5

CREATING A VIRTUAL PRIVATE NETWORK

A *virtual private network (VPN)* is a means of providing privacy and security for communications over the public internet. If you don't want a malicious third party to intercept your Google search traffic as it traverses the internet from your local laptop to Google's servers, you should use a VPN to encrypt the traffic between the two endpoints. If you frequently transfer sensitive files or data, such as personally identifiable information or banking information, it's wise to protect this information using encryption.

The other primary function of a VPN is to extend a private network, such as those in homes and offices, from one geographic location to another. A VPN creates a tunnel over the internet from one network to a second network. This means that if a user usually based in Australia is traveling in Europe, they could connect to their home network from Europe as if they were physically located in Australia. Conversely, if a user located in Australia wants to *appear* as if they're physically located in Europe, they can place the VPN endpoint in Europe, usually via some third-party service.

This chapter outlines a method for creating a private VPN whose *exit node* (that is, the place where the VPN tunnel ends) is located somewhere outside of your local network, in a different geographic location somewhere in the world, to make your actual physical location private. We'll discuss how to achieve this with OpenVPN or Wireguard.

Drawbacks of Third-Party VPNs and Remote Access Services

Although you could subscribe to a VPN service like NordVPN or ExpressVPN, operating your own VPN is beneficial because you control everything about it, including connection and traffic logging levels, as well as the cost of the service. Also, whereas third-party services provide some benefits, such as the possibility of using multiple exit nodes in different locations, they usually don't offer the ability to connect into your own network remotely. One last challenge of using third-party VPN services is that they usually set a limit to the number of devices you can connect at a time. A privately managed VPN has no such limitation.

Recently, there's been a boom in the number of applications designed to allow remote access to endpoints from the wider internet. This includes software and vendors such as Teamviewer and AnyDesk. Although these solutions are convenient and have a low barrier to entry, they increase the attack surface of your private network by opening up remote access from your computer to the internet, something you should rarely do, if ever. There have also been several well-known compromises of these solutions, indicating that they are vulnerable to attack. A VPN provides a much more secure solution.

OpenVPN

OpenVPN is one of the most common VPN solutions available. Because of its age and ubiquity, you can be confident in its security when compared to newer solutions, which have been less rigorously tested for bugs and vulnerabilities. OpenVPN comes built-in to a variety of networking hardware, which is beneficial because in a lot of cases your router can act as your VPN endpoint inside of your network (that is, the VPN server). This means your router can also act as a VPN client, connecting to a VPN server in the cloud, and then everything connected to the router on your internal network can send and receive traffic via your VPN tunnel. Encrypting your internet traffic in this way provides much greater privacy than using the internet without a VPN. Ideally, though, you'll want more control over the VPN exit node than this allows; most routers use either a cut-down version of Linux or a proprietary operating system, so you'll learn how to create a VPN server using Ubuntu for greater flexibility.

EasyRSA

EasyRSA is a command-line utility for creating and managing certificate authorities. To encrypt and protect traffic, OpenVPN requires a *certificate authority (CA)* to issue certificates. *Digital certificates* are used to enable trust between different parties, usually networks and computers. *Public key infrastructure (PKI)* is responsible for the distribution, authentication, and revocation of public key certificates, which are used to verify ownership of digital certificates. These certificates contain the public key that an entity uses as part of a public/private key pair to encrypt data, which can then be decrypted only by the public key owner with the matching private key. This method secures most communication on the internet today.

The CA you create will generate, sign, verify, and revoke (if necessary) all the certificates required to encrypt and secure communications between the VPN server and VPN clients. Technically, you can install both OpenVPN and the CA on the same server, but doing so is less secure than installing them on separate servers. Any adversary who gains access to the server would have access to the certificates and private keys used by the server, as well as the ability to generate new certificates. Therefore, you'll need two Ubuntu servers: one to act as the OpenVPN server and one to serve as the certificate server. You'll use the certificate server to sign requests generated on the OpenVPN server for both the VPN server and any client devices connecting to the VPN, whether they're laptops, workstations, mobile devices, or any other type of device.

Wireguard

Wireguard, a relatively new alternative to OpenVPN, is simple and incredibly fast by comparison. The drawback of being newer is that although Wireguard is open source, it's had less time to be tested for bugs and vulnerabilities. However, it has gathered a sizable following in the security community and a good reputation for being reliable and secure.

NOTE *If you plan to connect to your private network remotely, keep in mind that you'll need a static IP address on your home or office internet connection, as well as some port forwarding on your border router. Most internet service providers supply static IP addresses upon request, usually for a nominal fee.*

#19: Creating a VPN with OpenVPN

In this first project, you'll start by creating an OpenVPN server and a certificate authority to secure communication via the VPN. Next, you'll generate the relevant certificates, create the OpenVPN configuration files, configure the host firewall, and start the VPN. Finally, you'll configure each of the VPN clients that will use this VPN to send and receive traffic, and you'll connect to and test the VPN connection.

The entire process of spinning up an OpenVPN server in the cloud and connecting a client to it should take no longer than a couple of hours. Adding subsequent clients should take up to 30 minutes per endpoint. You'll need to enable and configure a firewall on your server as part of creating your VPN. Ubuntu's built-in firewall, the *Uncomplicated Firewall (UFW)*, is designed to reduce the complexity of firewall configuration. It's much simpler than solutions like iptables (covered in Chapter 3). We'll introduce you to UFW and its use in this project as an alternative host firewall solution. Alternatively, you can apply what you learned in Chapter 3 and implement the same rules described for UFW in an iptables deployment. Even if you have a perimeter firewall in place like pfSense, be sure to enable either the host-based firewall offered by Ubuntu or iptables, as laid out in Chapter 3, to provide an additional layer of protection at the host level. Implementing a host-based firewall also allows more granular configuration of the servers' network connections.

Once you enable the firewall, you'll have to adjust the settings of the Ubuntu installation so that OpenVPN traffic is capable of traversing that firewall. (I'll cover how to do this later in the project.)

Securing internet traffic originating inside your network will require a VPN exit node elsewhere, as well as a certificate server, so follow Project 3 in Chapter 1 to create two base Ubuntu servers in the cloud, using the cloud service provider of your choice.

Once your Ubuntu servers are up and running, log in to the server you plan to use as your OpenVPN server (as opposed to the certificate authority) via SSH as a standard, non-root user:

```
$ ssh user@your_server_IP
```

Once logged in to the OpenVPN server, at the bash terminal, use `apt` to install OpenVPN:

```
$ sudo apt install openvpn -y
```

You also need to install EasyRSA on both the OpenVPN server and the certificate server. Install the latest version using `apt` as well:

```
$ sudo apt install easy-rsa -y
```

Make sure to install this on both Ubuntu servers. EasyRSA will be installed to the */usr/share/easy-rsa/* directory by default.

Set Up the Certificate Authority

Next, you must configure and build the certificate server to act as a CA. The easiest way to do this is to make a copy of the template that EasyRSA provides and then modify its configuration to suit your needs. You can then initialize the PKI, build the CA, and generate its public certificate and private key.

Navigate to the *easy-rsa* folder on the certificate server and then create a copy of the *vars.example* file. Call it *vars*:

```
$ cd /usr/share/easy-rsa/

$ sudo cp vars.example vars
```

Keep in mind that most of the time when a command in bash runs successfully, there will be no output to the screen, and you'll be returned to the prompt.

Open the resulting *vars* file in a text editor:

```
$ sudo nano vars
```

In the file, find the *organizational fields* that contain information about the organization for which the certificates will be generated by this server; for example:

```
--snip--

#set_var EASYRSA_REQ_COUNTRY    "US"
```

```
#set_var EASYRSA_REQ_PROVINCE   "California"

#set_var EASYRSA_REQ_CITY       "San Francisco"

#set_var EASYRSA_REQ_ORG        "Copyleft Certificate Co"

#set_var EASYRSA_REQ_EMAIL      "me@example.net"

#set_var EASYRSA_REQ_OU         "My Organizational Unit"

--snip--
```

Each of these lines in the file is a comment by default, so they won't be read or interpreted when the file is run; they'll be ignored or suppressed. Remove the hash (`#`) at the beginning of each line to ensure they're read when this file is invoked. Alter the values in quotations on the right to match your organization or personal network. The values can be anything you choose, but they can't be blank. Here's an example:

```
--snip--

set_var EASYRSA_REQ_COUNTRY    "AU"

set_var EASYRSA_REQ_PROVINCE   "Queensland"

set_var EASYRSA_REQ_CITY       "Brisbane"

set_var EASYRSA_REQ_ORG        "Smithco"

set_var EASYRSA_REQ_EMAIL      "john@smithco.net"

set_var EASYRSA_REQ_OU         "Cyber Unit"

--snip--
```

Save and close the file. Execute the `easyrsa` script within the *easy-rsa* folder (which should still be your current working directory) to initialize the PKI and then build the CA with the same `easyrsa` script, which will generate both the CA public certificate (*ca.crt*) and the private key (*ca.key*):

```
$ sudo ./easyrsa init-pki

--snip--

Your newly created PKI dir is: /usr/share/easy-rsa/pki

$ sudo ./easyrsa build-ca nopass

--snip--

CA creation complete and you may now import and sign cert requests.

Your new CA certificate file for publishing is at:

/usr/share/easy-rsa/pki/ca.crt
```

When prompted for the server's Common Name, you can enter any string of characters you'd like, but it's often easier to use the hostname of the server or press ENTER to accept the default Common Name. The output will contain the path to your PKI directory and *ca.crt* file; the *ca.key* file will be inside the *private* folder in the same location. The `nopass` option keeps you from being prompted for a password every time the CA is queried during this process.

That concludes the configuration of the CA server for now. The next set of configuration steps takes place on the OpenVPN server.

Create the OpenVPN Server Certificate and Key

Each client you plan to connect to the VPN needs its own public certificate and private key. These files allow the certificate server, the VPN server, and any other clients on the VPN to authenticate the client and enable communication between all devices within the VPN. The VPN server also needs its own certificate and key for the same reasons. This part of the project describes how to sign a certificate and generate a key for the OpenVPN server. You'll follow a similar process for connecting clients to the OpenVPN server.

On the OpenVPN server, navigate to the *easy-rsa* folder, and initialize the PKI for this server in the same way as before:

```
$ cd /usr/share/easy-rsa

$ sudo ./easyrsa init-pki
```

Just as every client connected to the VPN requires a certificate and key, the OpenVPN server itself needs a certificate signed by the CA. To this end, generate a certificate request from the OpenVPN server:

```
$ sudo ./easyrsa gen-req server nopass

Using SSL: openssl OpenSSL 1.1.1f 31 Mar 2020

Generating a RSA private key

...............................+++++

.......................................+++++

writing new private key to '/usr/share/easy-
rsa/pki/private/server.key.2ljAQtgUYY'

-----
```

```
You are about to be asked to enter information that will be
incorporated

into your certificate request.

What you are about to enter is what is called a Distinguished Name or
a DN.

There are quite a few fields but you can leave some blank

For some fields there will be a default value,

If you enter '.', the field will be left blank.

-----

Common Name (eg: your user, host, or server name) [server]:

Keypair and certificate request completed. Your files are:

req: /usr/share/easy-rsa/pki/reqs/server.req

key: /usr/share/easy-rsa/pki/private/server.key
```

When prompted, press ENTER to accept the default Common Name for the VPN server, `server`, or give it a custom name. The output indicates that an RSA private key is generated and shows where the script stored the resulting server key and certificate request.

Copy the generated *server.key* file to the OpenVPN configuration directory on the VPN server:

```
$ sudo cp /usr/share/easy-rsa/pki/private/server.key
/etc/openvpn/
```

Copy the *server.req* file to your certificate server using `rsync`, replacing the user and CA-ip placeholders with the relevant username and IP address for your certificate server:

```
$ sudo rsync -ruhP /usr/share/easy-rsa/pki/reqs/server.req
user @ CA _ ip :/tmp/
```

Next, enter the following commands to log in to your certificate server and then import and sign the VPN certificate request generated earlier, enabling the VPN communications to be encrypted and secured:

```
$ ssh user @ CA_ip

$ cd /usr/share/easy-rsa/

$ sudo ./easyrsa import-req /tmp/server.req ❶ server

$ sudo ./easyrsa sign-req ❷ server
```

The first `easyrsa import-req` command imports the request. The second argument is the Common Name you created for your VPN server earlier ❶. To sign the request, pass `easyrsa sign-req` the argument `server` ❷ to specify the request type and then the Common Name again. (Later, when signing client requests, you'll use the same command with `client` as the argument.)

When asked to confirm whether the details are correct, double-check to ensure the Common Name is set as expected and then type `yes` and press ENTER to complete the import and signing process. You'll need to copy the resulting *server.crt* certificate file belonging to the OpenVPN server (along with the CA certificate) back to the OpenVPN server from the CA server so that each can authenticate the other:

```
$ sudo rsync -ruhP /usr/share/easy-rsa/pki/issued/server.crt
user @ vpn_ip :/tmp/
```

```
$ sudo rsync -ruhP /usr/share/easy-rsa/pki/ca.crt
user @ vpn_ip :/tmp/
```

On the OpenVPN server, move the relevant files to the */etc/openvpn/* directory:

```
$ sudo mv /tmp/server.crt /etc/openvpn/

$ sudo mv /tmp/ca.crt /etc/openvpn/
```

Next, you'll need a Diffie-Hellman key to exchange keys between devices. A *Diffie-Hellman key exchange* is a way to communicate public and private key information between two parties over a public communication channel securely. Without this capability, it wouldn't be possible to create secure encrypted channels over a public network like the internet.

You'll also need an *HMAC signature* to make the process more secure. An HMAC signature, used in HMAC authentication and with a secret key, is a method of verifying the integrity of a message or payload. Using an HMAC signature in this process will maintain the key exchange's integrity and allow you to verify the keys' authenticity.

On your VPN server, navigate to your *easy-rsa* directory and generate a shared secret key using the `easyrsa` script created earlier:

```
$ cd /usr/share/easy-rsa/

$ sudo ./easyrsa ❶ gen-dh

$ sudo ❷ openvpn --genkey secret ta.key

$ sudo cp /usr/share/easy-rsa/ta.key /etc/openvpn/

$ sudo cp /usr/share/easy-rsa/pki/dh.pem /etc/openvpn/
```

The `gen-dh` argument ❶ creates the Diffie-Hellman key, which may take a long time and generate a lot of output. The `openvpn --gen-key secret` ❷ command quickly generates the HMAC signature, and you'll see no output if it's successful. These processes create the */usr/share/easy-rsa/ta.key* and */usr/share/easy-rsa/pki/dh.pem* files. Copy each of them to the OpenVPN configuration directory, */etc/openvpn/*, on your OpenVPN server:

```
$ sudo cp /usr/share/easy-rsa/ta.key /etc/openvpn/

$ sudo cp /usr/share/easy-rsa/pki/dh.pem /etc/openvpn/
```

At this point, you've created all the required certificates and keys for the servers.

Create a Client Certificate

Next, you'll need to create client certificates and keys to allow clients to connect to the VPN, which are the same as the server certificates but relate to each individual client device. The most efficient way to do this is to create the necessary files on the server, rather than on the client, which prevents you from having to transfer files between devices unnecessarily. On the OpenVPN server, create a safe location for the files:

```
$ sudo mkdir -p /etc/openvpn/client-configs/keys/
```

Navigate to the *easy-rsa* directory, generate a new certificate request for the client, copy the key to the directory you just created, and securely copy the request file to your CA server as shown here:

```
$ cd /usr/share/easy-rsa/

$ sudo ./easyrsa gen-req ❶ myclient nopass

$ sudo cp /usr/share/easy-rsa/pki/private/ myclient .key \
    /etc/openvpn/client-configs/keys/

$ sudo rsync -ruhP /usr/share/easy-
rsa/pki/reqs/ myclient .req user @ CA_ip :/tmp/
```

You'll be asked for a passphrase for the request; enter one and be sure to save it for later reference. You'll also be asked for a Common Name for your VPN client. This name will need to be different for each client that you provide access to the VPN, so consider using the client host-name (`myclient` in this example; change `myclient` ❶ to the client name of your choice).

On your certificate server, navigate to the *easy-rsa* directory:

```
$ cd /usr/share/easy-rsa/
```

Import the request using the client's Common Name (`myclient` in this example) and then sign it using the `client` directive, rather than the `server` directive you used earlier:

```
$ sudo ./easyrsa import-req /tmp/ myclient .req myclient

$ sudo ./easyrsa sign-req client myclient
```

Confirm that the Common Name is correct and then type `yes` and press ENTER to complete the command.

Finally, securely copy the newly generated certificate back to your OpenVPN server:

```
$ sudo rsync -ruhP /usr/share/easy-
rsa/pki/issued/myclient.crt user@vpn_ip :/tmp/
```

For the VPN to function correctly, the *ta.key* and *ca.crt* files you cre-ated earlier, as well as the *myclient.crt* file, need to be in the client configuration directory on the OpenVPN server. On your VPN server, copy those files to the */etc/openvpn/client-configs/keys/* directory:

```
$ sudo cp /usr/share/easy-rsa/ta.key /etc/openvpn/client-configs/keys/

$ sudo mv /tmp/ myclient .crt /etc/openvpn/client-configs/keys/

$ sudo cp /etc/openvpn/ca.crt /etc/openvpn/client-configs/keys/
```

And with that, you've created the necessary files to connect a client to the OpenVPN server. You can repeat this process as many times as necessary. Just be sure to change the client name from `myclient` to something else each time you generate files for a new client.

Configure OpenVPN

Now that the certificate server is set up, you can configure the OpenVPN server. To do so, you'll copy a template OpenVPN configuration and modify it to suit your needs.

On your OpenVPN server, copy the configuration template to the OpenVPN configuration directory:

```
$ sudo cp /usr/share/doc/openvpn/examples/sample-config-files/server.conf /etc/openvpn/
```

Open the resulting *server.conf* file in a text editor (this example uses nano):

```
$ sudo nano /etc/openvpn/server.conf
```

As with any configuration file, open it and familiarize yourself with its contents. You might notice that these configuration files use both `#`

and `;` to mark lines as comments.

Once you feel comfortable with the options available, you might decide to alter the port or protocol your VPN uses. Find the lines that start with `port` or `proto`, and notice a semicolon is used to comment out the inactive lines:

```
--snip--

port 1194

--snip--

; proto tcp

proto udp

--snip--
```

OpenVPN can run over either UDP or TCP, but it uses UDP by default, and the default port is 1194. However, you can tell it to run over any port you like, but if you make changes, you'll need to make those same changes in any commands or files that follow. Also, make sure that the certificates and keys mentioned in this file match your configurations from earlier sections of the chapter:

```
--snip--

ca ca.crt

cert server.crt

key server.key

--snip--
```

When you reach the Diffie-Hellman section, ensure that the file matches the one you created earlier; the configuration file lists *dh2048.pem* by default, which will need to be changed to *dh.pem*:

```
--snip--

# dh dh2048.pem

dh dh.pem

--snip--
```

In addition, the `redirect-gateway` and `dhcp-option` DNS directives should not be commented out, so remove the semicolons at the beginning of those lines:

```
--snip--

push "redirect-gateway def1 bypass-dhcp"

--snip--

push "dhcp-option DNS 208.67.222.222"

push "dhcp-option DNS 208.67.220.220"

--snip--
```

These directives ensure that all traffic will traverse the VPN and not the unsecured internet. You can leave DNS with the default settings, or you can set it to any DNS servers you desire, such as Quad9 (*9.9.9.9*), Google (*8.8.8.8*), or your Pi-Hole DNS server if you have one configured as described in **Chapter 7**.

Next, check that the `tls-auth` directive is set to `0` and not commented out with a semicolon and that the `cipher` is set to `AES-256-`

`CBC`. Then, immediately after the `cipher` directive, add an `auth` directive:

```
--snip--

tls-auth ta.key 0

--snip--

cipher AES-256-CBC

auth SHA256

--snip--
```

The `tls-auth` directive ensures that the HMAC signature you configured earlier will indeed be used to secure the VPN. Several settings are available for the cipher, and AES-256 is a reasonable choice as the encryption offered is good and well supported. The `SHA256` indicates the algorithm used for the HMAC message digest, meaning the hash calculated will be an SHA256 hash, which is considered secure and less prone to hash collisions than some other hashing algorithms.

To make the VPN more secure, remove the semicolons from the `user` and `group` directives, which makes the VPN service run with fewer privileges and ideally mitigates the risk of privilege escalation attacks:

```
--snip--

user nobody

group nogroup

--snip--
```

After making these changes, save and close the configuration file.

The OpenVPN configuration is complete, but you'll need to make some changes to the server's network settings. First, you must configure IP forwarding or the VPN won't do anything with the traffic received:

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

Reload `sysctl` to make the change take effect, as follows.

```
$ sudo sysctl -p

net.ipv4.ip_forward = 1
```

The command may output the lines modified in the *sysctl.conf* file.

Configure the Firewall

The first step in this process is to find your VPN server's public network interface; your server may have multiple network interfaces, and selecting the wrong interface for the following commands would result in a VPN that is unable to route traffic to the internet correctly:

```
$ ip route | grep -i default

default via 202.182.98.1 dev ens3 proto dhcp src 202.182.98.40 metric 100
```

In this output, the network interface is called `ens3`, but yours might be different. The *default route* shown by `ip route` will be the public network interface of your host. You'll need this to configure your firewall correctly.

In most firewalls, the order in which you set your rules is the most important consideration. In UFW, rules are evaluated from rule files in the following order: first *before.rules*, then *user.rules*, and finally *after.rules*. The firewall must correctly identify and push through the

VPN traffic, so rules are needed at the top of the firewall configuration. To do this in UFW, open the *before.rules* file in a text editor:

```
$ sudo nano /etc/ufw/before.rules
```

Then add these lines at the top of the file to allow OpenVPN client traffic via the public network interface you identified in the previous commands:

```
*nat

:POSTROUTING ACCEPT [0:0]

-A POSTROUTING -s 10.8.0.0/24 -o ens3 -j MASQUERADE

COMMIT
```

The network *10.8.0.0/24* indicates the addresses that clients connecting to your VPN will be assigned. These addresses should be different from the addresses used in your network. If you use *192.168.1.x* addresses in your network, do not use *192.168.1.x* addresses for your VPN network addressing. As long as your network uses addresses other than *10.8.0.x*, the previous configuration is safe to use.

Save and close the file. UFW also needs to accept, rather than drop, forwarded packets. You can allow this by changing the UFW configuration file:

```
$ sudo ufw default allow FORWARD
```

Finally, the firewall needs to allow the port and protocol used for the VPN to send and receive traffic, as well as SSH for server administration. Enter the following command to allow the correct port and protocol based on the configurations you set in *etc/openvpn/server.conf*:

```
$ sudo ufw allow 1194/udp
```

Next, allow OpenSSH:

```
$ sudo ufw allow OpenSSH
```

Restart the firewall for the changes to take effect permanently:

```
$ sudo ufw disable

$ sudo ufw enable
```

Your SSH connection might be interrupted as the firewall restarts, and you may need to log in again.

Start the VPN

At this point, you're ready to start the VPN. Do so using `systemctl`, the utility used to control services in Ubuntu, passing it your server's Common Name:

```
$ sudo systemctl start openvpn@server
```

Check the VPN's status:

```
$ sudo systemctl status openvpn@server
```

If it's working properly, the output should say `active (running)`.

Press Q to return to the terminal and then make the VPN start whenever the server boots:

```
$ sudo systemctl enable openvpn@server
```

Your VPN should now be up and running and ready for client connections.

Configure a VPN Client

Clients must have *.ovpn* files configured to connect to the VPN server and send and receive traffic across the secure tunnel. Creating these configurations can be tedious if you have several clients to connect, so we'll use an easily repeatable procedure to do it for us. We'll generate configuration files on the OpenVPN server and then transfer those configuration files to the relevant clients.

On your OpenVPN server, create a safe location for the client configuration files (such as */etc/openvpn/client-configs/files/*), copy another template provided by OpenVPN, and open the resulting *base.conf* file in a text editor:

```
$ sudo mkdir -p /etc/openvpn/client-configs/files/
$ sudo cp /usr/share/doc/openvpn/examples/sample-config-files/client.conf \
    /etc/openvpn/client-configs/base.conf
$ nano /etc/openvpn/client-configs/base.conf
```

Familiarize yourself with the file's contents. If you made changes to the port or protocol in previous steps, make the same changes in this file.

```
--snip--
;proto tcp
proto udp
--snip--
remote vpn_ip 1194
```

```
;remote vpn_ip 1194

--snip--
```

Also, uncomment the `user` and `group` directives:

```
--snip--

user nobody

group nogroup

--snip--
```

Comment out the SSL/TLS parameters:

```
--snip--

# ca ca.crt

# cert client.crt

# key client.key

--snip--
```

Comment out the `tls-auth` directive:

```
--snip--

# tls-auth ta.key 1

--snip--
```

Set the `cipher` and `auth` directives to the values found in the other configuration files:

```
--snip--
cipher AES-256-CBC
auth SHA256
--snip--
```

Finally, add the following line to the end of the file:

```
--snip--
key-direction 1
```

The `key-direction` directive indicates to the client which device in the client-server pair will provide the key and therefore the encryption for the VPN tunnel. This can be set to either `0` or `1`, but this configuration should be set to `1`, as this should provide better overall security by forcing different keys to be used for client-server and server-client communication. Save and close the file.

You can easily create client configurations by writing and executing a script to pull all of these elements together. Create an *.sh* file in which to put your script, make it executable, and then open it with a text editor (nano in this example):

```
$ sudo touch /etc/openvpn/client-configs/client_config.sh
$ sudo chmod +x /etc/openvpn/client-configs/client_config.sh
$ sudo nano /etc/openvpn/client-configs/client_config.sh
```

Copy the script in **Listing 5-1** into the file.

```
#!/bin/bash

KEY_DIR=/etc/openvpn/client-configs/keys

OUTPUT_DIR=/etc/openvpn/client-configs/files

BASE_CONFIG=/etc/openvpn/client-configs/base.conf

cat ${BASE_CONFIG} \

  <(echo -e '<ca>') ${KEY_DIR}/ca.crt \

  <(echo -e '</ca>\n<cert>') ${KEY_DIR}/${1}.crt \

  <(echo -e '</cert>\n<key>') ${KEY_DIR}/${1}.key \

  <(echo -e '</key>\n<tls-auth>') ${KEY_DIR}/ta.key \

  <(echo -e '</tls-auth>') > $ {OUTPUT_DIR}/${1}.ovpn
```

Listing 5-1*: A script for generating client configuration (*.ovpn*) file*

Save and close the file. The first line tells bash that what follows in this file is a script. The next three lines are variables, which you can modify if your key directory, output directory, or base config files and folders are different from the examples in this chapter.

Execute the script from within the *client-configs* directory as shown in Listing 5-2, with a client name as the only parameter. The client name should match one in the certificate and key files you created in earlier steps. To generate configuration files for further clients, be sure to generate their certificates and keys, and then use those files to create the relevant *.ovpn* file for that client with the script in Listing 5-1. Don't forget this entails creating a certificate request, transferring it to your certificate server for signing, and then transferring it back to your VPN server, in the *client-configs* directory.

Listing 5-2 shows a run of the script for the `myclient` client, and a command to list the resulting file.

```
$ cd /etc/openvpn/client-configs/

$ sudo ./client_config.sh myclient

$ ls -lah /etc/openvpn/client-configs/files/

total 20

drwxrwxr-x 2 test test 4096 Apr 28 23:22 ./

drwxrwxr-x 4 test test 4096 Apr 28 23:21 ../

-rw-rw-r-- 1 test test 11842 Apr 28 23:22 myclient .ovpn
```

Listing 5-2*: Executing the script from* ***Listing 5-1***

Once the *.ovpn* file is created for this client, download the file to your local machine via rsync and then import it into the OpenVPN client for that device.

```
$ rsync -ruhP user@vpn_ip: /etc/openvpn/client-
configs/files/ myclient .ovpn ./
```

OpenVPN has client applications for most operating systems, including Windows, Linux, macOS, iOS, and Android. You can find these on the OpenVPN website: ***https://openvpn.net/community-downloads/***.

With that done, you can now import the *.ovpn* configuration file, connect to your VPN, and use the internet in a much more private and secure manner. If you plan to connect to your VPN using a Linux client, you can install OpenVPN using the following command:

```
$ sudo apt install openvpn -y
```

Then, connect to your VPN using your configuration file and this command:

```
$ sudo openvpn myclient .ovpn
```

See "Test Your VPN" on page 89 for additional testing you can do to ensure your VPN is secure.

#20: Creating a VPN with Wireguard

Modern versions of Ubuntu (those from March 2020 onward) have Wireguard built into the kernel, so it's simple to install and get up and running. Wireguard isn't built into a lot of networking hardware at this stage, so you'll have to configure each of your endpoints to connect to it manually, rather than simply configuring your router and passing all network traffic through the VPN tunnel. In this project, you'll create a Wireguard server using the instructions to create a virtual machine in the cloud, and then you'll install and configure Wireguard. We'll create the relevant public and private key pairs for the server and any clients, configure the server firewall as required, configure and connect a client, and test the VPN to ensure that it's working correctly. Your internet traffic will then be safe and secure, as long as you're connected to your Wireguard VPN.

Installing Wireguard

Create a new Ubuntu server using the instructions provided in Project 3 in Chapter 1. Log in to the server via SSH as a standard, non-root user:

```
$ ssh user@your_server_IP
```

Then, use `apt` to install Wireguard, specifying `-y` to skip the confirmation prompt:

```
$ sudo apt install wireguard -y
```

Next, you'll create the necessary public and private keys required to connect to and encrypt your VPN.

Set Up the Key Pairs

Due to the sensitive nature of the files or folders you're about to create, it's wise to enforce more restrictive permissions than usual. You can run the following command to ensure that only the owner of a file can read and write to that file:

```
$ umask 077
```

This `umask` command won't last after you exit the terminal session, but only the owner is allowed to read and write to the folders and files you create during this session.

Now, using the `wg genkey` command, create the private Wireguard key:

```
$ wg genkey | sudo tee /etc/wireguard/private.key
```

The output shown in the terminal is your private key, which will be stored in the *private.key* file specified in the command. Do not share this key. Treat it like a password—it's how your VPN will be secured.

With the private key created, you'll need a corresponding public key to provide to your clients so they can authenticate to the server:

```
$ sudo cat /etc/wireguard/private.key | wg pubkey | sudo tee
/etc/wireguard/public.key
```

This command first reads the contents of the *private.key* file using `cat`. Then, the `wg pubkey` command uses the private key to generate the public key. The public key is then output to the terminal and saved to the *public.key* file.

Now that you have your public/private key pair, you can configure your VPN server and clients.

Configure Wireguard

Wireguard requires a configuration file to function. This file is not created when Wireguard is installed, so you need to create one from scratch. Create and open the */etc/wireguard/wg0.conf* file using a text editor:

```
$ sudo nano /etc/wireguard/wg0.conf
```

Add the following contents to the file:

```
[Interface]

PrivateKey = your_private_key

Address = 10.8.0.1/24

ListenPort = 26535

SaveConfig = true
```

Replace *your_private_key* with the private key you created earlier. Your key will be the contents of your */etc/wireguard/private.key* file. The address will be the address of your server within the subnet you want your VPN clients to be assigned when they connect to your server; ensure that this subnet is different from your private network. For example, if you use *192.168.1.x* addresses in your network, avoid using *192.168.1.x* addresses for your VPN. The listening port should be any port between 1025 and 65535, chosen at random. This port is the one your server and clients will use to communicate. Once complete, save and exit the configuration file.

At this point, the server's network settings require some modification. Configure IP forwarding so the VPN will forward the traffic it receives using the following command and then restart `sysctl` so that the changes take effect:

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

$ sudo sysctl -p
```

Next, you need to configure the firewall to allow VPN traffic to ingress and egress the server.

Configure the Firewall

In this section we'll discuss the use of the *Uncomplicated Firewall (UFW)*, Ubuntu's built-in firewall that is designed to reduce the complexity of firewall configuration. To configure the firewall, first identify the correct network interface for the VPN. Specifying the wrong interface will result in a nonfunctional VPN. Enter the following command to locate your server's default network interface:

```
$ ip route | grep -i default

default via 172.16.90.1 dev ens33 proto dhcp metric 100
```

In this output, the network interface is called `ens33` (yours might be different). The *default route* shown by `ip route` will be your host's public network interface. You'll need this to configure your firewall correctly.

Next, add the following rules to the bottom of your Wireguard configuration file by opening */etc/wireguard/wg0.conf* with a text editor again and replacing `ens33` with your network interface name:

```
$ sudo nano /etc/wireguard/wg0.conf
```

```
--snip--

SaveConfig = true

PostUp = ufw route allow in on wg0 out on ens33

PostUp = iptables -t nat -I POSTROUTING -o ens33 -j MASQUERADE

PreDown = ufw route delete allow in on wg0 out on ens33

PreDown = iptables -t nat -D POSTROUTING -o ens33 -j MASQUERADE
```

Save and close the file. This allows Wireguard to modify the firewall configuration after Wireguard starts and before it stops to enable the VPN to function correctly.

Additionally, you need to allow traffic via the listening port you configured earlier in the chapter (port 26535 in the example):

```
$ sudo ufw allow 26535/udp
```

Next, allow OpenSSH:

```
$ sudo ufw allow ssh
```

Finally, with this rule updated, you need to disable and enable UFW to reload the rules (your SSH session might be interrupted, and you may need to log in again):

```
$ sudo ufw disable

$ sudo ufw enable
```

And with that, your firewall configuration is complete.

Identify the DNS Server

To secure your internet traffic, your VPN needs to have correctly configured DNS to prevent DNS leaks, which can compromise your security. To solve this problem, you'll force your Wireguard VPN to use the DNS that's used by the Wireguard server itself. Identify that DNS server(s) with the following command:

```
$ resolvectl dns ens33
```

The resulting output is the DNS address you will provide to the client in its configuration later in this project—take note of it.

Start the VPN

Ideally, the VPN should start and be ready to accept client connections whenever the server starts up. You can achieve this by creating and starting a Wireguard system service using `systemctl`:

```
$ sudo systemctl enable wg-quick@wg0.service

$ sudo systemctl start wg-quick@wg0.service
```

Once done, check the status to ensure Wireguard is running:

```
$ sudo systemctl status wg-quick@wg0.service
```

If it's working properly, the output should say `active`. If the service is not active or has a failed status, double-check your configuration file and firewall status to ensure there are no typos or other errors in your configuration.

Configure a VPN Client

There are official client applications available for Wireguard for Windows, macOS, Android, and iOS—the setup of which is reasonably similar across the board. The Linux client setup is a little more involved, but if you've been able to configure the Wireguard server successfully, configuring a Linux client will seem very familiar.

Windows, macOS, Android, or iOS Client Configuration

To configure a client on any of these operating systems, follow these steps:

1. Download and install the relevant client program from *https://www.wireguard.com/install/*.
2. In the client interface, click + or **Add Tunnel** ▸ **Add Empty Tunnel** to create a new VPN profile from scratch.
3. Note that the public and private keys for the client are displayed.
4. Supply a friendly name in the Name field.
5. Ignore any On Demand settings or check boxes.
6. 6. Add the following details to the configuration, below the PrivateKey automatically generated for the client:

```
--snip--
Address = 10.8.0.2
DNS = 108.61.10.10
[Peer]
PublicKey = server_public_key
AllowedIPs = 0.0.0.0/0
Endpoint = server_public_ip:listening_port
```

`Address` is the IP address you want your client to have within the VPN subnet and should be different for every VPN client. `DNS` should be the IP address of the DNS server you identified in "**Identify the DNS Server**" on **page 85**. `PublicKey` is the public key you created for your Wireguard server earlier in the process. `AllowedIPs` is a setting used for *split tunneling*; traffic to and from the networks or addresses listed with this directive will be sent through the VPN tunnel, and all other traffic will go straight out and circumvent the VPN. Setting this to `0.0.0.0/0` sends all traffic from your client through the VPN. `Endpoint` is the public IP address of your VPN server, followed by the listening port you specified earlier (26535 in the example).

7. Save the configuration.
8. 8. On the Wireguard server, stop the Wireguard service, noting that there will be downtime for any users currently connected, using the following:

```
$ sudo systemctl stop wg-quick@wg0.service
```

9. 9. Open the */etc/wireguard/wg0.conf* configuration file with a text editor:

```
$ sudo nano /etc/wireguard/wg0.conf
```

10. 10. Add the client details to the bottom of the configuration file, keeping in mind that each peer you add will need its own [Peer] section added to this file:

```
--snip--
[Peer]
PublicKey = client_public_key
AllowedIPs = 10.8.0.2
```

This instance of `PublicKey` is the public key created for your Wireguard client by the client application. Within the `[Peer]` section of the file, `AllowedIPs` refers to the IP addresses allowed to send traffic through the VPN tunnel. Set this to the specific host IP you want your client to have on the VPN network, which must match the IP you configured for this peer in the client configuration.

11. 11. Save and close the file.
12. 12. Start the Wireguard service and double-check that the status says `active`:

```
$ sudo systemctl start wg-quick@wg0.service
$ sudo systemctl status wg-quick@wg0.service
```

Back on your client, activate the VPN connection. Once successfully connected, ping your Wireguard server's VPN address (such as 10.8.0.1):

```
$ ping 10.8.0.1
```

PING 10.8.0.1 (10.8.0.1): 56 data bytes

```
64 bytes from 10.8.0.1: icmp_seq=0 ttl=57 time=43.969 ms
64 bytes from 10.8.0.1: icmp_seq=0 ttl=57 time=43.969 ms
64 bytes from 10.8.0.1: icmp_seq=0 ttl=57 time=43.969 ms
64 bytes from 10.8.0.1: icmp_seq=0 ttl=57 time=43.969 ms
--- 10.8.0.1 ping statistics ---
4 packets transmitted, 4 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 43.969/43.969/43.969/0 ms
```

A successful result indicates your VPN connection is working between your client and server. Repeat this process for any additional clients.

Linux Client

To configure a Linux client, follow these steps:

1. 1. Install Wireguard and resolvconf (used for DNS configuration):

```
$ sudo apt install wireguard resolvconf -y
```

2. 2. Generate the client public/private key pair for the client:

```
$ wg genkey | sudo tee /etc/wireguard/private.key
$ sudo cat /etc/wireguard/private.key | wg pubkey | sudo tee \
    /etc/wireguard/public.key
```

3. 3. Create the Wireguard client configuration file:

```
$ sudo nano /etc/wireguard/wg0.conf
[Interface]
PrivateKey = client_private_key
Address = 10.8.0.3
DNS = 108.61.10.10
[Peer]
PublicKey = server_public_key
AllowedIPs = 0.0.0.0/0
Endpoint = server_public_ip:listening_port
```

4. Save and close the file.

5. 5. On the Wireguard server, stop the Wireguard service:

```
$ sudo systemctl stop wg-quick@wg0.service
```

6. 6. Open the */etc/wireguard/wg0.conf* configuration file with a text editor:

```
$ sudo nano /etc/wireguard/wg0.conf
```

7. 7. Add the client details to the bottom of the configuration file:

```
--snip--
[Peer]
PublicKey = client_public_key
AllowedIPs = 10.8.0.3
```

This instance of `PublicKey` is the public key created for your Wireguard client by the client application. Within the `[Peer]` section of the file, `AllowedIPs` refers to the IP addresses allowed to send traffic through the VPN tunnel. Set this to the specific host IP you want your client to have on the VPN network.

8. Save and close the file.

9. 9. Start the Wireguard service and double-check that the status says active:

```
$ sudo systemctl start wg-quick@wg0.service
$ sudo systemctl status wg-quick@wg0.service
```

Back on your client, activate the VPN connection using the following command:

```
$ wg-quick up wg0
```

Once successfully connected, ping your Wireguard server's VPN address (such as 10.8.0.1). A successful result indicates your VPN connection is working between your client and server. To disconnect a Linux client from your VPN server, use the following command:

```
$ wg-quick down wg0
```

Repeat this process for any additional clients you want to add.

Test Your VPN

Regardless of which VPN you chose, find your public IP address from a website such as *https://www.whatismyip.com/* while not connected to the VPN. Once done, connect to your VPN and refresh the page. Your public IP address should now be the IP address of your VPN server.

Another way to test your VPN is to use a service such as DNS leak at *https://dnsleaktest.com/*. Performing a standard test should show you clearly whether there are any issues with your VPN configuration. If your actual public IP is masked and the DNS leak test shows only the DNS servers you've configured the VPN to use, then you've been successful in setting up your own private VPN server.

Summary

Connecting multiple clients to either your OpenVPN or Wireguard servers will allow traffic to pass between them as if they were on the same network. This means you can easily manage multiple devices remotely simply by having them all connected to your VPN at the same time. This chapter covered setting up your own private VPN, which provides you with complete control, using either OpenVPN or the much lighter and faster Wireguard. Your private internet traffic will now be truly private and secure while you are connected to your VPN.