An OpenVPN server using Ubuntu with containerization options using Docker

Unix Systems Administration and Programming (Linux) – RMIT University

2024, 2nd semester – Hanoi Campus, Vietnam

Ramon Casas i Luque

Contents

1. Introduction

Virtual Private Networks (VPNs) are usually used either to access local network resources from companies or institutions (in more professional settings) or to bypass regional restrictions, as well as in order to hide public IPs for various reasons. In this assignment we will implement an OpenVPN server using a Linux machine. We will demonstrate how clients can connect to this server and tunnel their traffic through it in order to access resources available from the server's IP.

To describe the project, we will first explain the installation of the server, including its requirements and technologies used. Then we will describe our implementation together with a use-case. Afterwards, we will provide proof of testing to show that our configuration is successfully working. In the last section we will provide some user documentation to answer the questions a user may have when using the server.

2. Plan for the installation

2.1. Technology choices

To set up our VPN server we used Ubuntu, which is one of the most popular Linux distributions. It receives regular upgrades (every 6 months) and a wide long-term support for its different versions[1]. We have been using Ubuntu throughout the course and thought it would be a good option to test our capabilities in building a server using it. Apart from that, we also used OpenVPN, which is a free software solution to build VPN servers. It has a lot of configuration options, and best of all, there exist many materials and documentation, as well as a wide community[2], to provide solutions to problems we may face during our installation.

Besides the two resources above, we also implemented a version of our OpenVPN using Docker. In this case, we containerize the OpenVPN server so that the user can directly run the server by just running an instance of a docker image. As we will explain later, this option will still require some configuration by the user which we have considered not convenient to containerize.

2.2. Requirements specification

Hardware requirements[3]

- 2 GHz dual-core processor
- 4096 MiB of RAM
- 25GB of hard-drive space

[1] https://ubuntu.com/about
[2] https://community.openvpn.net/openvpn/wiki
[3] We decided to establish the same as Ubuntu: https://help.ubuntu.com/community/Installation/SystemRequirements

- Network interface with specific IP address for the server

Software requirements

- OS: Ubuntu 24.04 or later
- Uncomplicated Firewall (ufw)
- Easy-rsa
- OpenVPN (latest stable version)
- Docker*

*The Docker version of our project does not require to have OpenVPN installed. We offer this as an option, but the server works both with and without Docker.

2.3. Installation plan

1. Preliminaries

In an ubuntu machine:

- Update system packages: sudo apt update and sudo apt upgrade.
- Install ufw and easy-rsa.

2. OpenVPN installation and configuration

- Install OpenVPN using the ubuntu package manager.
- Configure OpenVPN with the appropriate configuration files for server and clients.
- Generate certificates using easy-rsa.
- Sign certificates and distribute them securely among clients.

3. Firewall configuration

- Enable and configure ufw to allow OpenVPN traffic.

4. Containerize the server configuration

- Use Docker to containerize the server so that it is easier for users to take advantage of the resource.

5. Test and verify

- Connectivity: ensure clients can connect to the server.
- Routing: ensure that clients can access the desired resources.

3. Description of the implementation (25%) (approx <1000 words)

3.1. Setting up the OpenVPN server on the Ubuntu VM

3.1.1. Installing OpenVPN and Easy-RSA

The installation of **OpenVPN** is straightforward. In Ubuntu, we follow the same steps as with many other applications:

```
sudo apt update
sudo apt upgrade
sudo apt install openvpn
```

To install easy-rsa, rather than doing it directly with apt-get, we clone the github repository of OpenVPN that contains easy-rsa 3. We follow the indications of the official documentation[4]: “On *NIX platforms you should look into using easy-rsa 3 instead”.

3.1.2. Generating keys and certificates

The next steps are to generate keys and certificates for our VPN. This will allow us to establish secure connections thanks to asymmetric cryptography[5]. The quickstart guide of the official OpenVPN repository[6] is a good resource to understand how to do so. Furthermore, chapter 4 of Keijser (2017) and chapter 3 by Crist and Keijser (2015) are more extensive practical (and with some theoretical context also) useful explanations on the topic. In the user documentation we provide the necessary steps to generate these files.

3.1.3. Server and client configuration files

We need a configuration file for the server and a configuration file for each client in order to use our VPN. OpenVPN provides some sample configuration files that can be found after its installation in:

```
/usr/share/doc/openvpn/examples/sample-config-files
```

We used these sample files to write ours. Here are some options of how they may look with a very basic configuration (and after removing comments):

server.conf file	client_name.conf file
`port 1194`	`client`
`proto udp`	`dev tun`
`dev tun`	`proto udp`
`ca ca.crt`	`remote server-ip 1194`
`cert server.crt`	`resolv-retry infinite`
`key server.key`	`nobind`

[4] https://openvpn.net/community-resources/how-to

[5] This is a good and very short introduction to Public Key Infrastructure (PKI): https://youtu.be/0ctat6RBrFo?si=7YGeXfTOJvpT8e__

[6] https://github.com/OpenVPN/easy-rsa/blob/master/README.quickstart.md

```
dh dh.pem
server 10.8.0.0 255.255.255.0
ifconfig-pool-persist ipp.txt
push "redirect-gateway def1 bypass-dhcp"
push "dhcp-option DNS 8.8.8.8"
push "dhcp-option DNS 8.8.4.4"
keepalive 10 120
cipher AES-256-CBC
persist-key
persist-tun
verb 3
```

```
persist-key
persist-tun
ca ca.crt
cert client.crt
key client.key
cipher AES-256-CBC
auth SHA256
key-direction 1
verb 3
```

Note that the files (ca.crt, server.crt, etc.) should be in the same folder as the .conf file for this to work. If not, we should write the complete path to them.

3.1.4. Network configuration

For our server to work, it is also very important to configure network settings correctly. These are the fundamental parts of this configuration:

- Firewall using UFW: control incoming and outoing traffic on server. Use commands like `sudo ufw enable`, `sudo ufw status`...
- Allow OpenVPN traffic: open port 1194 for OpenVPN connections (`sudo ufw allow 1194/udp`).
- Packet forwarding: enable it with `sudo nano /etc/sysctl.conf` and `net.ipv4.ip_forward=1`
- Network Address Translation (NAT): configure it to allow clients to access internet through the server (`sudo iptables -t nat -A POSTROUTING -o <interface> -j MASQUERADE`)
- Port forwarding: configure the router to allow incoming VPN connections. Not done in ubuntu itself.

The textbooks on OpenVPN do not provide much help with this basic networking settings, so it is necessary to look for other resources[7].

3.1.5. Enabling and starting OpenVPN

Once the previous configuration is done, it is easy to start OpenVPN. On the server side:

```
sudo openvpn server.conf
```

And on the client side:

```
sudo openvpn client_name.conf
```

3.1.6. Using Docker

There exist some images such as https://github.com/kylemanna/docker-openvpn, which include also the easy-rsa tool. However, we decided to keep this separate

[7] https://www.cyberciti.biz/faq/how-to-configure-firewall-with-ufw-on-ubuntu-20-04-lts/

because it may be more convenient in order to handle clients and signing certificates in a separate machine than the one that is running the server. To do so, we use a Docker volume[8] to store this data outside the container.

Moreover, we observed that it is not convenient to containerize network configuration, such as firewall management and routing. This would make our image less portable and usable across different machines.

Here's the Dockerfile which we used to build an image:

```
FROM ubuntu:latest

#install openvpn

RUN apt-get update && apt-get install -y openvpn && rm -rf
/var/lib/apt/lists/*

#copy server configuration file

COPY server.conf /etc/openvpn/

#mount the pki directory

VOLUME /etc/openvpn/pki

#expose the OpenVPN port

EXPOSE 1194/udp

#start

CMD [“openvpn”, “--config”, “/etc/openvpn/server.conf”]
```

To run the image we use:

```
sudo docker run -d --name my-vpn-container --privileged --network
host -p 1194:1194/udp -v /path/to/folder/pki/:/etc/openvpn/pki
myopenvpn
```

3.2. Configuring a client in different platforms

There are basically three main parts to configure a client:

- Install OpenVPN client software.
- Import configuration files.
- Connect to the VPN.

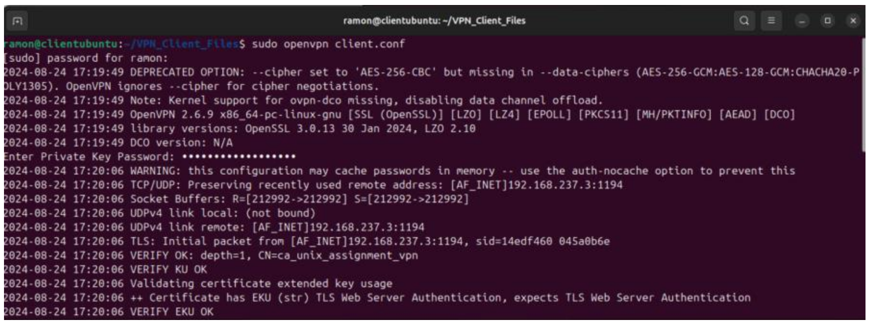
The first step to configuring a client in different platforms is to install the OpenVPN client software. This is a straightforward step that can be easily done. What is more tricky, however, is to achieve connectivity between the different devices in the local network through virtual box, its host only adapter and other virtual adapters.

Linux client

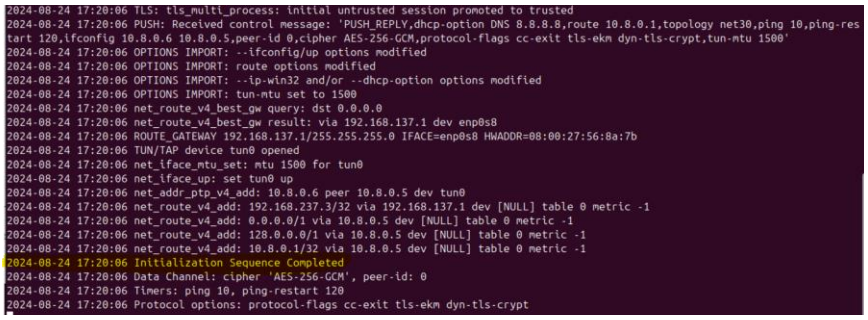
[8] https://docs.docker.com/engine/storage/volumes/

To connect from a linux client we install OpenVPN using apt-get, and we just launch the following command in the terminal:

```
sudo openvpn client.conf
```



```
ramon@clientubuntu: ~/VPN_Client_Files
ramon@clientubuntu:~/VPN_Client_Files$ sudo openvpn client.conf
[sudo] password for ramon:
2024-08-24 17:19:49 DEPRECATED OPTION: --cipher set to 'AES-256-CBC' but missing in --data-ciphers (AES-256-GCM:AES-128-GCM:CHACHA20-P
OLY1305). OpenVPN ignores --cipher for cipher negotiations.
2024-08-24 17:19:49 Note: Kernel support for ovpn-dco missing, disabling data channel offload.
2024-08-24 17:19:49 OpenVPN 2.6.9 x86_64-pc-linux-gnu [SSL (OpenSSL)] [LZO] [LZ4] [EPOLL] [PKCS11] [MH/PKTINFO] [AEAD] [DCO]
2024-08-24 17:19:49 library versions: OpenSSL 3.0.13 30 Jan 2024, LZO 2.10
2024-08-24 17:19:49 DCO version: N/A
Enter Private Key Password: ••••••••••••••••••
2024-08-24 17:20:06 WARNING: this configuration may cache passwords in memory -- use the auth-nocache option to prevent this
2024-08-24 17:20:06 TCP/UDP: Preserving recently used remote address: [AF_INET]192.168.237.3:1194
2024-08-24 17:20:06 Socket Buffers: R=[212992->212992] S=[212992->212992]
2024-08-24 17:20:06 UDPv4 link local: (not bound)
2024-08-24 17:20:06 UDPv4 link remote: [AF_INET]192.168.237.3:1194
2024-08-24 17:20:06 TLS: Initial packet from [AF_INET]192.168.237.3:1194, sid=14edf460 045a0b6e
2024-08-24 17:20:06 VERIFY OK: depth=1, CN=ca_unix_assignment_vpn
2024-08-24 17:20:06 VERIFY KU OK
2024-08-24 17:20:06 Validating certificate extended key usage
2024-08-24 17:20:06 ++ Certificate has EKU (str) TLS Web Server Authentication, expects TLS Web Server Authentication
2024-08-24 17:20:06 VERIFY EKU OK
```



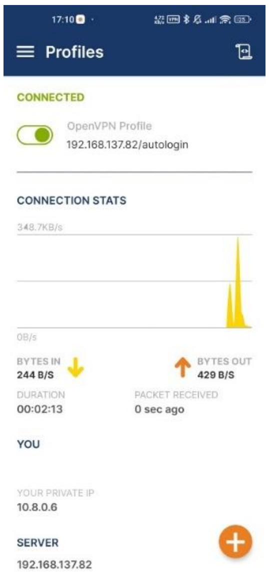
```
2024-08-24 17:20:06 TLS: tls_multi_process: initial untrusted session promoted to trusted
2024-08-24 17:20:06 PUSH: Received control message: 'PUSH_REPLY,dhcp-option DNS 8.8.8.8,route 10.8.0.1,topology net30,ping 10,ping-res
tart 120,ifconfig 10.8.0.6 10.8.0.5,peer-id 0,cipher AES-256-GCM,protocol-flags cc-exit tls-ekm dyn-tls-crypt,tun-mtu 1500'
2024-08-24 17:20:06 OPTIONS IMPORT: --ifconfig/up options modified
2024-08-24 17:20:06 OPTIONS IMPORT: route options modified
2024-08-24 17:20:06 OPTIONS IMPORT: --ip-win32 and/or --dhcp-option options modified
2024-08-24 17:20:06 OPTIONS IMPORT: tun-mtu set to 1500
2024-08-24 17:20:06 net_route_v4_best_gw query: dst 0.0.0.0
2024-08-24 17:20:06 net_route_v4_best_gw result: via 192.168.137.1 dev enp0s8
2024-08-24 17:20:06 ROUTE_GATEWAY 192.168.137.1/255.255.255.0 IFACE=enp0s8 HWADDR=08:00:27:56:8a:7b
2024-08-24 17:20:06 TUN/TAP device tun0 opened
2024-08-24 17:20:06 net_iface_mtu_set: mtu 1500 for tun0
2024-08-24 17:20:06 net_iface_up: set tun0 up
2024-08-24 17:20:06 net_addr_ptp_v4_add: 10.8.0.6 peer 10.8.0.5 dev tun0
2024-08-24 17:20:06 net_route_v4_add: 192.168.237.3/32 via 192.168.137.1 dev [NULL] table 0 metric -1
2024-08-24 17:20:06 net_route_v4_add: 0.0.0.0/1 via 10.8.0.5 dev [NULL] table 0 metric -1
2024-08-24 17:20:06 net_route_v4_add: 128.0.0.0/1 via 10.8.0.5 dev [NULL] table 0 metric -1
2024-08-24 17:20:06 net_route_v4_add: 10.8.0.1/32 via 10.8.0.5 dev [NULL] table 0 metric -1
2024-08-24 17:20:06 Initialization Sequence Completed
2024-08-24 17:20:06 Data Channel: cipher 'AES-256-GCM', peer-id: 0
2024-08-24 17:20:06 Timers: ping 10, ping-restart 120
2024-08-24 17:20:06 Protocol options: protocol-flags cc-exit tls-ekm dyn-tls-crypt
```

Figure 1. Output of connecting the client to the VPN server

And we can observe that the initialization is completed.

Android client

We can download the client from any app store[9]. Once we have it installed, we transfer the client files to the phone and start the session. Here we can see how our phone, connected only to the VPN hosted by the VM, gets internet access:



17:10

Profiles

CONNECTED

OpenVPN Profile
192.168.137.82/autologin

CONNECTION STATS

348.7KB/s

0B/s

BYTES IN
244 B/S

BYTES OUT
429 B/S

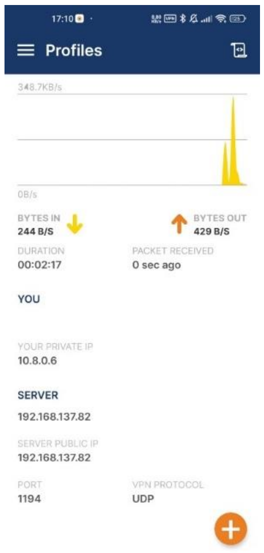
DURATION
00:02:13

PACKET RECEIVED
0 sec ago

YOU

YOUR PRIVATE IP
10.8.0.6

SERVER
192.168.137.82



17:10

Profiles

348.7KB/s

0B/s

BYTES IN
244 B/S

BYTES OUT
429 B/S

DURATION
00:02:17

PACKET RECEIVED
0 sec ago

YOU

YOUR PRIVATE IP
10.8.0.6

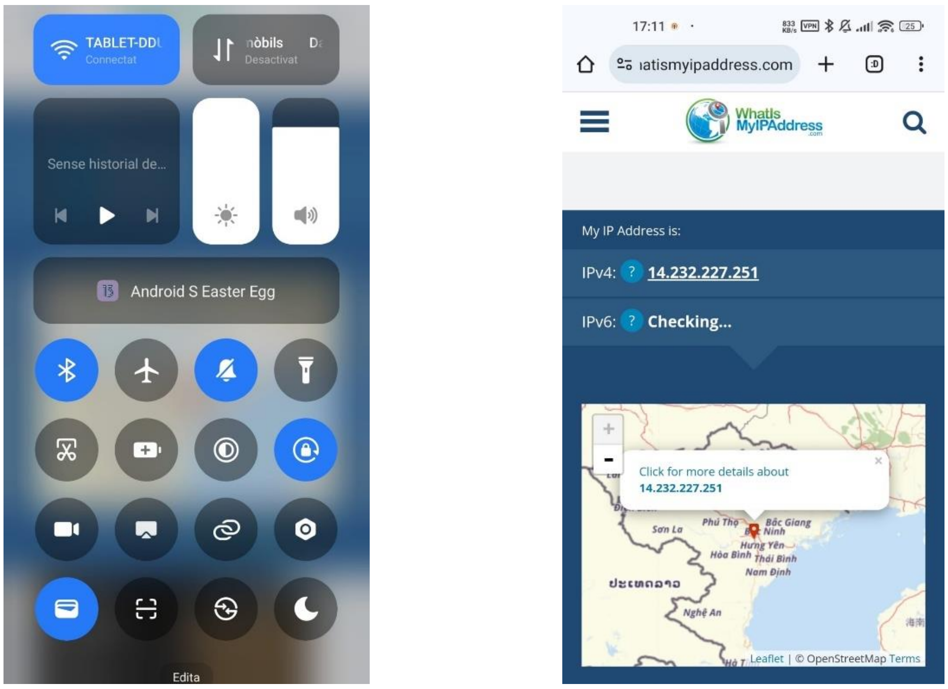
SERVER
192.168.137.82

SERVER PUBLIC IP
192.168.137.82

PORT
1194

VPN PROTOCOL
UDP

[9] We used the Google Play Store: https://play.google.com/store/apps/details?id=net.openvpn.openvpn&hl=en



My IP Address is:

IPv4: 14.232.227.251

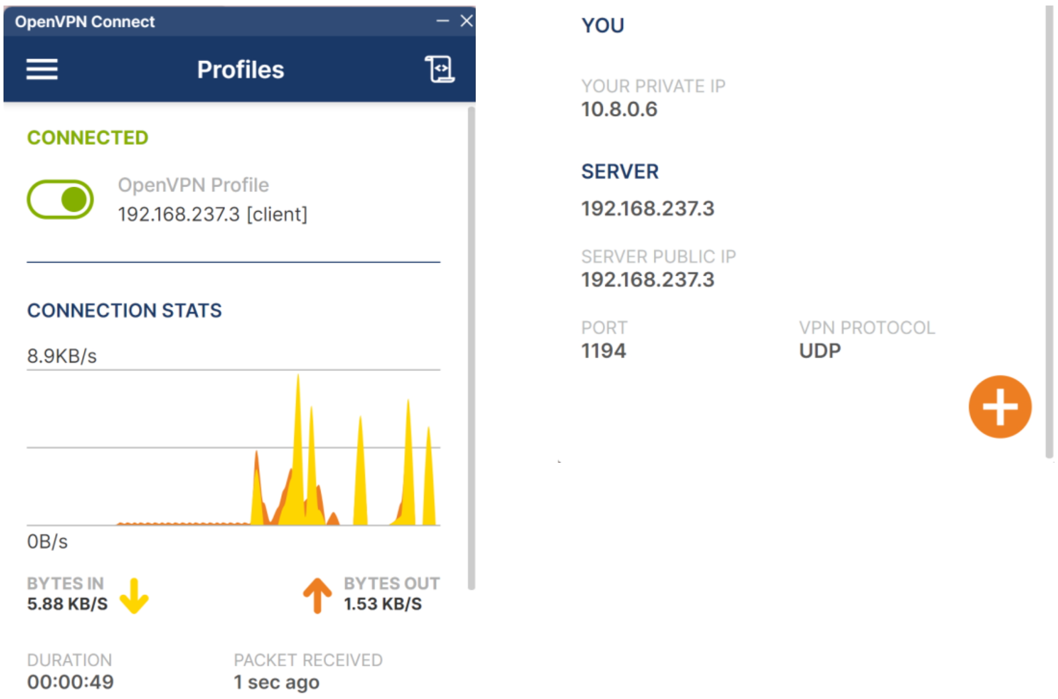
IPv6: Checking...

Click for more details about 14.232.227.251

Figure 2. Screenshots of the successful configuration in an Android device

Windows client

In this case, we download and install the client from the official OpenVPN website[10]. Once we have done this, we launch it and we import the client.conf file. Note that we need to change the extension to .ovpn instead of .conf. Once we do this, the client successfully connects:



CONNECTED

OpenVPN Profile
192.168.237.3 [client]

CONNECTION STATS

8.9KB/s

0B/s

BYTES IN 5.88 KB/S

BYTES OUT 1.53 KB/S

DURATION 00:00:49

PACKET RECEIVED 1 sec ago

YOU

YOUR PRIVATE IP 10.8.0.6

SERVER

192.168.237.3

SERVER PUBLIC IP 192.168.237.3

PORT 1194

VPN PROTOCOL UDP

Figure 3. Screenshots of the successful configuration in a Windows device

[10] https://openvpn.net/client/client-connect-vpn-for-windows/

3.3. Use case

Our tool can be used in order to bypass some regional restrictions on content imposed by web managers. We will work with an example: Imagine a user wants to watch some content, such as:

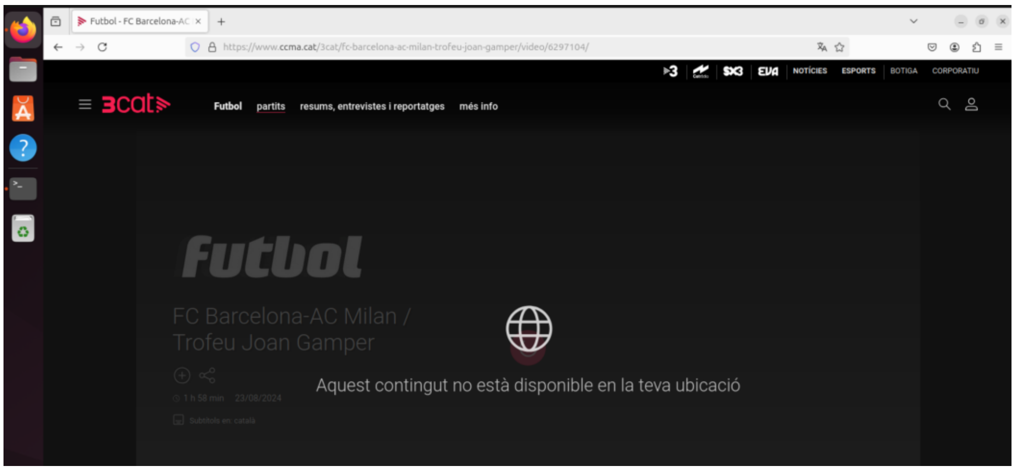
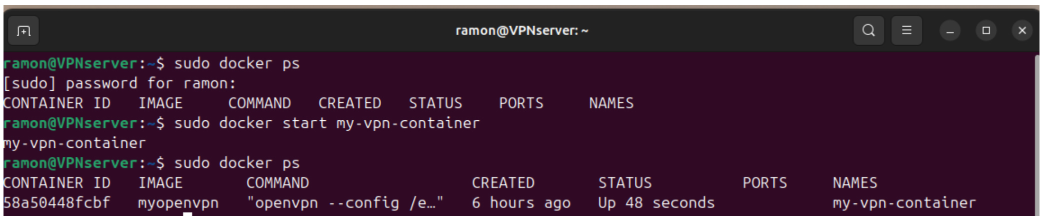

Futbol
FC Barcelona-AC Milan / Trofeu Joan Gamper
Aquest contingut no està disponible en la teva ubicació

Figure 4. Content not available due to regional restrictions based on IP

The server identifies the IP address from Vietnam and doesn't allow the user to view it. At this moment, he establishes a connection to the VPN Server that he has running in a local machine using Ubuntu of his own in Spain, using the Docker image we provided to him:



```
ramon@VPNserver:~$ sudo docker ps
[sudo] password for ramon:
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
ramon@VPNserver:~$ sudo docker start my-vpn-container
my-vpn-container
ramon@VPNserver:~$ sudo docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED       STATUS          PORTS     NAMES
58a50448fcbf   myopenvpn   "openvpn --config /e…"   6 hours ago   Up 48 seconds             my-vpn-container
```

Figure 5. Starting the docker image containing the VPN server

Once the server is running, he can connect to it from different systems as we explained in the previous section. After doing so, he reloads the page and this is the result:

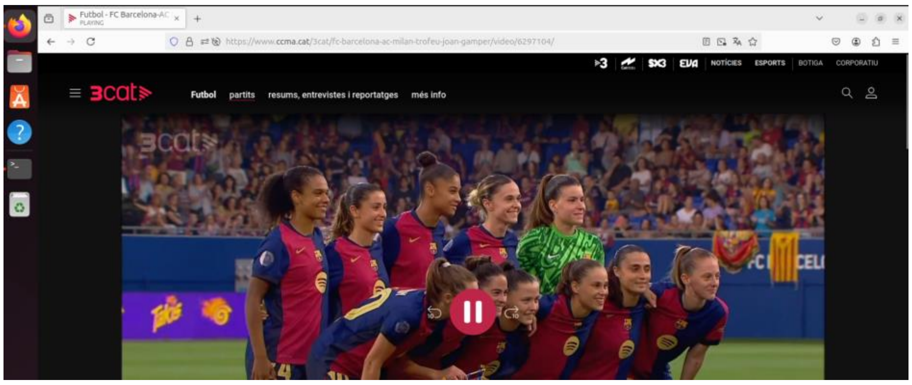


Figure 6. Successfully watching content thanks to the VPN

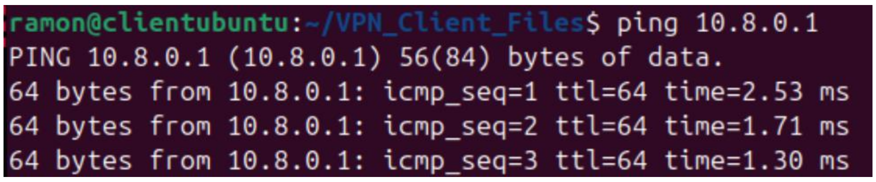
As we can see, our application provides easy configuration steps to set up a VPN server to achieve this.

4. Testing

In order to demonstrate the system is functional for our client, we did some testing while setting up our server and connecting as a client. The tricky part about the testing was to achieve connectivity between different devices using Virtual Box and the different network adapters.

Checking connectivity between client and server

One of the first steps to check after setting up a VPN server is to check if there is connectivity to it from a client. To do so, we ping between client and the server address. It is important to note that we don't need to ping the public address of the server (the one present in the client.conf file), as it is obviously reachable. We need to ping the internal address, i.e., the IP address assigned to the client by the server. Hence, we executed the ping 10.8.0.1 command from the client machine, where 10.8.0.1 is the server's internal VPN address. A successful response to the ping confirms that the VPN tunnel is active:



```
ramon@clientubuntu:~/VPN_Client_Files$ ping 10.8.0.1
PING 10.8.0.1 (10.8.0.1) 56(84) bytes of data.
64 bytes from 10.8.0.1: icmp_seq=1 ttl=64 time=2.53 ms
64 bytes from 10.8.0.1: icmp_seq=2 ttl=64 time=1.71 ms
64 bytes from 10.8.0.1: icmp_seq=3 ttl=64 time=1.30 ms
```

Accessing the internet

Once the VPN is established, we need to test if the client can access the internet. This needs to be done by configuring the server to route the traffic in a proper way. With the virtualbox bridged adapter, after we had configured the firewall already using ufw commands, together with some packet forwarding instructions, the client had access to the internet through the VPN server.

However, we wanted to simulate and test if the client could access content from different regions (as in the use case). We used a VPN in the host machine and set up the server VM with a NAT adapter, instead of bridging, which made things more difficult to set up. We realized that the client could ping a DHCP server (like 8.8.8.8) but could not ping websites (google.com). So we added firewall rules to allow DNS traffic:

```
sudo iptables -A FORWARD -p udp --dport 53 -j ACCEPT

sudo iptables -A FORWARD -p tcp --dport 53 -j ACCEPT
```

After this, the client could ping websites form the terminal, but still couldn't access them through the browser. This made us think to something related to ports and

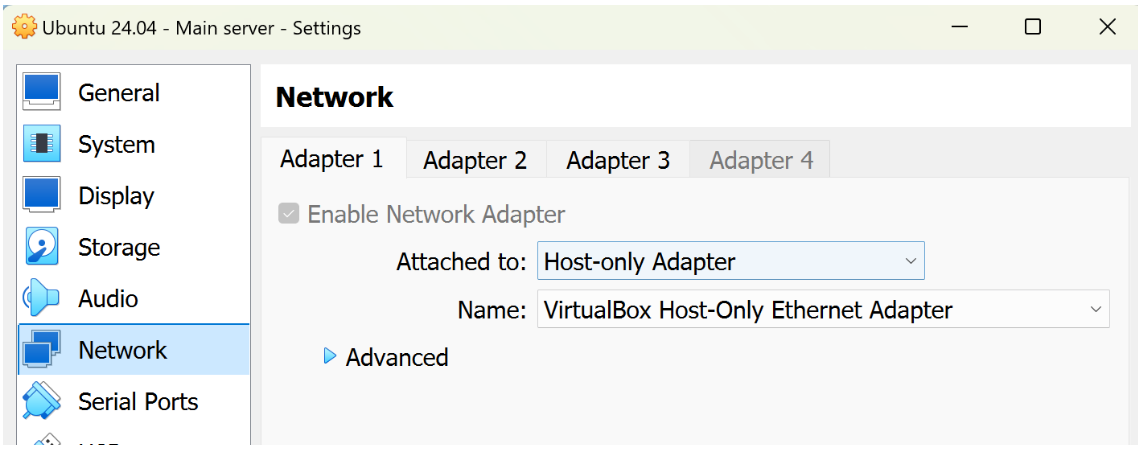
HTTP and HTTPS protocols. After some research, we implemented the following commands:

```
sudo iptables -A FORWARD -p tcp --dport 80 -j ACCEPT
sudo iptables -A FORWARD -p tcp --dport 443 -j ACCEPT
```

After this, the client had full connectivity to the internet.

Achieving connectivity between Android phone and the VPN server in the VM

In order to simulate and test the connection of an Android phone, we had to connect it in some way to the same network as the virtual box VM acting as a server. To do so, after trying different option without success, we thought about a trick that ended up working. As we explained, we were using the host only adapter:

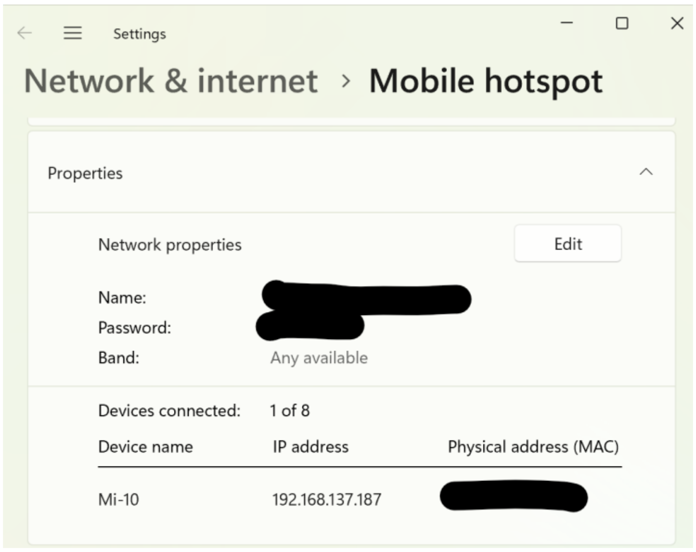


Enable Network Adapter

Attached to: Host-only Adapter

Name: VirtualBox Host-Only Ethernet Adapter

It was very difficult to achieve connectivity to this adapter from the Android phone, so we connected the phone to the host machine using the hotspot option of Windows[11]. At this point, our phone obtained an IP local address on the interface:



Network properties

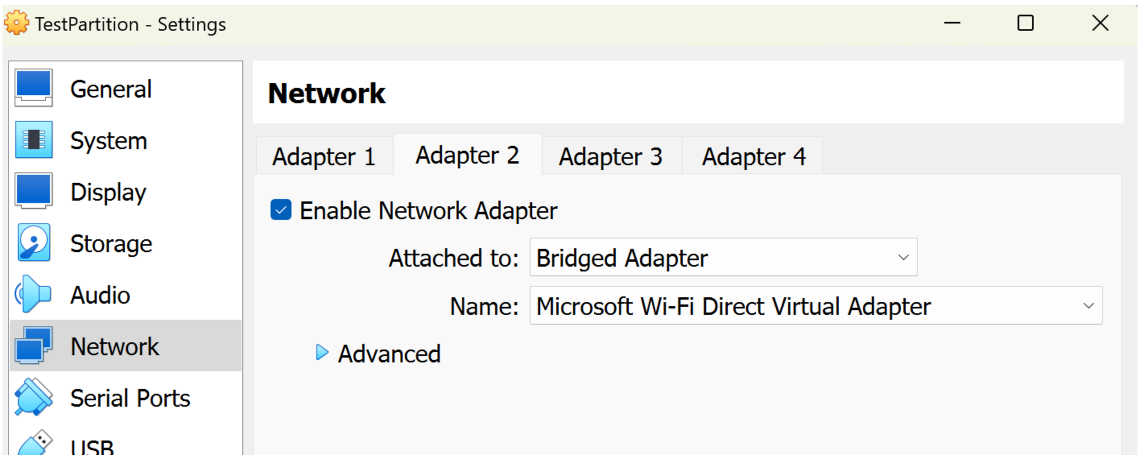
Name: [redacted]
Password: [redacted]
Band: Any available

Devices connected: 1 of 8

Device name	IP address	Physical address (MAC)
Mi-10	192.168.137.187	[redacted]

[11] https://support.microsoft.com/en-us/windows/use-your-windows-pc-as-a-mobile-hotspot-c89b0fad-72d5-41e8-f7ea-406ad9036b85

So now we added this same adapter to the VM:


TestPartition - Settings

General
System
Display
Storage
Audio
Network
Serial Ports
USB

Network

Adapter 1 | Adapter 2 | Adapter 3 | Adapter 4

☑ Enable Network Adapter

Attached to: Bridged Adapter

Name: Microsoft Wi-Fi Direct Virtual Adapter

▷ Advanced

And then we changed the client.ovpn file to insert the IP address of the VM in the same subnetwork as the Android device. After this settings, we were able to successfully test connectivity between the phone and the VPN server.

5. User Documentation

In this section you will find a brief user documentation with the key topics to set up your VPN.

5.1. Creating and distributing PKI files

First we initialize the pki directory. Here are the steps to initialize it in an Ubuntu machine:

```
sudo apt update
sudo apt install openssl git
git clone https://github.com/OpenVPN/easy-rsa.git
cd easy-rsa
./easyrsa3 init-pki
cd easyrsa3
./easyrsa init-pki
```

1. CA certificate and key

In the same directory we initialized the new PKI, we also create a Certificate Authority (CA). I.e., on the CA system we perform the following commands:

```
./easyrsa init-pki
./easyrsa build-ca
```

2. Server certificate and key

- On the server system:

  ```
  ./easyrsa init-pki
  ./easyrsa gen-req server_name
  ```

- Transfer the server.req file to the CA system.
- On the CA system:

  ```
  ./easyrsa import-req /path/to/server.req server
  ./easyrsa sign-req server server_name
  ```

- Transfer the server_name.crt back to the server system.

3. Client certificates and keys

- For each client, on the client system or a central management system:

  ```
  ./easyrsa init-pki
  ./easyrsa gen-req client_name
  ```

  Where client_name can be changed to the name we want.

- Transfer each client's .req file to the CA system.
- On the CA system, for each client we perform:

  ```
  ./easyrsa import-req /path/to/client_name.req client_name
  ./easyrsa sign-req client client_name
  ```

- Transfer each client's .crt file back to the respective client system.

4. Diffie-Hellman parameters

- On the server system:

  ```
  ./easyrsa gen-dh
  ```

5. We distribute files

- CA certificate (ca.crt): every client and the server need it.
- Server certificate (server.crt) and key (server.key): the server needs them.
- Client certificates and keys: each client needs its own pair.
- DH parameters (dh.pem): the server needs it.

5.2. Network configuration

To make the VPN server operate correctly, it is essential to configure the network properly. Here is a summary:

Firewall Setup: On the server, configure the firewall to allow OpenVPN traffic.

IP Forwarding: Enable IP forwarding to allow the server to route traffic between clients and the internet.

```
sudo sysctl -w net.ipv4.ip_forward=1
```

You should enable permanently editing /etc/sysctl.conf and uncommenting the line net.ipv4.ip_forward=1.

NAT Configuration: configure it to allow clients to access the internet using the server. For example:

```
sudo iptables -t nat -A POSTROUTING -o <interface> -j MASQUERADE
```

Where <interface> should be changed by the network interface your device is using to access the internet.

5.3. Using Docker to start the OpenVPN server

The OpenVPN server can be easily started using Docker. You do not need to build the image yourself or do annoying configurations. Just follow the steps below:

1. Install Docker:

   ```
   sudo apt update
   sudo apt install docker.io
   ```

2. Pull the Docker image:

   ```
   docker pull ramonupf/openvpncontainer:latest
   ```

3. Run the Docker Container and start it:
   sudo docker run -d --name vpnserver --privileged --network host -p 1194:1194/udp -v /path/to/pki/:/etc/openvpn/pki ramonupf/openvpncontainer

These steps will allow you to have the server running and ready for clients to connect.

5.4. Using OpenVPN in different platforms as a client

There are clients of OpenVPN for various platforms. For all of them you will need to have the ca file, your key an crt files together with your configuration file. We have tested the following, which you as a user can use:

- **Linux**: install OpenVPN via package manager and then connect using the configuration file.

  ```
  sudo apt install openvpn
  sudo openvpn client_name.conf
  ```

- **Windows**: download and install OpenVPN from the official site, import the .ovpn file with the other necessary files and connect.
- **Android**: install the app from the app store. Proceed as in Windows.

5.5. FAQS

What happens if I run the server from a different machine? Will the IP be maintained?

No, the public IP address will change unless you have a static IP. You'll need to update the client configuration files with the new IP.

How do I allow a new client to connect to the server?

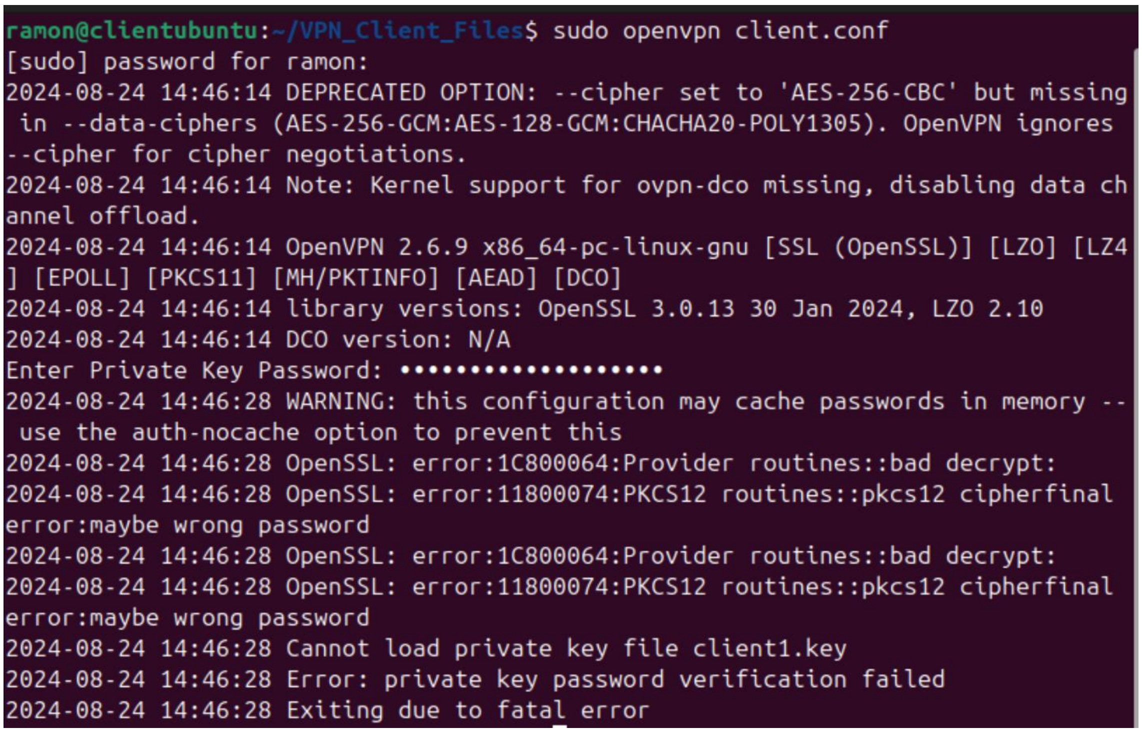
You can generate new client certificates and keys as explained in section 5.1. Distribute them securely to the new client.

Changing the CA in case of security breaches.

If the CA is compromised, revoke all certificates, create a new CA, and redo the certificates for all the clients and the server.

Can I use the same client files in two different devices? Simultaneously?

Although it is not recommended for security reasons, you can use the same client files in different devices. However, our testing has shown that it is not possible to do it simultaneously.



```
ramon@clientubuntu:~/VPN_Client_Files$ sudo openvpn client.conf
[sudo] password for ramon:
2024-08-24 14:46:14 DEPRECATED OPTION: --cipher set to 'AES-256-CBC' but missing
 in --data-ciphers (AES-256-GCM:AES-128-GCM:CHACHA20-POLY1305). OpenVPN ignores
--cipher for cipher negotiations.
2024-08-24 14:46:14 Note: Kernel support for ovpn-dco missing, disabling data ch
annel offload.
2024-08-24 14:46:14 OpenVPN 2.6.9 x86_64-pc-linux-gnu [SSL (OpenSSL)] [LZO] [LZ4
] [EPOLL] [PKCS11] [MH/PKTINFO] [AEAD] [DCO]
2024-08-24 14:46:14 library versions: OpenSSL 3.0.13 30 Jan 2024, LZO 2.10
2024-08-24 14:46:14 DCO version: N/A
Enter Private Key Password: •••••••••••••••••••
2024-08-24 14:46:28 WARNING: this configuration may cache passwords in memory --
 use the auth-nocache option to prevent this
2024-08-24 14:46:28 OpenSSL: error:1C800064:Provider routines::bad decrypt:
2024-08-24 14:46:28 OpenSSL: error:11800074:PKCS12 routines::pkcs12 cipherfinal
error:maybe wrong password
2024-08-24 14:46:28 OpenSSL: error:1C800064:Provider routines::bad decrypt:
2024-08-24 14:46:28 OpenSSL: error:11800074:PKCS12 routines::pkcs12 cipherfinal
error:maybe wrong password
2024-08-24 14:46:28 Cannot load private key file client1.key
2024-08-24 14:46:28 Error: private key password verification failed
2024-08-24 14:46:28 Exiting due to fatal error
```

Figure 7. Trying to access with the same client files from two devices at the same time

6. Reflection on learning and skills acquired

During this project I have increased my understanding of VPNs and its associated technologies. Setting up a VPN required knowledge of networking concepts I didn't have before. Moreover, this work has provided me with a first hands-on experience as a Linux system administrator. The task was more complex than it seemed at the beginning of it, and I realized the importance of having a deep understanding of Linux networking tools and commands. In addition, I understood why Linux is such a valuable tool: it lets the administrator configure almost anything very easily and without relying on third-party administrators.

One of the main difficulties I faced during this project is that I didn't have no access to any real network with a router I could work with. However, this turned out to be an opportunity to learn more than I expected. I invested a lot of time looking for alternatives on how to do so. I learned a lot about networking and port forwarding, and I also understood the importance virtual environments for simulation purposes.

Finally, writing the report really made me deepen into things I would not have focus my attention on. Having to document every step and keep track of what you do is a valuable resource to learn and solidify knowledge. In the following table there is an estimate of the time I invested in each part of the project:

Tool/concept	Work done	Time invested
OpenVPN	Reading documentation, experimenting with various configurations, learning VPN theory, setting up the server and client, and troubleshooting network issues.	10 hours
PKI and Easyrsa	Understanding cryptographic concepts, installing EasyRSA, creating and managing certificates, learning about certificate authorities, and securing client-server communications.	5 hours
Docker	Reading Docker documentation, learning about containerization, configuring containers with network options, building Docker images, testing containerized VPN servers, and trying previously built containers from github.	4 hours
Virtual Box	Building virtual machines, learning about networking inside Virtual Box, trying different configurations.	10 hours
Networking	Configuring firewalls, setting up IP forwarding, NAT, and port forwarding, troubleshooting connectivity issues across different devices, and testing configurations in a virtualized environment.	15 hours

Troubleshooting and others	Thinking about the structure of the project, launching different configurations, solving small problems that didn't let me advance...	15 hours

7. Video demonstration

The video demonstration can be found here: https://youtu.be/qVJZCv6i_os

Please feel free to contact me for demonstrations of any of the parts in the report.

8. External resources used

Canonical Ltd. (2024). *About the Ubuntu project | Ubuntu*. [online] Ubuntu. Available at: https://ubuntu.com/about

Openvpn.net (2024). *OpenVPN Community*. [online] Available at: https://community.openvpn.net/openvpn/wiki

Ubuntu (2024). *Installation/SystemRequirements - Community Help Wiki*. [online] Ubuntu.com. Available at: https://help.ubuntu.com/community/Installation/SystemRequirements

OpenVPN (2024). *How To Guide: Set Up & Configure OpenVPN Client/server VPN*. [online] Available at: https://openvpn.net/community-resources/how-to

IBM Technology (2022). *Tech Talk: What is Public Key Infrastructure (PKI)?*. 10 September. Available at: https://youtu.be/0ctat6RBrFo?si=7YGeXfTOJvpT8e__

OpenVPN (2024). *easy-rsa - Simple shell based CA utility. Github*. https://github.com/OpenVPN/easy-rsa

Crist, E.F. and Keijser, J.J., 2015. *Mastering OpenVPN*. Packt Publishing Ltd.

Keijser, J.J., 2017. *OpenVPN Cookbook*. Packt Publishing Ltd.

Kylemanna (2020). *OpenVPN server in a Docker container complete with an EasyRSA PKI CA. Github*. https://github.com/kylemanna/docker-openvpn

Gite, V. (2020). *How To Configure Firewall with UFW on Ubuntu 20.04 LTS*. [online] nixCraft. Available at: https://www.cyberciti.biz/faq/how-to-configure-firewall-with-ufw-on-ubuntu-20-04-lts/