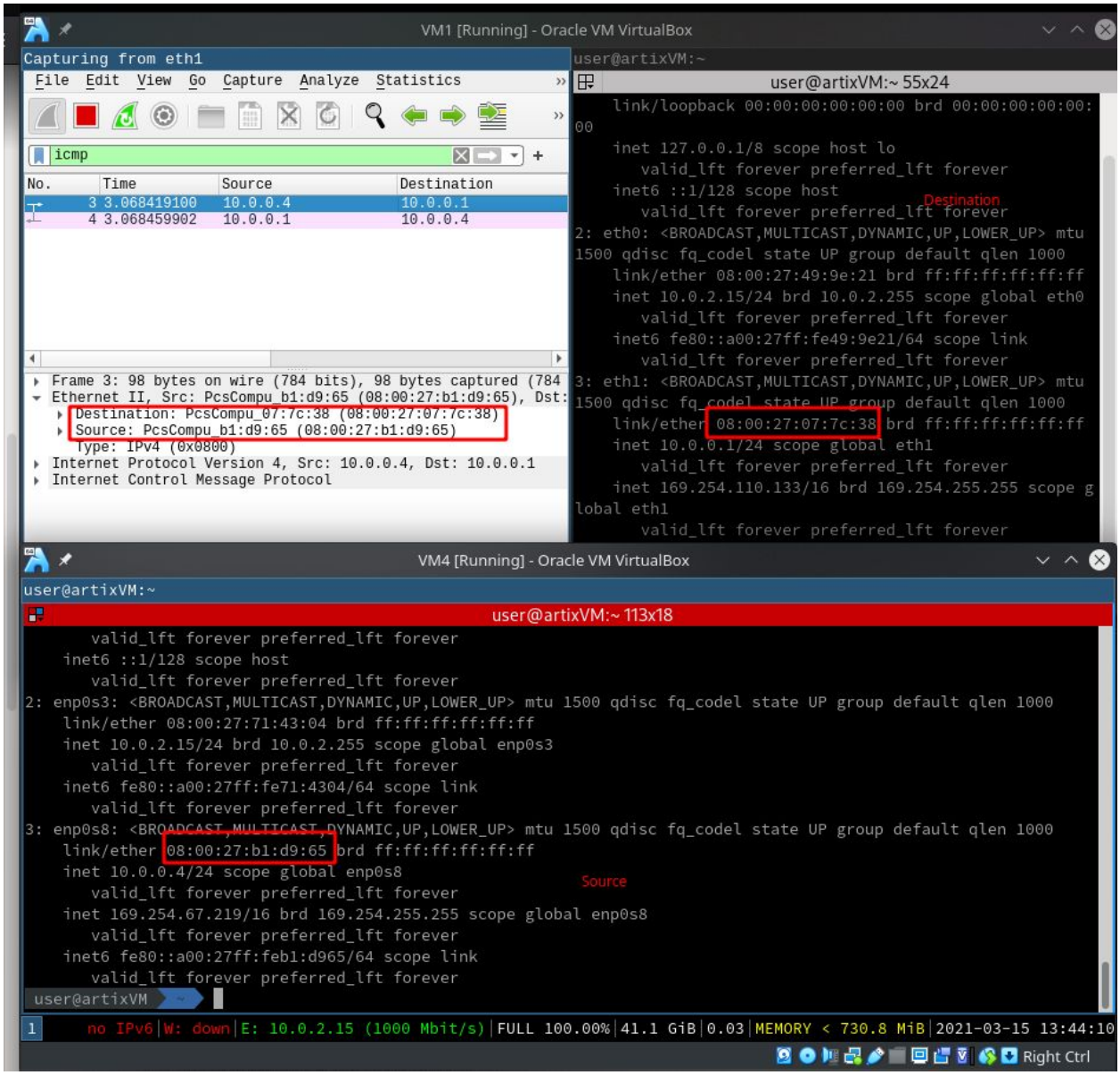



When accessing server on VM1 from VM4:




We can see that no IP4 packets are received from VM3 on VM1.

2. Initially:




Using commands:

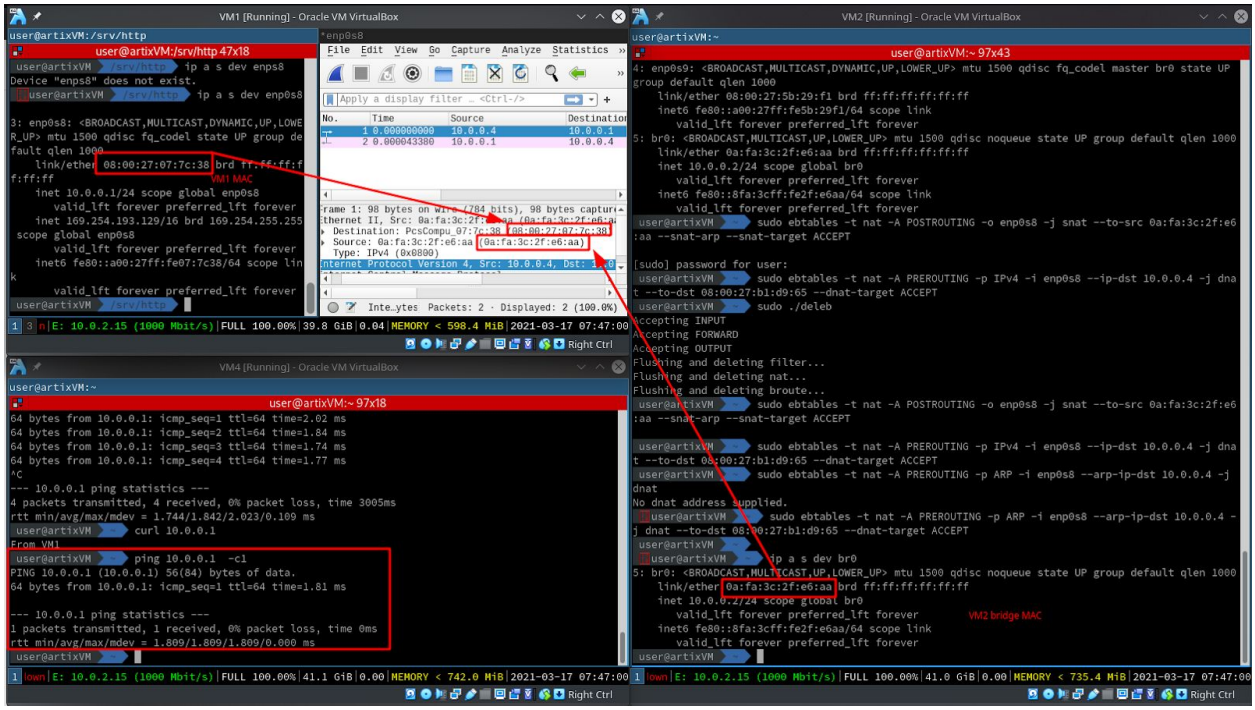
For source nat:

```
sudo ebtables -t nat -A POSTROUTING -o enp0s8 -j snat --to-src
0a:fa:3c:2f:e6:aa --snat-arp --snat-target ACCEPT
```

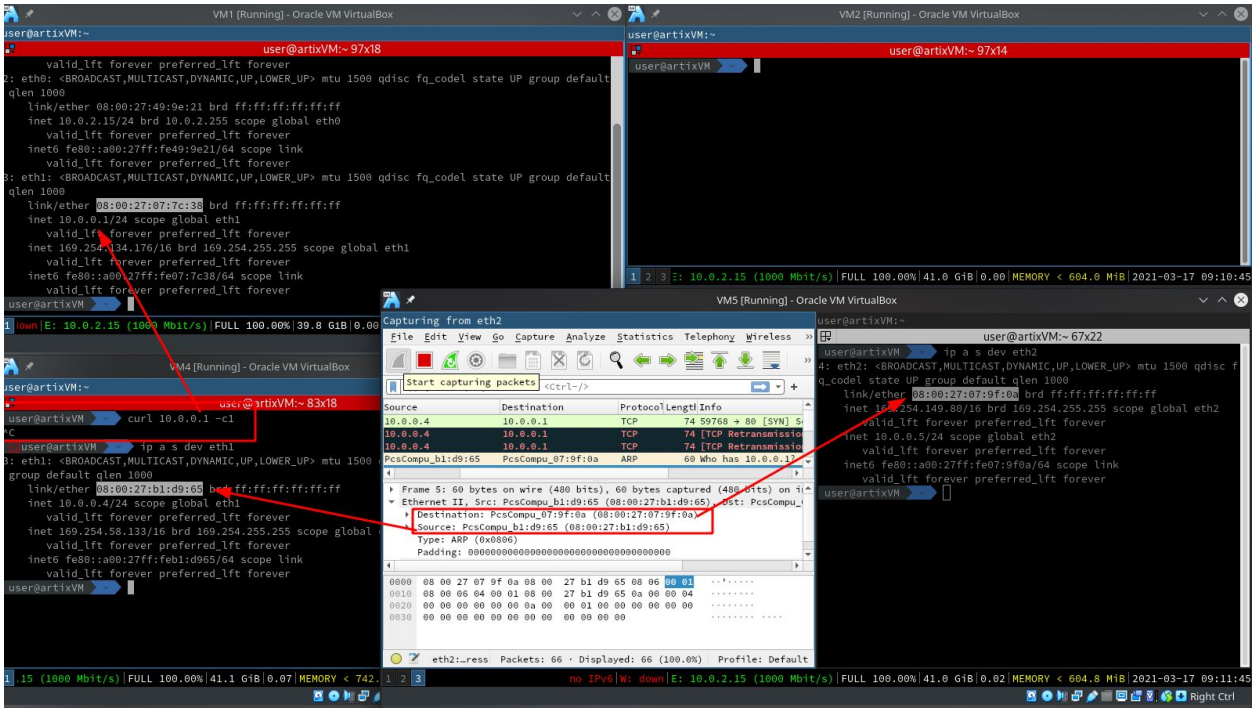
For destination nat, mapping between IP address and MAC for VM4:

```
sudo ebtables -t nat -A PREROUTING -p IPv4 -i enp0s8 --ip-dst
10.0.0.4 -j dnat --to-dst 08:00:27:b1:d9:65 --dnat-target ACCEPT
sudo ebtables -t nat -A PREROUTING -p ARP -i enp0s8 --arp-ip-dst
10.0.0.4 -j dnat --to-dst 08:00:27:b1:d9:65 --dnat-target ACCEPT
```

Similarly this can be done for VM3 and VM5.




3. Since the source and nat-ed output should be on the different interfaces on a bridge, the following changes were made:
   1) Another interface was added to VM2(eth3) and VM5(eth2: 10.0.0.5/24)
   2) Added eth3 to existing br0
   Command to do MAC address NAT, all the packets of destination VM1(08:00:27:07:7c:38) will be forwarded to VM5(08:00:27:07:9f:0a) but VM5 will drop the packet since it's IP address is different from destination IP address in the packet :
   `sudo ebtables -t nat -A PREROUTING -d 08:00:27:07:7c:38 -i eth2 -j dnat --to-destination 08:00:27:07:9f:0a`




4. This thing cannot be attained by IP tables because bridge network work at Layer 2 and packet is forwarded before accessing through IP table hooks which work at Layer 3.

# Part 2

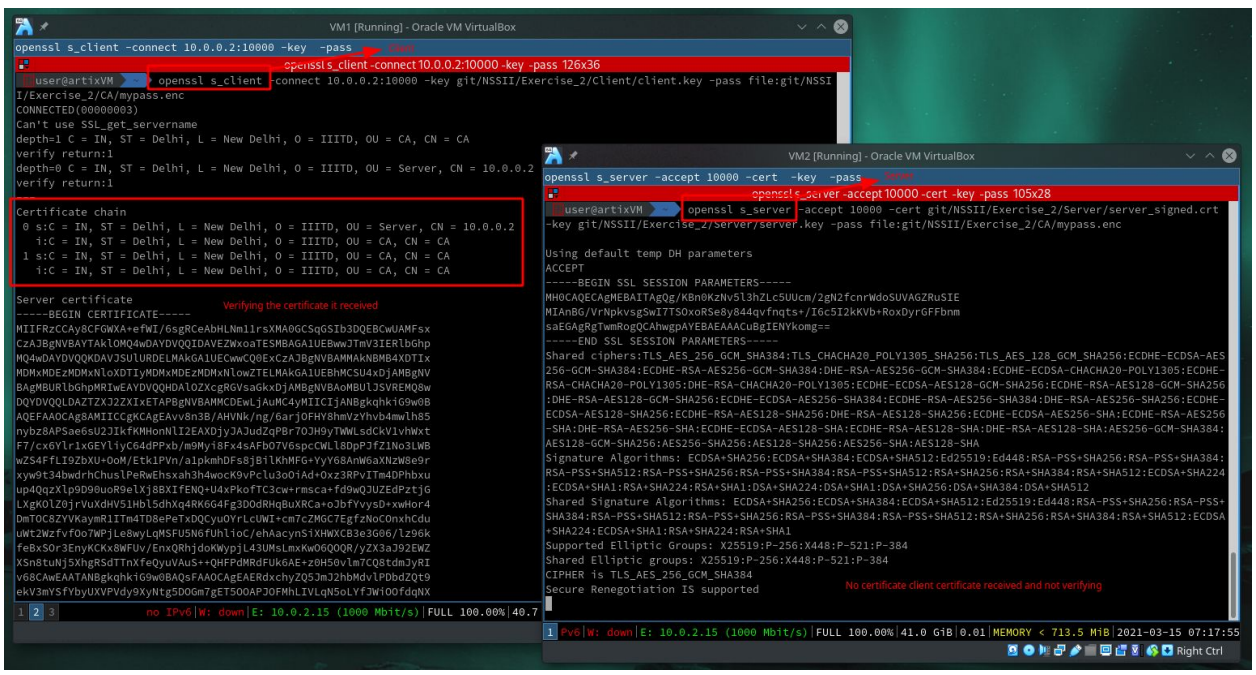
1. On the host machine:
    a. For CA,
        i. `openssl genrsa -out ca.key 4096.` Creates a new private key for CA.
        ii. `openssl req -new -x509 -days 365 -key ca.key -out ca.cert.pem.` Creates a self signed certificate in x509 format for the CA.
    b. For Server
        i. `openssl genrsa -out server.key 4096.` Creates a new private key for the server.
        ii. `openssl req -new -key server.key -out server.csr.` Creates a new certificate signing request for the server.
        iii. `openssl x509 -req -days 365 -in server.csr -CA ../CA/ca.cert.pem -CAkey ../CA/ca.key -CAcreateserial -out server_signed.crt.` Creates a certificate for the server signed by the CA.
    c. For Client
        i. `openssl genrsa -out client.key 4096.` Creates a new private key for the client.

ii. `openssl req -new -key client.key -out client.csr.` Creates a new certificate signing request for the client.

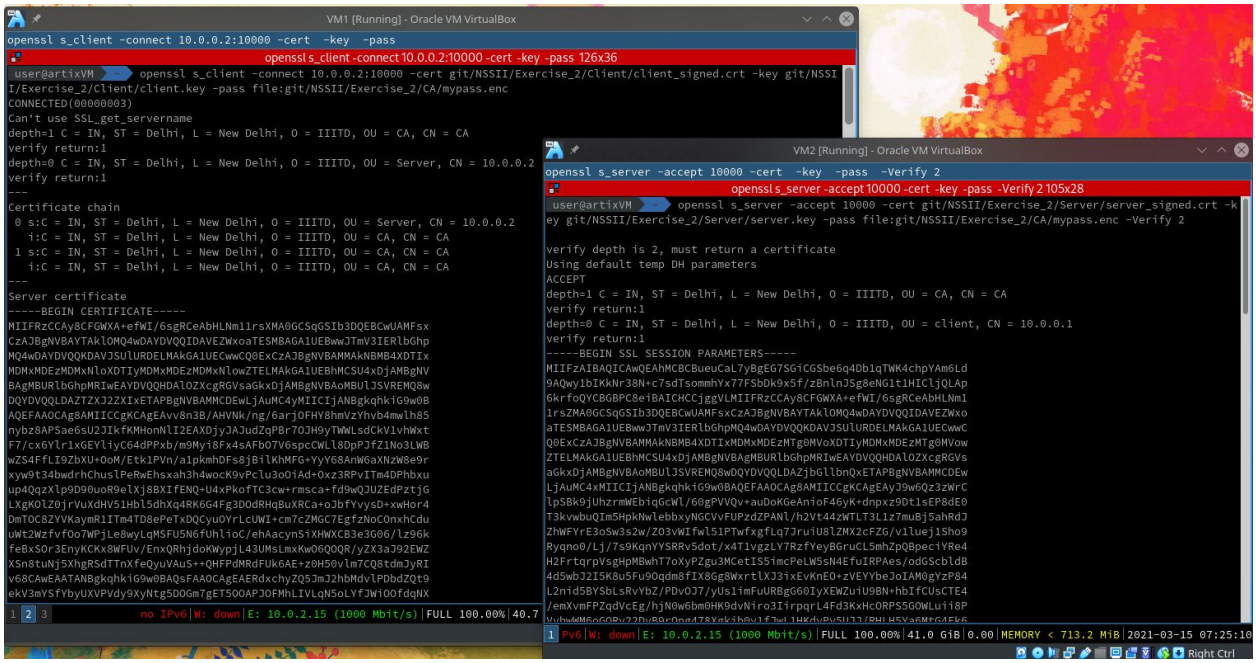
iii. `openssl x509 -req -days 365 -in client.csr -CA ../CA/ca.cert.pem -CAkey ../CA/ca.key -CAcreateserial -out client_signed.crt.` Creates a certificate for the server signed by the CA.

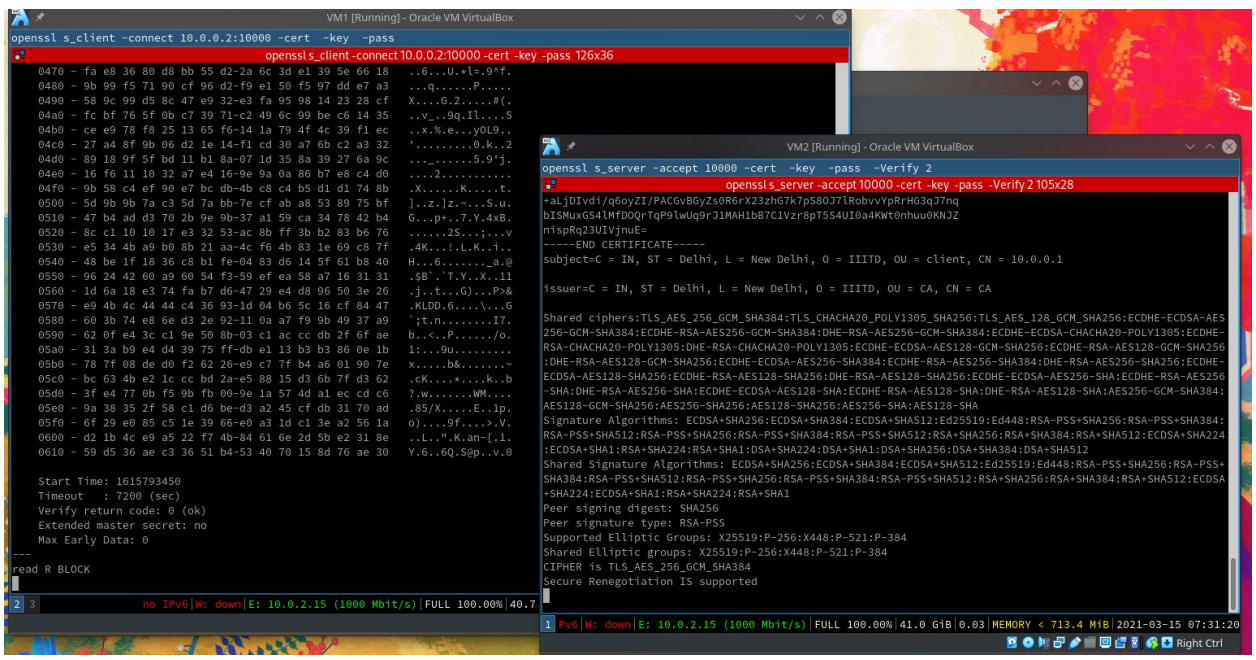
Install the appropriate certificates on the Server and Client using `trust anchor --store` *`myCA`*`.crt`

2. Starting the server on port 10000: `openssl s_server -accept 10000 -cert git/NSSII/Exercise_2/Server/server_signed.crt -key git/NSSII/Exercise_2/Server/server.key`
3. Connecting the server from the client without verifying the client. `openssl s_client -connect 10.0.0.2:10000 -key git/NSSII/Exercise_2/Client/client.key -pass file:git/NSSII/Exercise_2/CA/mypass.enc`

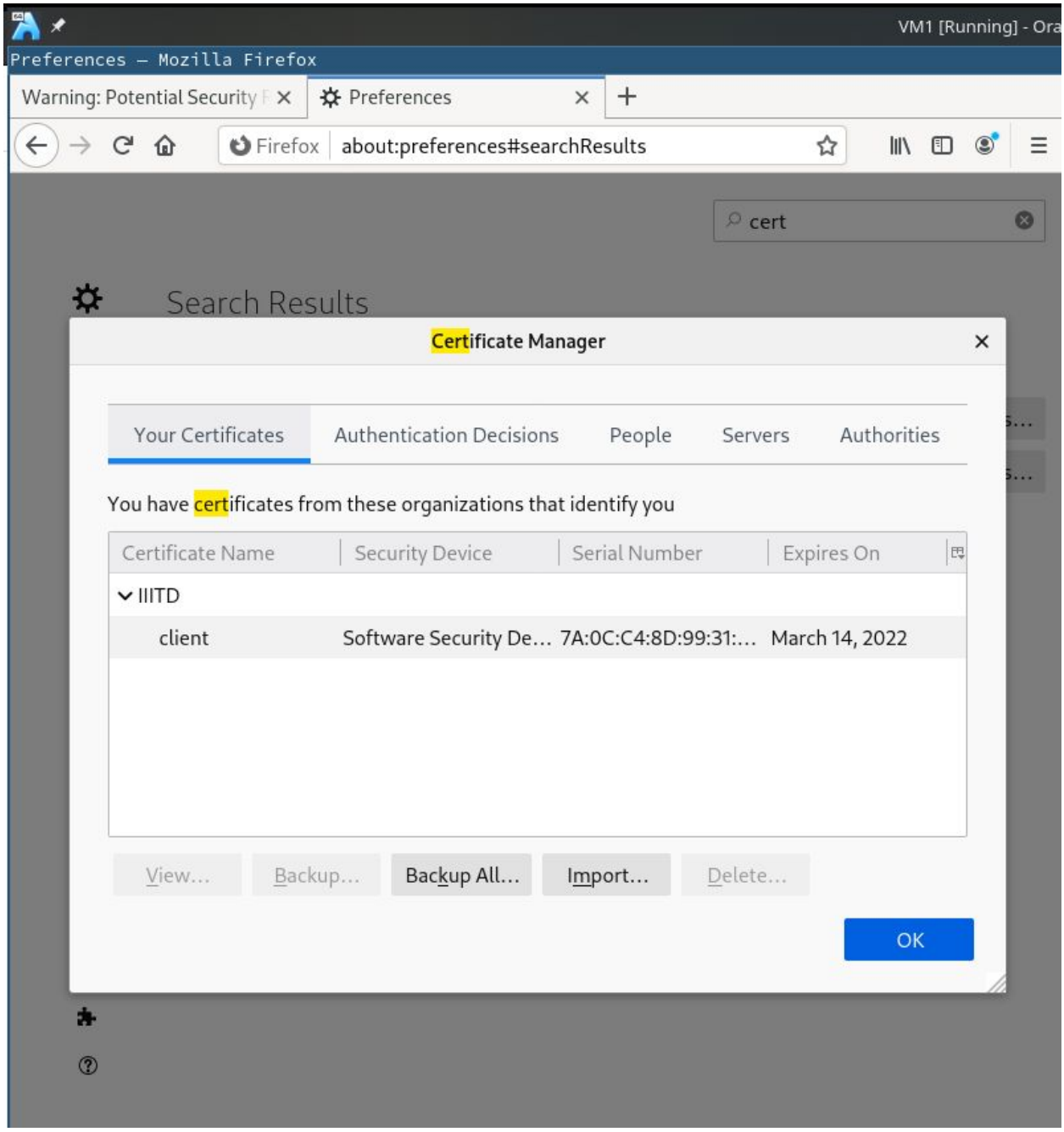


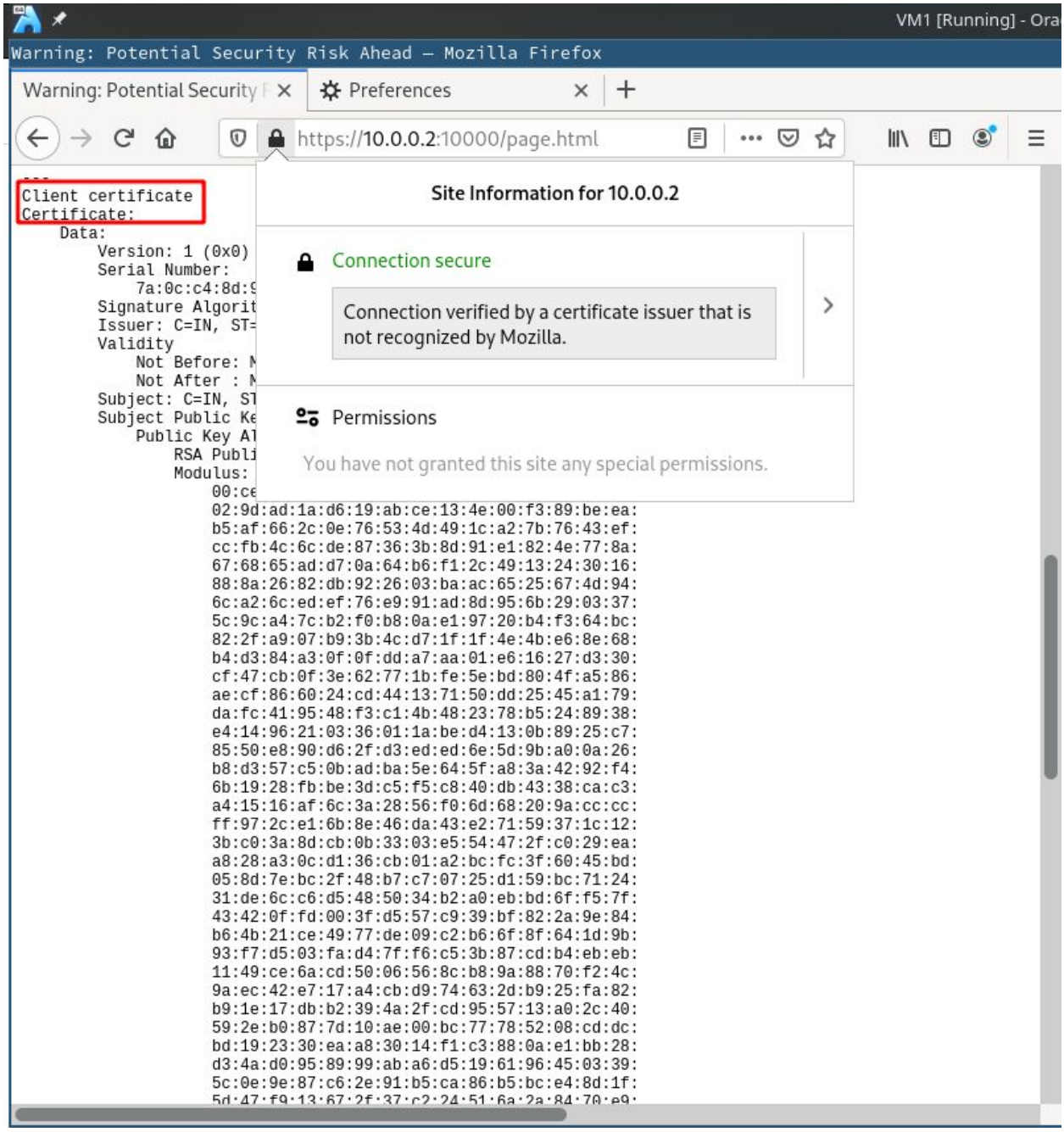

4. On the server side: `openssl s_server -accept 10000 -cert git/NSSII/Exercise_2/Server/server_signed.crt -key git/NSSII/Exercise_2/Server/server.key -pass file:git/NSSII/Exercise_2/CA/mypass.enc -Verify 2`
   On the client side: `openssl s_client -connect 10.0.0.2:10000 -cert git/NSSII/Exercise_2/Client/client_signed.crt -key git/NSSII/Exercise_2/Client/client.key -pass file:git/NSSII/Exercise_2/CA/mypass.enc`
   The -Verify option on the server makes it mandatory for the client to submit a certificate, and then server verifies it.




5. Server Command: `openssl s_server -accept 10000 -cert git/NSSII/Exercise_2/Server/server_signed.crt -key git/NSSII/Exercise_2/Server/server.key -pass file:git/NSSII/Exercise_2/CA/mypass.enc -www -Verify 2`
   Use ‘`openssl pkcs12 -export -in server.crt -inkey server.key -out server.p12`’ to make a file containing certificate and the private key to import into firefox. Verify upto depth 2 but since the depth of our certificate is 1, it will work.

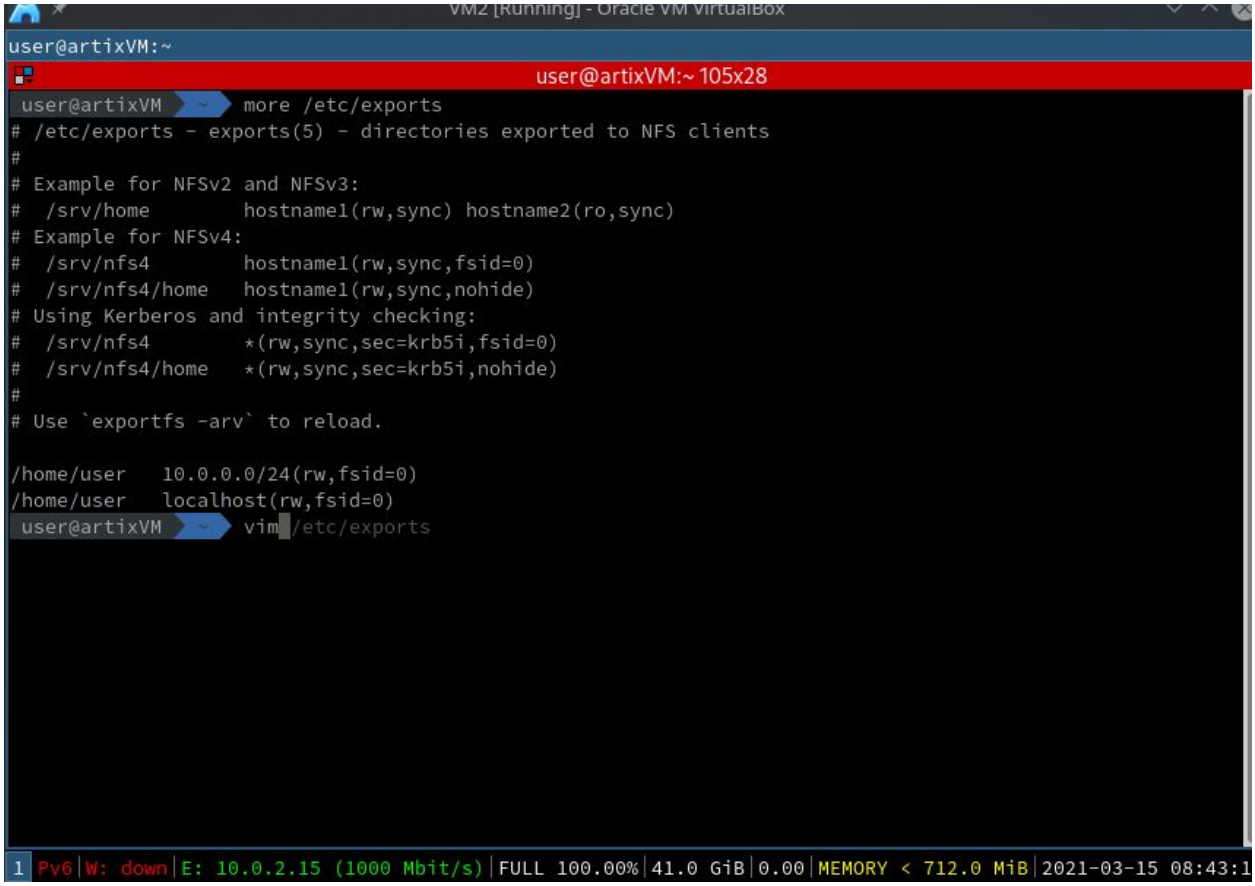

VM1 [Running] - Ora
Preferences — Mozilla Firefox
Warning: Potential Security
Preferences
Firefox
about:preferences#searchResults
cert
Search Results
Certificate Manager
Your Certificates
Authentication Decisions
People
Servers
Authorities
You have certificates from these organizations that identify you
Certificate Name
Security Device
Serial Number
Expires On
IIITD
client
Software Security De…
7A:0C:C4:8D:99:31:…
March 14, 2022
View…
Backup…
Backup All…
Import…
Delete…
OK



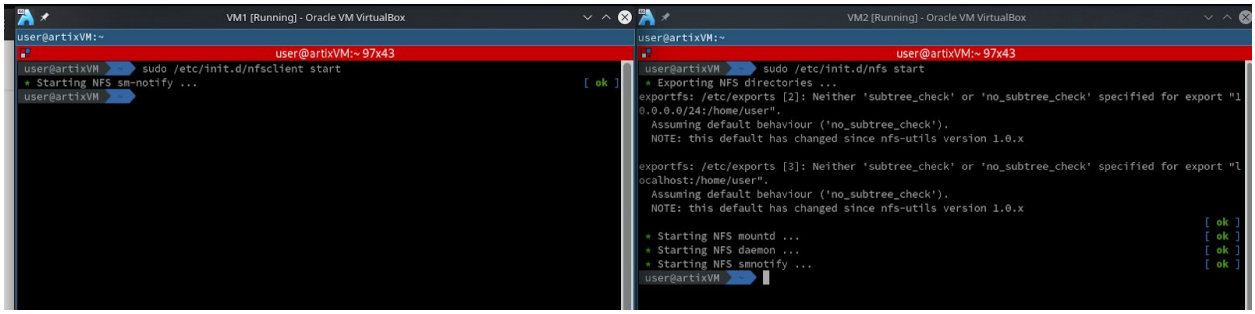

# Part 3

1. Export home folder in /etc/exports, make fsid=0 to make /home/user as fake root of the export. The directory is also exported to localhost to be compatible with stunnel in next

steps.




2. Export this file using `exportfs -arv`.
3. Run the nfs server using `/etc/init.d/nfs start`. On the client use `/etc/init.d/nfsclient` before mounting.



4. On client use `sudo mount 10.0.0.2:/home/user /mnt` to mount the directory.

```
user@artixVM ~ sudo /etc/init.d/nfsclient start
 * Starting NFS sm-notify ...                                          [ ok ]
user@artixVM ~ sudo mount 10.0.0.2:/home/user /mnt
user@artixVM ~ ls
Downloads  Exercise2_client  keygen    server_stunnel.cert  stunnel.pem
Exercis_2  git               myCA.key  start-network        testscript.sh
user@artixVM ~ cd /mnt
user@artixVM /mnt ls
Exercise2  half-open.pcapng    nossl.pcapng         start-network        stunnel.pem
git        normal-http.pcapng  server_stunnel.cert  start-network-local
user@artixVM /mnt
```

5. Configure stunnel on server(/etc/stunnel/stunnel.conf) as follows

```
 * Starting NFS smnotify ...
user@artixVM ~ cat /etc/stunnel/stunnel.conf
; BOM composed of non printable characters. It is here, before the semicolon!
setuid = stunnel
setgid = stunnel
output = /tmp/nfs.log
[nfs]
debug       = 3
verify      = 0
accept      = 12345
connect     = localhost:2049
cert        = /etc/stunnel/stunnel.pem
key         = /etc/stunnel/stunnel.pem
setuid      = stunnel
setgid      = stunnel
user@artixVM ~
```

stunnel.pem is a file containing a key and a self signed certificate, it acts as a pre shared secret for both client and server. It is generated using command `openssl req -new -x509 -days 365 -nodes -out stunnel.pem -keyout stunnel.pem`. A screenshot from https://www.stunnel.org/howto.html to know the parameters.

-days 365
: make this key valid for 1 year, after which it is not to be used any more

-new
: Generate a new key

-x509
: Generate an X509 certificate (self sign)

-nodes
: Do not put a password on this key.

-config `stunnel.cnf`
: the OpenSSL configuration file to use

-out `stunnel.pem`
: where to put the SSL certificate

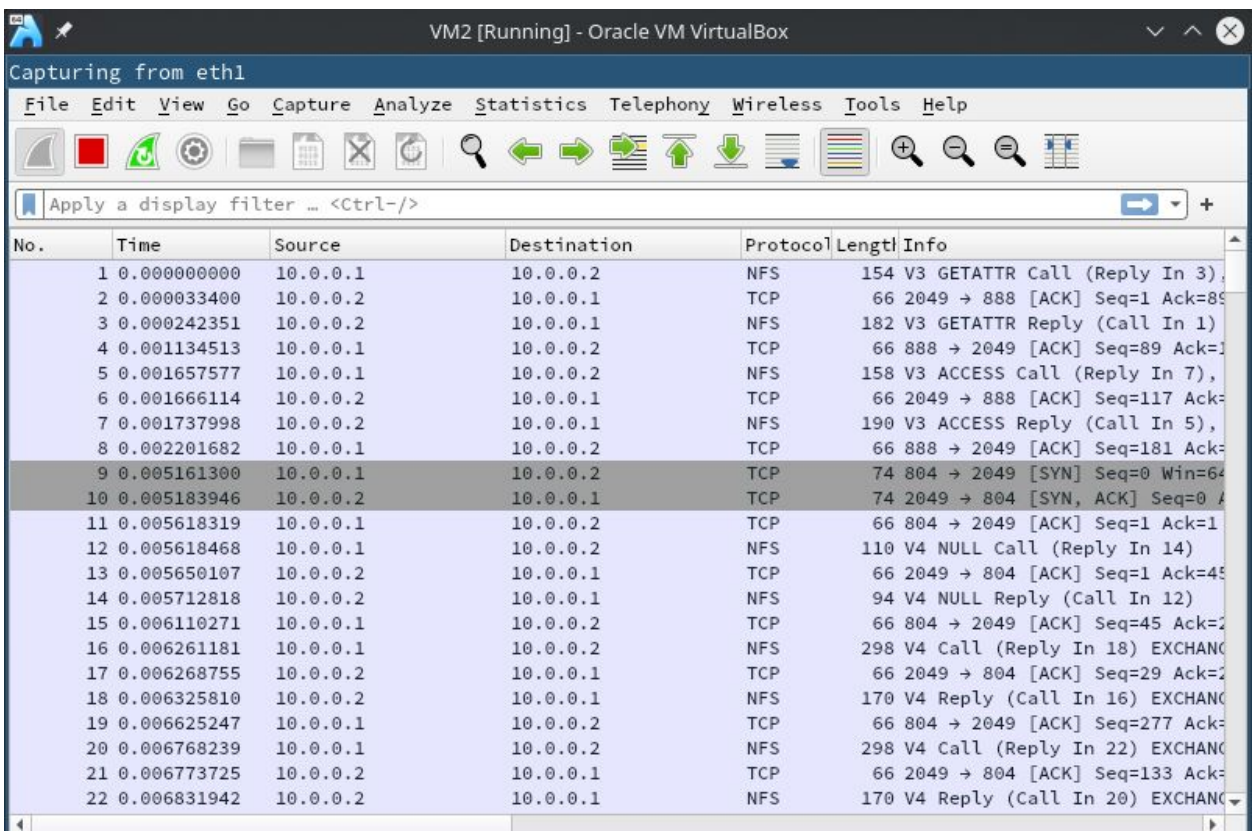
-keyout `stunnel.pem`
: put the key in this file

6. Configure stunnel on client(/etc/stunnel/stunnel.conf) as follows

```
user@artixVM  /mnt  cat /etc/stunnel/stunnel.conf
; BOM composed of non printable characters. It is here, before the semicolon!
setuid = stunnel
setgid = stunnel
output = /tmp/nfs.log
[nfs]
client     = yes
accept     = localhost:20490
connect    = 10.0.0.2:12345
debug      = 3
cert       = /etc/stunnel/stunnel.pem
key        = /etc/stunnel/stunnel.pem
setuid     = stunnel
setgid     = stunnel
```

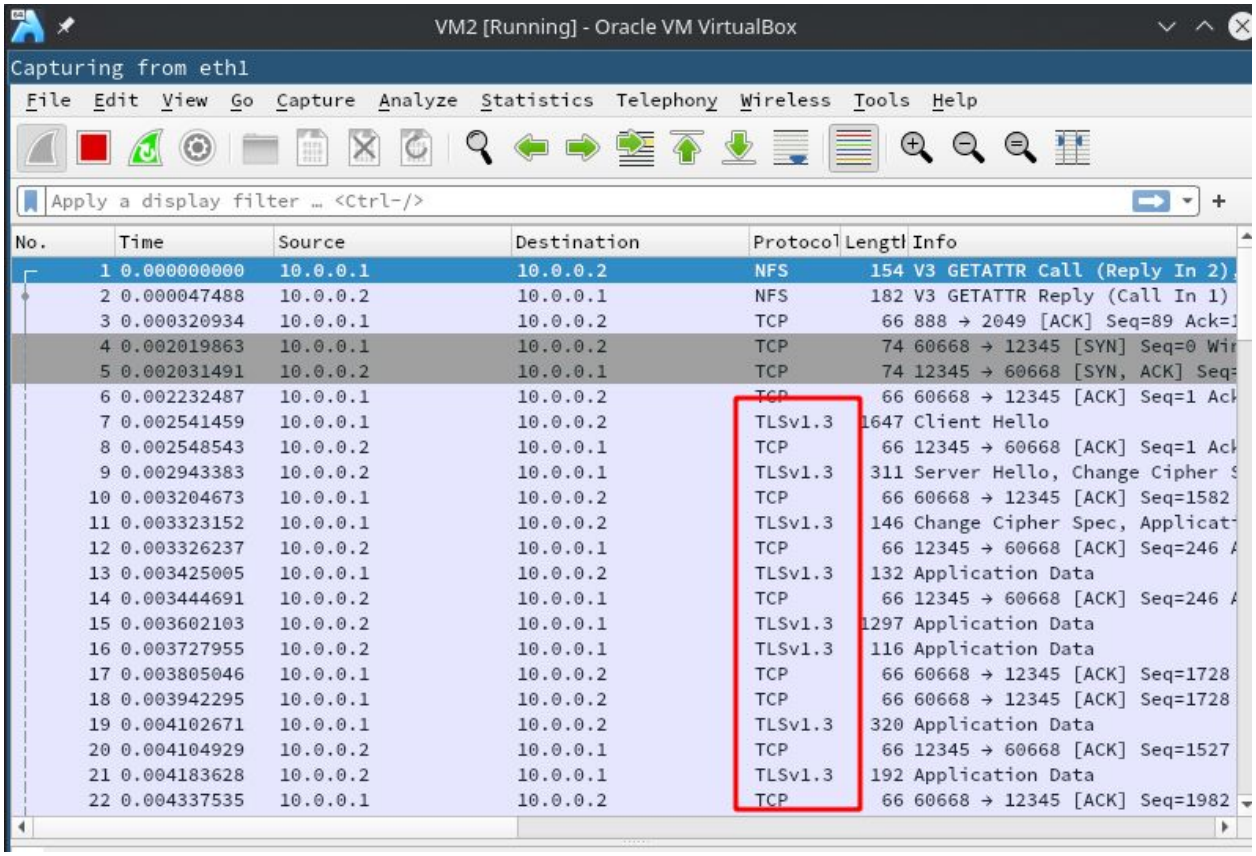
Client will interact with its port 20490, stunnel will read the data from that port and make a secure connection with port 12345 on server. Stunnel on server will decrypt the data and send it to port 2049(which is used by nfs server). This connection will work both way round.

7. Mount the encrypted partition using: `sudo mount -o port=20490 localhost:/home/user /mnt`
8. Unsecure data packet capture will look as follows




| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1 | 0.000000000 | 10.0.0.1 | 10.0.0.2 | NFS | 154 | V3 GETATTR Call (Reply In 3), |
| 2 | 0.000033400 | 10.0.0.2 | 10.0.0.1 | TCP | 66 | 2049 → 888 [ACK] Seq=1 Ack=89 |
| 3 | 0.000242351 | 10.0.0.2 | 10.0.0.1 | NFS | 182 | V3 GETATTR Reply (Call In 1) |
| 4 | 0.001134513 | 10.0.0.1 | 10.0.0.2 | TCP | 66 | 888 → 2049 [ACK] Seq=89 Ack=1 |
| 5 | 0.001657577 | 10.0.0.1 | 10.0.0.2 | NFS | 158 | V3 ACCESS Call (Reply In 7), |
| 6 | 0.001666114 | 10.0.0.2 | 10.0.0.1 | TCP | 66 | 2049 → 888 [ACK] Seq=117 Ack= |
| 7 | 0.001737998 | 10.0.0.2 | 10.0.0.1 | NFS | 190 | V3 ACCESS Reply (Call In 5), |
| 8 | 0.002201682 | 10.0.0.1 | 10.0.0.2 | TCP | 66 | 888 → 2049 [ACK] Seq=181 Ack= |
| 9 | 0.005161300 | 10.0.0.1 | 10.0.0.2 | TCP | 74 | 804 → 2049 [SYN] Seq=0 Win=64 |
| 10 | 0.005183946 | 10.0.0.2 | 10.0.0.1 | TCP | 74 | 2049 → 804 [SYN, ACK] Seq=0 A |
| 11 | 0.005618319 | 10.0.0.1 | 10.0.0.2 | TCP | 66 | 804 → 2049 [ACK] Seq=1 Ack=1 |
| 12 | 0.005618468 | 10.0.0.1 | 10.0.0.2 | NFS | 110 | V4 NULL Call (Reply In 14) |
| 13 | 0.005650107 | 10.0.0.2 | 10.0.0.1 | TCP | 66 | 2049 → 804 [ACK] Seq=1 Ack=45 |
| 14 | 0.005712818 | 10.0.0.2 | 10.0.0.1 | NFS | 94 | V4 NULL Reply (Call In 12) |
| 15 | 0.006110271 | 10.0.0.1 | 10.0.0.2 | TCP | 66 | 804 → 2049 [ACK] Seq=45 Ack=2 |
| 16 | 0.006261181 | 10.0.0.1 | 10.0.0.2 | NFS | 298 | V4 Call (Reply In 18) EXCHANG |
| 17 | 0.006268755 | 10.0.0.2 | 10.0.0.1 | TCP | 66 | 2049 → 804 [ACK] Seq=29 Ack=2 |
| 18 | 0.006325810 | 10.0.0.2 | 10.0.0.1 | NFS | 170 | V4 Reply (Call In 16) EXCHANG |
| 19 | 0.006625247 | 10.0.0.1 | 10.0.0.2 | TCP | 66 | 804 → 2049 [ACK] Seq=277 Ack= |
| 20 | 0.006768239 | 10.0.0.1 | 10.0.0.2 | NFS | 298 | V4 Call (Reply In 22) EXCHANG |
| 21 | 0.006773725 | 10.0.0.2 | 10.0.0.1 | TCP | 66 | 2049 → 804 [ACK] Seq=133 Ack= |
| 22 | 0.006831942 | 10.0.0.2 | 10.0.0.1 | NFS | 170 | V4 Reply (Call In 20) EXCHANG |

9. TLS secure data packets will look as follows




| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1 | 0.000000000 | 10.0.0.1 | 10.0.0.2 | NFS | 154 | V3 GETATTR Call (Reply In 2) |
| 2 | 0.000047488 | 10.0.0.2 | 10.0.0.1 | NFS | 182 | V3 GETATTR Reply (Call In 1) |
| 3 | 0.000320934 | 10.0.0.1 | 10.0.0.2 | TCP | 66 | 888 → 2049 [ACK] Seq=89 Ack=1 |
| 4 | 0.002019863 | 10.0.0.1 | 10.0.0.2 | TCP | 74 | 60668 → 12345 [SYN] Seq=0 Wi |
| 5 | 0.002031491 | 10.0.0.2 | 10.0.0.1 | TCP | 74 | 12345 → 60668 [SYN, ACK] Seq= |
| 6 | 0.002232487 | 10.0.0.1 | 10.0.0.2 | TCP | 66 | 60668 → 12345 [ACK] Seq=1 Ack |
| 7 | 0.002541459 | 10.0.0.1 | 10.0.0.2 | TLSv1.3 | 1647 | Client Hello |
| 8 | 0.002548543 | 10.0.0.2 | 10.0.0.1 | TCP | 66 | 12345 → 60668 [ACK] Seq=1 Ack |
| 9 | 0.002943383 | 10.0.0.2 | 10.0.0.1 | TLSv1.3 | 311 | Server Hello, Change Cipher S |
| 10 | 0.003204673 | 10.0.0.1 | 10.0.0.2 | TCP | 66 | 60668 → 12345 [ACK] Seq=1582 |
| 11 | 0.003323152 | 10.0.0.1 | 10.0.0.2 | TLSv1.3 | 146 | Change Cipher Spec, Applicat |
| 12 | 0.003326237 | 10.0.0.2 | 10.0.0.1 | TCP | 66 | 12345 → 60668 [ACK] Seq=246 A |
| 13 | 0.003425005 | 10.0.0.1 | 10.0.0.2 | TLSv1.3 | 132 | Application Data |
| 14 | 0.003444691 | 10.0.0.2 | 10.0.0.1 | TCP | 66 | 12345 → 60668 [ACK] Seq=246 A |
| 15 | 0.003602103 | 10.0.0.2 | 10.0.0.1 | TLSv1.3 | 1297 | Application Data |
| 16 | 0.003727955 | 10.0.0.2 | 10.0.0.1 | TLSv1.3 | 116 | Application Data |
| 17 | 0.003805046 | 10.0.0.1 | 10.0.0.2 | TCP | 66 | 60668 → 12345 [ACK] Seq=1728 |
| 18 | 0.003942295 | 10.0.0.1 | 10.0.0.2 | TCP | 66 | 60668 → 12345 [ACK] Seq=1728 |
| 19 | 0.004102671 | 10.0.0.1 | 10.0.0.2 | TLSv1.3 | 320 | Application Data |
| 20 | 0.004104929 | 10.0.0.2 | 10.0.0.1 | TCP | 66 | 12345 → 60668 [ACK] Seq=1527 |
| 21 | 0.004183628 | 10.0.0.2 | 10.0.0.1 | TLSv1.3 | 192 | Application Data |
| 22 | 0.004337535 | 10.0.0.1 | 10.0.0.2 | TCP | 66 | 60668 → 12345 [ACK] Seq=1982 |

# References

https://www.thegeekstuff.com/2017/06/brctl-bridge/ brctl command
http://www.microhowto.info/howto/bridge_traffic_between_two_or_more_ethernet_interfaces_on_linux.html using bridge interface
https://ebtables.netfilter.org/examples/basic.html#ex_nat Destination nating

https://wiki.archlinux.org/index.php/OpenSSL openssl
https://deliciousbrains.com/ssl-certificate-authority-for-local-https-development/ set up CA

https://www.golinuxcloud.com/create-certificate-authority-root-ca-linux/
https://wiki.archlinux.org/index.php/User:Grawity/Adding_a_trusted_CA_certificate#System-wide_%E2%80%93_Arch,_Fedora_(p11-kit) adding certificate

https://security.stackexchange.com/questions/108508/how-do-i-produce-a-ca-signed-public-key
https://www.golinuxcloud.com/openssl-create-client-server-certificate/ signing public keys certificates with CA key

https://www.openssl.org/docs/man1.0.2/man1/openssl-s_server.html
https://www.openssl.org/docs/man1.0.2/man1/openssl-s_client.html openssl man page

https://wiki.archlinux.org/index.php/NFS using nfs server
https://www.linuxjournal.com/content/encrypting-nfsv4-stunnel-tls nfs

https://www.stunnel.org/howto.html stunnel config