Virtual Private Network (VPN)

COSC301 Laboratory Manual

Over recent years the nature of the internet has changed. In the early days there was an assumption that all the traffic was trusted. This is definitely not the case these days. There is a move towards encrypting traffic, whether it is to/from various services (the 's' variants of the protocols, for example https, smtps, etc.). In this lab we explore configuring a VPN. VPNs are used to grant access to internal resources to a client connecting over an insecure network.

You have seen the term VLAN before now. VLANs are used to segment portions of the network into logical (or broadcast) domains. For example, within a company, there may be a logical network for the engineering department that is separate from the marketing department. Yet they both share the same network infrastructure (i.e. the switches and routers of the organisation).

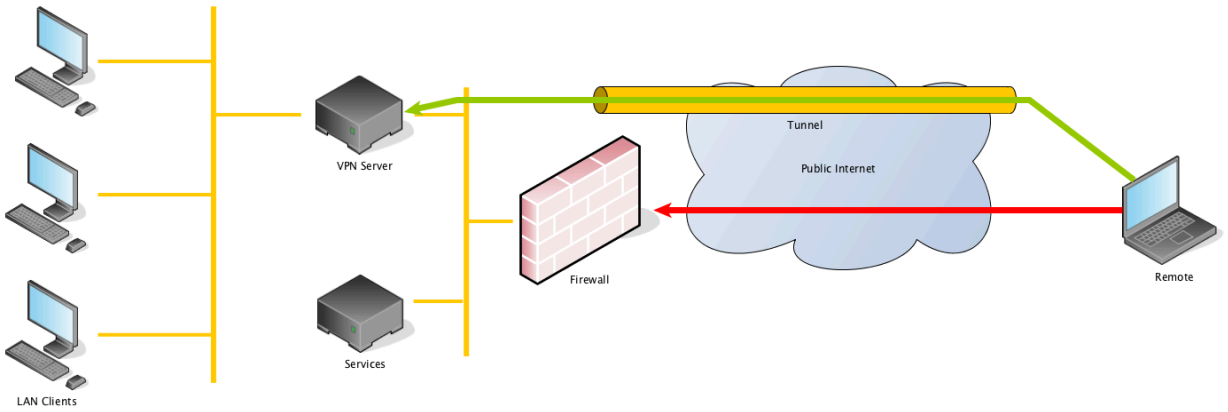
VPNs on the other hand are generally used to link two (or more) networks[1] together over an insecure channel. For example, a remote worker might need access to some internal resource (like a shared file server) from their home over the public internet.

To understand the difference, let's have a brief look at what happens to the packets as they travel across the VPN. Normally, the client addresses the packet to the destination and routers along the way will pass the packet closer to the destination. In a VPN, this packet is packaged inside another that's directed to the VPN server. You can think of it as if you take a letter and envelope, addressed to Bob, and place it inside another one addressed to Carol. You trust Carol to unwrap her envelop and then pass the contained letter to Bob. Note the contained packet (or letter) is encrypted by VPN.

In this lab we will simulate a remote worker using OpenVPN to connect to Server1 (their trusted home server) so that they can ping to Client1 using it's internal address (i.e. 192.168.1.11). OpenVPN uses port number 1194. The firewall would have to open that port in order for Server1 to provide the VPN service. If you would like more information about OpenVPN, have a look at the OpenVPN Home Page [https://openvpn.net], specifically, the Community Pages [https://openvpn.net/index.php/open-source.html][2].

[1] We can connect both whole networks (at the router level) as well as individual client computers.
[2] As is common trying to earn money from open source, they provide a preconfigured services and support for a fee.

Figure 1. Remote Client VPN setup

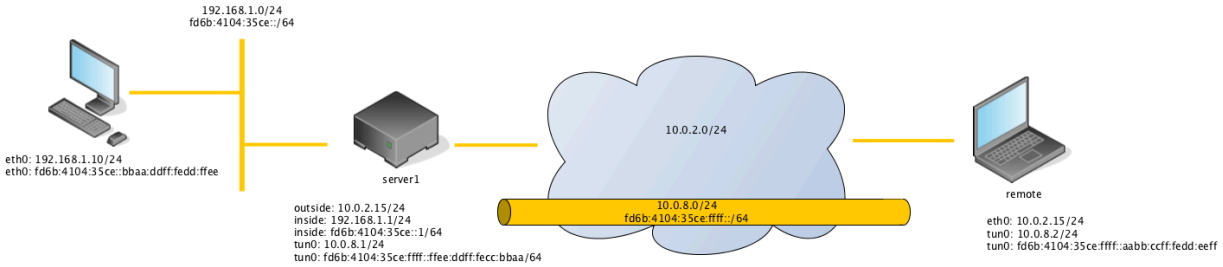


The above figure shows how the VPN works when used across the internet. Access to services have been blocked by firewall rules and the remote client, therefore, cannot access them. By tunneling the network traffic across an encrypted connection the remote client is able to access the services as if they were on one of the LAN Clients.

1. Configure VirtualBox with the Topology

> **Warning**
> You should take a snapshot of Server1 in order to undo these changes for future labs.

Figure 2. Interior Routing Network Topology



The above figure, Figure 2, “Interior Routing Network Topology”, shows the (eventual) network setup we're going to achieve in this lab. At each end of the tunnel (the yellow tube) a virtual device (tap0) is created automatically once the VPN is set up. It shows the topology at layer 3 (the network layer or the IP layer) and also at layer 1 (the physical layer, i.e., the cables), to help you to appreciate how the devices would physically connect to each other. We'll explain more details later in the lab.

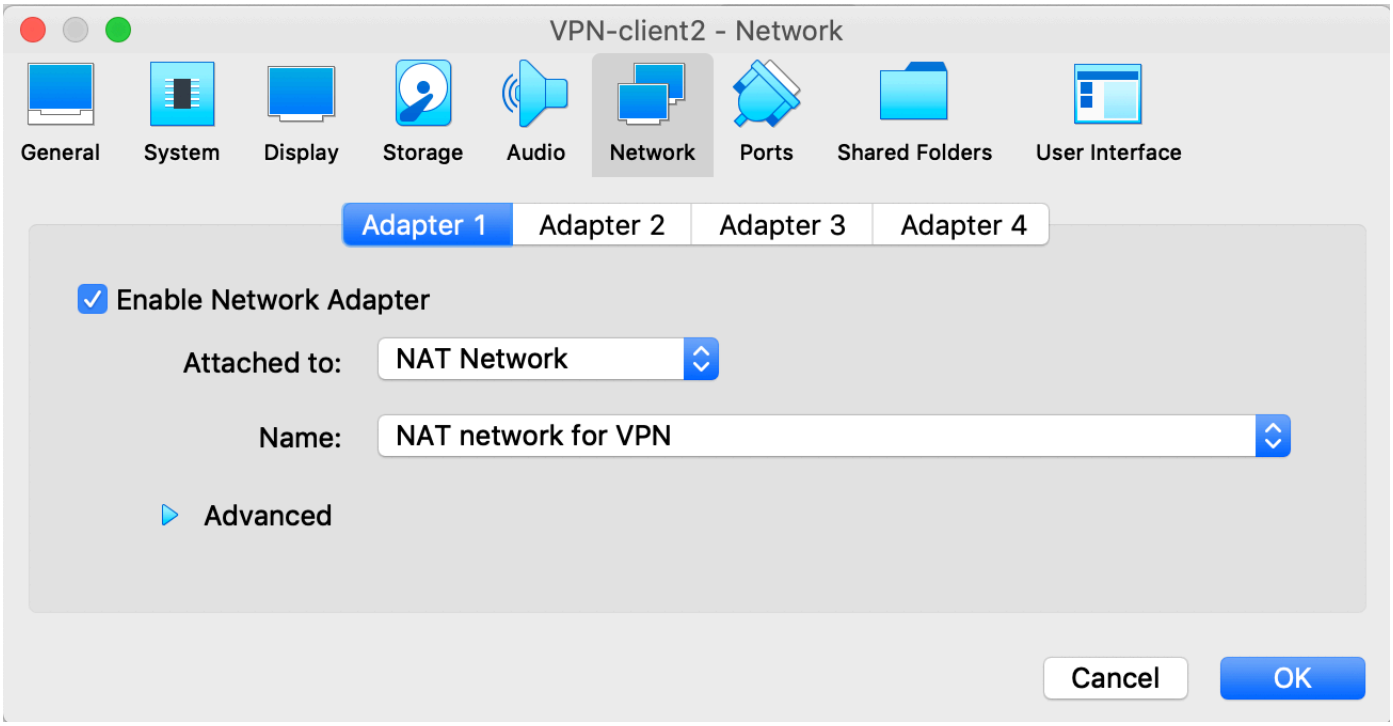
In this section you will be using VirtualBox to create, configure and connect the devices in the network:

1. You will create a temporary virtual machine for the remote host (like you did before for Client2), connecting it and server1 appropriately to a NAT Network. This defines the connection at the physical layer (layer 1).

2. The instructions of setting a NAT network for both Server1 and Remote are given in Figure 3, “NAT Network settings”.

3. After booting the devices in the network, you will configure the software inside them to build the network layer connection (layer 3).

Figure 3. NAT Network settings



The above figure, Figure 3, “NAT Network settings”, shows the NAT Network settings used for this lab. Make sure both Server1 and Remote use the same name for the NAT network, e.g., `NAT network for VPN`. A NAT network allows guests to connect to each other via the internet. It is as if both the server and the remote client were directly connected to the internet. This means we don't need to worry about setting up NAT ourselves for Internet connection.

Since both Server1 and Remote will need to be connected to a NAT network as opposed to just plain NAT[3], as we did before, we need to reconfigure their adaptors.

Open VirtualBox preferences and navigate to the Network tab. Change Server1's Adapter 1 so that it's attached to a NAT network as shown in Figure 3, “NAT Network settings”. Do the same to Remote.

Boot into server1 and you should still be able to access the internet. Test this by installing the openvpn and easy-rsa packages using **sudo apt-get install openvpn easy-rsa**.

While server1 is installing packages, if you haven't done so yet, start Client2 as the Remote, booting from the live CD. This is just going to be a temporary machine that we will use to test that we've configured the VPN correctly and so won't need a hard disk. Make sure that Remote's Adapter 1 is connected to the same NAT Network.

You need to:

[3]VirtualBox doesn't allow Virtual Machines on NAT to see one another by design.

1. **Screenshot**
   Make sure Server1's outside interface is connected to the NAT network and show the IP address of the outside interface in a screenshot (we will need this later when connecting to the VPN from Remote).

2. **Screenshot**
   Show the IP address of Remote's Ethernet interface in a screenshot.

3. **Screenshot**
   Finally, a screenshot showing Server1 and Remote can ping each other

2. Public-Key Infrastructure (PKI)

Central to the public key infrastructure is the idea of a certificate. Open your browser and go to any SSL enabled website (such as Google [https://www.google.com]) and click on the padlock symbol in the address bar and show the certificate. In safari (and I'm sure other browsers) will show the chain of trust along with the details of the certificate. Expand the details and peruse through and note any fields of interest.

Some important terms in the process:

Server
: The server is where the OpenVPN connections terminate. It has generated a certificate (called the 'issued certificate') that has been signed by a certificate authority.

Client
: The client is the OpenVPN program running on the remote machine. It must have access to the certificate of the certificate authority used to sign the server's issued certificate.

User
: The user of the machine, knows their username and password.

When connecting to an OpenVPN server, both client and server mutually authenticate. The purpose of this is to ensure that the client it connecting to the correct server and vice versa. It is a similar process when SSH connections are initiated or when you visit SSL enabled websites.

Once the connection has been started the server presents its issued certificate to the client. The client can then verify that the server is the one its claiming to be. If the client determines that the server is lying then the client will terminate the connection. The other way the client will terminate the connection is if the server's certificate has been revoked (the location of a 'certificate revocation list' is included as part of the certificate authority's certificate).

If the client has a set of certificates then it presents them to the server. These certificates are optional and depend on the setup of the client. The server then makes the same decisions about the client's certificates, it checks that: they're signed by the same certificate authority (if not, the connection is terminated); the client certificate is not on the revocation list (otherwise the connection is terminated); the certificate is valid (hasn't expired and is after the issue date).

If the client is configured to send user credentials to the server then it does so now. Once the server has received the credentials the server checks that they're valid. Often this is performed by a separate program or script. In our case we will be using Linux's Pluggable Authentication Modules (PAM) which we will setup later on. If the checks fail, then the connection is terminated.

Once we have authenticated, the configuration is exchanged and the tunnel is brought up.

An important part of any VPN is the authentication of remote clients. There are several methods that can be used to authenticate a client to the server. In order from easiest (and least secure) to more complex (and more secure) they are:

- No authentication
- Username/password
- Client/Server certificates

Self-assessment

1. While it is a small distraction to talk about web certificates, I'd like you think about how the web browsers manage the certificate authorities. How do they end up in your browser? Who makes the decision? How can someone get their certificate authority trusted in the browser? What happens if the authority mis-issues some certificates?

2. Write some brief notes on the advantages and disadvantages of each of the authentication methods described above (and any others you may be able to find). To focus the notes, think about why the shared secret is bad, and why the client/server certificates are good. Is there a better method?

3. In the next section we are going to setup our own public key infrastructure. Before we start, briefly define the following PKI-related terms:

- Certificate Store
- Certificate Authority (CA)
- Registration Authority
- Central Directory
- Certificate Management System
- Certificate Policy
- x509 Certificate
- Public Key
- Private Key

3. Server Certificates

Now that you understand the role PKI plays, we need to setup our own so that we can issue certificates as needed. By the end of this section we will have created our own certificate authority, along with a public/private key pair that clients will use to identify the server.

Caution

At this point it's worth noting that each of the certificates created below will be cryptographically unique. This means that (like the SSH keys you would have created in an earlier lab) there is no way to recover a public key from a private one (and vice versa). Once you start signing certificates with one CA, you can't suddenly switch to another. You will have to start again from this point.

The easy-rsa package, that we installed previously, contains a set of scripts that do a lot of the heavy lifting. Run **make-cadir ~/openvpn-ca** and change into the directory `~/openvpn-ca`. This is our working directory and makes it easier to keep the generated files separated other (personal) files.

You will see various scripts and some configuration files. The `vars` file sets up various variables used during the process of certificate generation. We have included some annotations in the listing below. You will need to edit `vars` as suggested below.

```
…
Take note of this warning!
# WARNING: clean-all will do
# a rm -rf on this directory
# so make sure you define
# it correctly!
export KEY_DIR="$EASY_RSA/keys"

# Issue rm -rf warning
echo NOTE: If you run ./clean-all, it will be doing a rm -rf on $KEY_DIR

The following is an old warning.
# Increase this to 2048 if you
# are paranoid.  This will slow
# down TLS negotiation performance
# as well as the one-time DH parms
# generation process.
export KEY_SIZE=2048

# In how many days should the root CA key expire?
export CA_EXPIRE=3650 This is 10 years

# In how many days should certificates expire?
export KEY_EXPIRE=3650 For the certificates generated by these scripts -- again 10 years

# These are the default values for fields
# which will be placed in the certificate.
# Don't leave any of these fields blank.
export KEY_COUNTRY="US" Change this to "NZ"
export KEY_PROVINCE="CA" Change this to "Otago"
export KEY_CITY="SanFrancisco" Change this to "Dunedin"
export KEY_ORG="Fort-Funston" This is the organisation -- like Google, Otago University, etc.
export KEY_EMAIL="me@myhost.mydomain" In the real-world you should use a legitimate address.
export KEY_OU="MyOrganizationalUnit" Something like the Help Desk, or Research and Development.
…
```

Tip

The note about KEY_SIZE in the listing above refers to 'DH' -- Diffie-Hellman. If you're interested in how two people can establish a shared secret (to use for encryption) over an insecure channel (such as the internet), have a look at the Diffie-Hellman key exchange.

> **Tip**
>
> You may be wondering why we need to set parameters for ORG and OU and all that stuff. Well, its related to the ideas of centralised account management as used in large organisations.

Now that we've configured the certificate variables, we need to generate the certificate authority. Run the following commands:

```
mal@server1:~/openvpn-ca$ ln -s openssl-1.0.0.cnf openssl.cnf
mal@server1:~/openvpn-ca$ source vars
NOTE: If you run ./clean-all, it will be doing a rm -rf on /home/mal/openvpn-ca/keys
mal@server1:~/openvpn-ca$ ./clean-all
mal@server1:~/openvpn-ca$ ./build-ca
Generating a 2048 bit RSA private key
...
writing new private key to 'ca.key' This should be kept private and secure.
-----
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 as 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.
-----

These values below are obtained from the vars file we edited before, hit enter to accept the
value in square brackets, or change them here.

Country Name (2 letter code) [NZ]:
State or Province Name (full name) [Otago]:
Locality Name (eg, city) [Dunedin]:
Organizational Name (eg, company) [...]:
Organizational Unit Name (eg, section [...]:
Common Name (eg, your name or your server's hostname) [...]:
Name [EasyRSA]:
Email Address [...]:
```

Now that we've generated the certificate authority, we need to create a public/private key pair that the clients use to authenticate the server.

> **Important**
>
> The following command is different to the previous one -- take care!

```
mal@server1:~/openvpn-ca$ ./build-key-server server1
Generating a 2048 bit RSA private key
...
writing new private key to 'server1.key' This should be kept private and secure.
-----
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 as 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.
-----

These values below are obtained from the vars file we edited before, hit enter to accept the
value in square brackets, or change them here.
```

```
Country Name (2 letter code) [NZ]:
State or Province Name (full name) [Otago]:
Locality Name (eg, city) [Dunedin]:
Organizational Name (eg, company) [...]:
Organizational Unit Name (eg, section [...]:
Common Name (eg, your name or your server's hostname) [server1]:
Name [EasyRSA]:
Email Address [...]:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []: Leave this blank
An optional company name []: Leave this blank
Using configuration from /home/mal/openvpn-ca/openssl-1.0.0.cnf
Check that the request matche the signature
Signature ok
The Subjects Distinguished Name is as follows:
countryName           :PRINTABLE:'NZ'
stateOrProvinceName   :PRINTABLE:'OTAGO'
localityName          :PRINTABLE:'Dunedin'
organizationalName    :PRINTABLE:'...'
organizationalUnitName:PRINTABLE:'...'
commonName            :PRINTABLE:'server1'
name                  :PRINTABLE:'EasyRSA'
emailAddress          :IA5STRING:'...'
Certificate is to be certified until Jan 16 22:20:16 2028 GMT (3560 days)
Sign the certificate? [y/n]: y

1 out of 1 certificate request certified, commit? [y/n] y
Write out database with 1 new entries
Data Base Updated
```

Now we generate the DH parameters which will be used to generate a shared secret for secure communication between the server and a client after the authentication process is successful.

```
mal@server1:~/openvpn-ca$ ./build-dh
Generating DH parameters, 2048 but long safe print, generator 2
This is going to take a long time
...
```

You should now have the following files which are needed by the OpenVPN daemon.

ca.crt
: The certificate authority's certificate, contains the public key of the CA.

server1.key
: The private key used by the server.

server1.crt
: The certificate used by the server, contains server1's public key and a digital signature from the CA.

dh2048.pem
: The Diffie-Hellman exchange parameters.

Now we have all the certificates and other paraphernalia that we need in order to setup the VPN. The final task is to copy the keys to the `/etc/openvpn/` directory. First **cd keys**, then run **sudo cp ca.crt server1.key server1.crt dh2048.pem /etc/openvpn/**.

Finally, generate a secret key for extra security of the communication between Server1 and Remote.

```
mal@server1:~/openvpn-ca$ sudo openvpn --genkey --secret /etc/openvpn/ta.key
```

Over the next few sections, we are going to walk through the following steps to incrementally build the VPN.

4. Initial Server Configuration

During this process we are going to use a separate network 10.8.0.0/24 for the remote clients to use as VPN. This is deliberately chosen to be different to the existing LAN network. The main reason is that it helps to keep the different networks logically separate (and it makes it more obvious where the traffic is flowing). When we come to the IPv6 addresses we will, again, use a separate network (we used https://www.ultratools.com/tools/rangeGenerator, and set the Global ID to '6b410435ce' and Subnet ID to 'ffff' -- to make it as visually different from the existing network as possible.).

The networks used by the server *must* be different to the network that the remote client is connecting through. If it's not, then the routing won't work properly. This is often a cause of conflicts and issues.

Use **gunzip -c /usr/share/doc/openvpn/examples/sample-config-files/server.conf.gz | sudo tee /etc/openvpn/server.conf** to copy the sample `server.conf` provided by the maintainers of openvpn. Read through the file and edit/add/adjust the file to match the options below leaving the other options at their default value. (We have removed the comments for brevity---you should be sure to understand what the options are doing).

```
port 1194
proto udp

dev tun

ca ca.crt
cert server1.crt
key server1.key
dh dh2048.pem

server 10.8.0.0 255.255.255.0 See the note below.

keepalive 10 120

tls-auth ta.key 0

cipher AES-256-CBC

This is where we're configuring the username/password authentication options.
verify-client-cert none
plugin /usr/lib/x86_64-linux-gnu/openvpn/plugins/openvpn-plugin-auth-pam.so login
```

port ...
: The port to listen on, 1194 is the default for OpenVPN.

proto ...
: The protocol to use to encapsulate the network traffic. It doesn't matter that the traffic going across the VPN is UDP or TCP.

dev ...
The virtual network device to use for the VPN tunnel. It can be one of tap or tun, which one depends on whether you want to bridge or route the VPN to the LAN respectively. (Or if the VPN needs to handle non-IP traffic -- in which case you need to use tap).

For our use case we are setting up a separate, routed, network with only IP-based traffic, hence the use of tun.

ca, cert, key, dh
The PKI files that we created previously.

server ...
The network addresses of the VPN that the server is going to create. Because we're setting up a routed network, we need to use a different network to any of the existing connected networks (viz. something that's not 192.168.1.0/24 or 10.0.2.0/24). The value here is the default value for this option.

keepalive 10 120
This is a shortcut to specify two options and the behaviour differs a little depending on where OpenVPN is running. For the server, it'll send a ping if there's been no activity for 10 seconds, and if it fails to receive a response in 120 seconds restart the connection[4].

tls-auth ta.key 0
This is an extra security beyond that provided by SSL/TLS. It can help block DoS attack and UDP port flooding. The server and each client must have a copy of this key. This file is secret and read-only by the owner. The second parameter should be 0 on the server and 1 on the client (Remote).

cipher AES-256-CBC
This is the selected cryptographic cipher. This option must be specified in the **openvpn** command on the client side or in the client config file.

verify-client-cert none
This makes the client certificates optional as we are only authenticating by username/password, and don't want to authenticate the client too. To make use of this, the client's certificates would need to be signed by the same CA (in the same manner as the server certs were).

plugin ...
This tells OpenVPN to use PAM to perform the username/password checking. Briefly, PAM allows other applications to check authentication against various options. In this case, we use the 'login' option with the options supplied to PAM from the OpenVPN client.

Bridging vs Routing

Bridging is where the LAN and VPN clients share the broadcast domain, and would allow the existing DHCP server to provide addresses to the VPN clients. The other consideration on this front is if there are any services that require neighbour discovery (some features of Windows servers need this).

Routing on the other hand allows you to segment the VPN traffic from the LAN and makes management of firewall rules easier. We shall see an example of this in the Firewalling lab.

[4] In the documentation, they use the example of keepalive 10 60, and say the restart happens in 120 seconds -- maybe there's a typo.

Tap vs Tun, performance issues

Tap devices: typically behave like a real network adapter (despite it being virtual); can transport any network protocol; work in layer 2 (so ethernet frames are passed over the VPN). However, they cause more broadcast traffic across the VPN; add ethernet frame overhead; suffers from poor scalability; and cannot be used with Android or iOS devices.

Tun devices: have lower overhead, only transports layer 3 traffic (IP). However, broadcast traffic is not transported; older versions of OpenVPN lacked support for IPv6; and cannot be used in bridges.

We're now ready to start the service using `systemd`, **systemctl start openvpn@server.service**. This looks a little different from other **service start** commands you would have seen before. We can have multiple VPN services defined each with their own configuration file. This command uses the token after the '@' to figure out which configuration file to use when starting the service. Check `/var/log/syslog` to make sure that the service started properly.

Screenshot

Take a screenshot showing the log information or `status` of the `openvpn@server.service` VPN service.

Once you've successfully started the service, have a look at the output of **ip addr show** and **route -n**. You should see something resembling the following. Note that a new gateway 10.8.0.2 is generated for the VPN in addition to the server's address 10.8.0.1.

```
mal@server1:~$ ip addr show
1: lo:
    ...
2: outside:
    ...
3: inside:
    ...
5: tun0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast ↵
  state UNKNOWN group default qlen 100
    link/none
    inet 10.8.0.1 peer 10.8.0.2/32 scope global tun0
    valid_lft forever preferred_lft forever
mal@server1:~$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.0.2.1        0.0.0.0         UG    0      0        0 outside
10.0.2.0        0.0.0.0         255.255.255.0   U     0      0        0 outside
10.8.0.0        10.8.0.2        255.255.255.0   UG    0      0        0 tun0 This is new...
10.8.0.2        0.0.0.0         255.255.255.255 UH    0      0        0 tun0 ... as is this.
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 inside
```

We now need to connect the remote client. For that, the remote client needs to have a copy of the `ca.crt`. On Remote, use SCP to copy the file across **scp mal@10.0.2.15:/etc/openvpn/ca.crt ~/.**. This copies the certificate of the certificate authority (yes, this sounds strange but correct and precise) at `/etc/openvpn/ca.crt` from Server1 to our home directory (preserving the filename).

We need also copy `ta.key` from Server1 to Remote using **scp**. Since `ta.key` is read-only by root, we will need to work out a way to copy the file to Remote. Once it is copied to Remote,

the file should be changed back to read-only by the owner. This problem is left for you to solve by yourself.

We've now got all the information the remote client needs to be able to connect to the vpn. Install the `openvpn` package on Remote and connect the VPN using **sudo openvpn --remote 10.0.2.15 --dev tun --auth-user-pass --ca ca.crt --client --remote-cert-tls server --tls-auth ta.key 1 --cipher AES-256-CBC --auth-nocache** under the home directory where `ca.crt` and `ta.key` are copied. You should see the status of the client's connection in the terminal. Use a different terminal to do the assessment below.

Assessment

1. What address ranges do you expect to ping when the VPN is connected? What subnets can you actually reach? Is this different from what you expected? Why or why not?

Screenshot

Take a screenshot showing which hosts you can reach and which you cannot.

2. Examine the routing table on the client before and after connecting to the VPN. What new routes are added? What do you notice about the metrics? Why do you suppose this is?

3. Run **sudo tcpdump -i enp0s3** on Remote. This examines the traffic on the ethernet interface and emulates someone performing a man-in-the-middle (MITM) attack. Run the following two commands on Server1 to **ping** Remote, and compare and contrast the tcpdump output.

- ping -c 1 `10.8.0.6`
- ping -c 1 `10.0.2.4`

What implications does this hold for (free) public VPN servers?

5. Routing VPN Traffic (optional)

At the end of the previous step, the remote client can only ping server1's VPN address. In this section we're going to adjust the configuration so that the remote clients traffic is routed properly.

We need to make a couple of small modifications to `/etc/openvpn/server.conf`, these are presented below. As previously, modify the configuration to take these tweaks into account.

```
push "redirect-gateway def1"
script-security 2
learn-address "/etc/openvpn/learn-address"
```

push ...
This tells the server to give any of the connected clients that parameter. In this case it tells the clients that all their traffic should be sent through the vpn.

script-security ...

learn-address ...
: While the push directive can get the traffic from the VPN onto the LAN, we need to be able to get the LAN traffic onto the VPN. I'll discuss this in more detail below.

This works well to get the VPN traffic to the LAN clients, but the LAN clients don't know how to respond to the traffic from the VPN --- there is no route. The`learn-address ...` script handles this case for us, so that the server acts as a proxy for the vpn clients.

You should be familiar with ARP (or Neighbour Discovery), but for the sake of completeness, I'll describe the process here. When a LAN client wants to send a packet to a vpn client, it performs its neighbour discovery protocol (ARP in IPv4), where it asks "who has 10.8.0.1?". Because server1 is acting as a proxy, server1 will respond with its MAC address, and eventually route the traffic through the vpn to the correct host.

Now that we know what needs to happen with the `learn-address ...` stanza. We have created a simple script to add the proxy-ing. This script should be placed in `/etc/openvpn/learn-address`, owned by root, and executable.

```
#!/bin/sh
action="$1"
addr="$2"

logger "learning: $action $addr" Show something in the syslog

case "$action" in
 add | update)
  ip neigh replace proxy "$addr" dev inside
  ;;
 delete)
  ip neigh del proxy "$addr" dev inside
  ;;
esac
```

This script is responsible for adding and removing routes for each of the remote devices when they connect and disconnect respectively[5]. The script adds/removes clients individually, we could have set it up so that the whole VPN subnet was routed to server 1, and we wouldn't have needed this script, however we wanted to demonstrate this feature.

Disconnect remote1, restart the service (checking syslog to make sure it started properly) and reconnect remote1. If you cannot connect, check the syslog on server1 to see if there are any problems with the service.

Screenshot

Take a screenshot showing which hosts you can reach and which you cannot.

6. IPv6 Additions (optional)

Now that we have a functioning IPv4 VPN, we need to setup the same for IPv6. We need to make a couple of small modifications to `/etc/openvpn/server.conf`, these are presented below. As previously, modify the configuration to take these tweaks into account. Disconnect the remote client, restart the service (checking the logs) and connect the remote client again.

```
tun-ipv6
push tun-ipv6
```

[5] Fortunately it requires no changes to work with IPv6!

```
server-ipv6 fd6b:4104:35ce:ffff::/64
push "route-ipv6 fd6b:4104:35ce:0::/64" We have added the '0' after the 35ce to make it clear that we're routing
```

> **Screenshot**
> Take a screenshot showing which hosts you can reach and which you cannot.

1. Run **tcpdump** on remote1's `enp0s3` interface. Do you see any IPv6 traffic when pinging server1's IPv6 address? Is this what you would expect? What's happening?

7. Client-side Configuration and DNS (optional)

On remote, we've been using the command line to manage the starting and stopping of the OpenVPN client. It's time we move this to a configuration file.

Configuration File

```
client
dev tun
proto udp

comp-lzo

remote 10.0.2.15 1194 This is the public address of the server

nobind
persist-key
persist-tun
ca ca.crt

auth-user-pass

The following lines are to help sort out DNS---which we'll do next.

resolv-retry infinite
script-security 2
up "/etc/openvpn/update-resolv-conf"
down "/etc/openvpn/update-resolv-conf"
```

> **Screenshot**
> Take a screenshot showing which hosts you can reach and which you cannot.

This section is optional. It is here so we can get make sure that the services are setup properly. If you have completed the DNS Lab, we strongly suggest that you complete these steps.

In the DNS lab you created an access list that restricted DNS queries to the local networks. Because we have created two new subnetworks (one for each IPv4 and IPv6), we need to allow queries from these hosts.

Edit the `/etc/bind/named.conf.options`, by creating a new ACL and add it to the `allow-query` and `allow-recursion`stanzas. Restart the `bind9` service in the usual way.

```
acl "clients" {
...
```

```
}
acl "vpn" {
 10.8.0.0/24;
 fd6b:4104:35ce:ffff::/64;
};
options {
 ...
 allow-query { "clients"; "vpn"; };
 allow-recursion { "clients"; "vpn"; };
 ...
};
```

Now the final remaining step is to push the dns server to the vpn clients. Edit `/etc/openvpn/server.conf` to include the `push "dhcp-option DNS 192.168.1.1"`. Disconnect remote, restart the service (check the logs), and reconnect the remote.

You should be able to resolve all the local (internal) DNS names we defined previously, as well as check that they're reachable via ping.

Screenshot

Take a screenshot showing that you can resolve the internal DNS names from the remote client. Take another screenshot to show that you can ping (by name) server1.

8. Final Words

In this lab we have setup and configured a VPN which allows remote clients to appear as if they were on the local LAN. This is a powerful (and useful) tool that is used to ensure the integrity of private data being transferred across an untrusted network.

We've been using username/password authentication to allow clients access. As an optional exercise, you could extend this by using the PKI infrastructure we setup at the start to generate certificates for the clients.