-

Introduction

Most of this app note has been directly imported from the following link:

https://www.digitalocean.com/community/tutorials/how-to-set-up-an-openvpn-server-on-ubuntu-18-04

we have made several edits to account for Flexy specific settings.

Green $ means it is on the OpenVPN server
Red $ means it is on the Certificate Authority Machine
Light blue $ means it is a command useful for debugging issues.

The stuff from digital ocean was written by Mark Drake 

Note: for easy navigation use the navigate by headings tool in available in Microsoft word.

Prerequisites

To complete this tutorial, you will need access to an Ubuntu 18.04 server to host your OpenVPN service. You will need to configure a non-root user with `sudo` privileges before you start this guide. You can follow our Ubuntu 18.04 initial server setup guide to set up a user with

appropriate permissions. The linked tutorial will also set up a firewall, which is assumed to be in place throughout this guide.
Additionally, you will need a separate machine to serve as your certificate authority (CA). While it's technically possible to use your OpenVPN server or your local machine as your CA, this is not recommended as it opens up your VPN to some security vulnerabilities. Per the official OpenVPN documentation, you should place your CA on a standalone machine that's dedicated to importing and signing certificate requests. For this reason, this guide assumes that your CA is on a separate Ubuntu 18.04 server that also has a non-root user with `sudo` privileges and a basic firewall.
Please note that if you disable password authentication while configuring these servers, you may run into difficulties when transferring files between them later on in this guide. To resolve this issue, you could re-enable password authentication on each server. Alternatively, you could generate an SSH keypair for each server, then add the OpenVPN server's public SSH key to the CA machine's `authorized_keys` file and vice versa. See How to Set Up SSH Keys on Ubuntu 18.04 for instructions on how to perform either of these solutions.
When you have these prerequisites in place, you can move on to Step 1 of this tutorial.

Step 0 — Environment Setup

A VirtualBox box will be used to create Ubuntu Server VM. Download VirtualBox from Oracle. Download the Ubuntu Server LTS 18.04 iso. The default VirtualBox settings can be used with the following exceptions.

1) Settings > Network (Also the VM to get a IP address independent of the host)
    a. Ensure Adapter 1 is enabled
    b. Select Attached to: Bridged Adapter
    c. Select the host network interface with access to the internet
2) Settings > Shared Folders (Also the user to pass files back and forth between the host)
    a. Add a shared folder on the host (example Folder Path: c:\temp)
    b. Folder Name is VMshared
    c. Read-only and Auto-mount are no not checked
    d. Mount point is blank
    e. Check Make Permanent
3) Download the VBoxGestAdditions_6.0.0_RC1.iso from this site http://download.virtualbox.org/virtualbox/6.0.0_RC1/
4) Install the .iso in the Storage device for the VM (this is done in Settings > Storage)

Start the VM. Run the command ifconfig to verify the network adapter settings resulted in a IP address being assigned to the VM. The output should show a network interface with an IP address that is different from the Host but on the same subnet.

```
$ ifconfig
```

The following commands must be run at the Linux command line to mount the shared folder in the VM.

```
$ sudo mkdir /mnt/cdrom
```

```
$ sudo mount /dev/cdrom /mnt/cdrom
```

```
$ sudo /mnt/cdrom/./VBoxLinuxAdditions.run
$ mkdir ~/VMshared
$ sudo mount -t vboxsf -t vboxsf VMshared ~/VMshared
$ touch ~/VMshared/test
```

Check the host machine to verify a file “test” is now visible in the shared directory. This directory is used to pass files between the Linux VM and the host machine. The mount command will need to be executed every time the VM is started. The edit the following files to mount the shared directory on boot up.

Edit the file /etc/fstab

Add this line using tabs to separate the zeros:

VMshared /home/<username>/VMshared vboxsf defaults 0 0

Edit the file /etc/modules

Add this line:

vboxsf

Step 1 — Installing OpenVPN and EasyRSA

To start off, update your VPN server’s package index and install OpenVPN. OpenVPN is available in Ubuntu's default repositories, so you can use `apt` for the installation:

```
$ sudo apt update
$ sudo apt install openvpn
```

OpenVPN is a TLS/SSL VPN. This means that it utilizes certificates in order to encrypt traffic between the server and clients. To issue trusted certificates, you will set up your own simple certificate authority (CA). To do this, we will download the latest version of EasyRSA, which we will use to build our CA public key infrastructure (PKI), from the project’s official GitHub repository.

As mentioned in the prerequisites, we will build the CA on a standalone server. The reason for this approach is that, if an attacker were able to infiltrate your server, they would be able to access your CA private key and use it to sign new certificates, giving them access to your VPN. Accordingly, managing the CA from a standalone machine helps to prevent unauthorized users from accessing your VPN. Note, as well, that it’s recommended that you keep the CA server turned off when not being used to sign keys as a further precautionary measure.

To begin building the CA and PKI infrastructure, use `wget` to download the latest version of EasyRSA on both your CA machine and your OpenVPN server. To get the latest version, go to

the Releases page on the official EasyRSA GitHub project, copy the download link for the file ending in `.tgz`, and then paste it into the following command:

```
$ wget -P ~/ https://github.com/OpenVPN/easy-rsa/releases/download/v3.0.4/EasyRSA-3.0.4.tgz
```

Then extract the tarball:

```
$ cd ~
$ tar xvf EasyRSA-3.0.4.tgz
```

You have successfully installed all the required software on your server and CA machine. Continue on to configure the variables used by EasyRSA and to set up a CA directory, from which you will generate the keys and certificates needed for your server and clients to access the VPN.

Step 2 — Configuring the EasyRSA Variables and Building the CA

EasyRSA comes installed with a configuration file which you can edit to define a number of variables for your CA.

On your CA machine, navigate to the EasyRSA directory:

```
$ cd ~/EasyRSA-3.0.4/
```

Inside this directory is a file named `vars.example`. Make a copy of this file, and name the copy `vars` without a file extension:

```
$ cp vars.example vars
```

Open this new file using your preferred text editor:

```
$ nano vars
```

Find and change the settings for new certificates. It will look something like this:

~/EasyRSA-3.0.4/vars

```
. . .

#set_var EASYRSA_REQ_COUNTRY    "US"
#set_var EASYRSA_REQ_PROVINCE   "New Hampshire"
#set_var EASYRSA_REQ_CITY       "Manchester"
#set_var EASYRSA_REQ_ORG        "HMS"
#set_var EASYRSA_REQ_EMAIL      "me@example.net"
#set_var EASYRSA_REQ_OU         "HMS Solutions Center"

. . .
```

Uncomment and update the following lines as well. These are specific to the Flexy.

```
. . .
```

```
#set_var EASYRSA_KEY_SIZE          2048
#set_var EASYRSA_NS_SUPPORT        "yes"
#set_var EASYRSA_NS_COMMENT        "Easy-RSA Generated Certificate"
. . .
```

When you are finished, save and close the file.

Note that the EASYRSA_NS commands enable the certificate variable ns-cert-type. This variable is a legacy configuration variable that is embedded in the Flexy core configuration. It is used to fix the certificate usage to either a client only certificate or a server only certificate. There is no way to disable this feature in the Flexy firmware so it must be used. There is no security risk associated with using this legacy feature.

Within the EasyRSA directory is a script called `easyrsa` which is called to perform a variety of tasks involved with building and managing the CA. Run this script with the `init-pki` option to initiate the public key infrastructure on the CA server:

```
$ ./easyrsa init-pki
```

```
Output
. . .
init-pki complete; you may now create a CA or requests.
Your newly created PKI dir is: /home/sammy/EasyRSA-3.0.4/pki
```

After this, call the `easyrsa` script again, following it with the `build-ca` option. This will build the CA and create two important files — `ca.crt` and `ca.key` — which make up the public and private sides of an SSL certificate.

`ca.crt` is the CA's public certificate file which, in the context of OpenVPN, the server and the client use to inform one another that they are part of the same web of trust and not someone performing a man-in-the-middle attack. For this reason, your server and all of your clients will need a copy of the `ca.crt` file.

`ca.key` is the private key which the CA machine uses to sign keys and certificates for servers and clients. If an attacker gains access to your CA and, in turn, your `ca.key` file, they will be able to sign certificate requests and gain access to your VPN, impeding its security. This is why your `ca.key` file should only be on your CA machine and that, ideally, your CA machine should be kept offline when not signing certificate requests as an extra security measure.
If you don't want to be prompted for a password every time you interact with your CA, you can run the `build-ca` command with the `nopass` option, like this:

```
$ ./easyrsa build-ca nopass
```

In the output, you'll be asked to confirm the *common name* for your CA:

```
Output

. . .
Common Name (eg: your user, host, or server name) [Easy-RSA CA]:
```

The common name is the name used to refer to this machine in the context of the certificate authority. You can enter any string of characters for the CA's common name but, for simplicity's sake, press `ENTER` to accept the default name.
With that, your CA is in place and it's ready to start signing certificate requests.

Step 3 — Creating the Server Certificate, Key, and Encryption Files

Now that you have a CA ready to go, you can generate a private key and certificate request from your server and then transfer the request over to your CA to be signed, creating the required certificate. You're also free to create some additional files used during the encryption process.

Note that if you are using the same VM as a Certificate Authority, there will need to be a separate EasyRSA directory for the CA stuff. It can be simpler to just keep things separate even when testing. For security reasons, do not use the same machine as a CA in production environments.

Start by navigating to the EasyRSA directory on your OpenVPN server:

```
$ cd EasyRSA-3.0.4/
```

Copy the vars file from the CA to this install of Easy RSA to ensure the configuration parameters are the same.

From there, run the `easyrsa` script with the `init-pki` option. Although you already ran this command on the CA machine, it's necessary to run it here because your server and CA will have separate PKI directories:

```
$ ./easyrsa init-pki
```

Then call the `easyrsa` script again, this time with the `gen-req` option followed by a common name for the machine. Again, this could be anything you like but it can be helpful to make it something descriptive. Throughout this tutorial, the OpenVPN server's common name will simply be "server". Be sure to include the `nopass` option as well. Failing to do so will password-protect the request file which could lead to permissions issues later on:
Note: If you choose a name other than "server" here, you will have to adjust some of the instructions below. For instance, when copying the generated files to the `/etc/openvpn`

directory, you will have to substitute the correct names. You will also have to modify the `/etc/openvpn/server.conf` file later to point to the correct `.crt` and `.key` files.

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

This will create a private key for the server and a certificate request file called `server.req`. Copy the server key to the `/etc/openvpn/server/` directory:

```
$ sudo cp ~/EasyRSA-3.0.4/pki/private/server.key /etc/openvpn/server/
```

Using a secure method (like SCP, in our example below), transfer the `server.req` file to your CA machine:

```
$ scp ~/EasyRSA-3.0.4/pki/reqs/server.req sammy@your_CA_ip:/tmp
```

Next, on your CA machine, navigate to the EasyRSA directory:

```
$ cd EasyRSA-3.0.4/
```

Using the `easyrsa` script again, import the `server.req` file, following the file path with its common name:

```
$ ./easyrsa import-req /tmp/server.req server
```

Then sign the request by running the `easyrsa` script with the `sign-req` option, followed by the *request type* and the common name. The request type can either be `client` or `server`, so for the OpenVPN server's certificate request, be sure to use the `server` request type:

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

In the output, you'll be asked to verify that the request comes from a trusted source. Type `yes` then press `ENTER` to confirm this:

```
You are about to sign the following certificate.
Please check over the details shown below for accuracy. Note that this request
has not been cryptographically verified. Please be sure it came from a trusted
source or that you have verified the request checksum with the sender.

Request subject, to be signed as a server certificate for 3650 days:

subject=
    commonName                = server
```

```
Type the word 'yes' to continue, or any other input to abort.
  Confirm request details: yes
```

If you encrypted your CA key, you'll be prompted for your password at this point.
Next, transfer the signed certificate back to your VPN server using a secure method:

```
$ scp pki/issued/server.crt sammy@your_server_ip:/tmp
```

Before logging out of your CA machine, transfer the `ca.crt` file to your server as well:

```
$ scp pki/ca.crt sammy@your_server_ip:/tmp
```

Next, log back into your OpenVPN server and copy the `server.crt` and `ca.crt` files into your `/etc/openvpn/server` directory:

```
$ sudo cp /tmp/{server.crt,ca.crt} /etc/openvpn/server
```

Then navigate to your EasyRSA directory:

```
$ cd EasyRSA-3.0.4/
```

From there, create a strong Diffie-Hellman key to use during key exchange by typing:

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

This may take a few minutes to complete. Once it does, generate an HMAC signature to strengthen the server's TLS integrity verification capabilities:

```
$ openvpn --genkey --secret ta.key
```

When the command finishes, copy the two new files to your `/etc/openvpn/server/` directory:

```
$ sudo cp ~/EasyRSA-3.0.4/ta.key /etc/openvpn/server/
```

```
$ sudo cp ~/EasyRSA-3.0.4/pki/dh.pem /etc/openvpn/server/
```

With that, all the certificate and key files needed by your server have been generated. You're ready to create the corresponding certificates and keys which your client machine will use to access your OpenVPN server.

Step 4 — Generating a Client Certificate and Key Pair

Although you can generate a private key and certificate request on your client machine and then send it to the CA to be signed, this guide outlines a process for generating the certificate request on the server. The benefit of this is that we can create a script which will automatically generate client configuration files that contain all of the required keys and certificates. This lets you avoid having to transfer keys, certificates, and configuration files to clients and streamlines the process of joining the VPN.

We will generate a single client key and certificate pair for this guide. If you have more than one client, you can repeat this process for each one. Please note, though, that you will need to pass a unique name value to the script for every client. Throughout this tutorial, the first certificate/key pair is referred to as `client1`.
Get started by creating a directory structure within your home directory to store the client certificate and key files:

```
$ mkdir -p ~/client-configs/keys
```

Since you will store your clients' certificate/key pairs and configuration files in this directory, you should lock down its permissions now as a security measure:

```
$ chmod -R 700 ~/client-configs
```

Next, navigate back to the EasyRSA directory and run the `easyrsa` script with the `gen-req` and `nopass` options, along with the common name for the client:

```
$ cd ~/EasyRSA-3.0.4/
```

```
$ ./easyrsa gen-req client1 nopass
```

Press `ENTER` to confirm the common name. Then, copy the `client1.key` file to the `/client-configs/keys/` directory you created earlier:

```
$ cp pki/private/client1.key ~/client-configs/keys/
```

Next, transfer the `client1.req` file to your CA machine using a secure method:

```
$ scp pki/reqs/client1.req sammy@your_CA_ip:/tmp
```

Log in to your CA machine, navigate to the EasyRSA directory, and import the certificate request:

```
$ ssh sammy@your_CA_ip
```

```
$ cd EasyRSA-3.0.4/
```

```
$ ./easyrsa import-req /tmp/client1.req client1
```

Then sign the request as you did for the server in the previous step. This time, though, be sure to specify the `client` request type:

```
$ ./easyrsa sign-req client client1
```

At the prompt, enter `yes` to confirm that you intend to sign the certificate request and that it came from a trusted source:

```
Output
Type the word 'yes' to continue, or any other input to abort.
```

```
  Confirm request details: yes
```

Again, if you encrypted your CA key, you'll be prompted for your password here.
This will create a client certificate file named `client1.crt`. Transfer this file back to the server:

```
$ scp pki/issued/client1.crt sammy@your_server_ip:/tmp
```

SSH back into your OpenVPN server and copy the client certificate to the `/client-configs/keys/` directory:

```
$ cp /tmp/client1.crt ~/client-configs/keys/
```

Next, copy the `ca.crt` and `ta.key` files to the `/client-configs/keys/` directory as well:

```
$ cp ~/EasyRSA-3.0.4/ta.key ~/client-configs/keys/
```

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

With that, your server and client's certificates and keys have all been generated and are stored in the appropriate directories on your server. There are still a few actions that need to be performed with these files, but those will come in a later step. For now, you can move on to configuring OpenVPN on your server.

Step 5 — Configuring the OpenVPN Service

Now that both your client and server's certificates and keys have been generated, you can begin configuring the OpenVPN service to use these credentials.
Start by copying a sample OpenVPN configuration file into the configuration directory and then extract it in order to use it as a basis for your setup:

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

```
$ sudo gzip -d /etc/openvpn/server.conf.gz
```

Open the server configuration file in your preferred text editor:

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

Find the HMAC section by looking for the `tls-auth` directive. This line should already be uncommented, but if isn't then remove the ";" to uncomment it. Below this line, add the `key-direction` parameter, set to "0":

/etc/openvpn/server.conf

```
tls-auth ta.key 0 # This file is secret
key-direction 0
```

Next, find the section on cryptographic ciphers by looking for the commented out `cipher` lines. The `AES-256-CBC` cipher offers a good level of encryption and is well supported. Again, this line should already be uncommented, but if it isn't then just remove the ";" preceding it:

/etc/openvpn/server.conf

```
cipher AES-256-CBC
```

Below this, add an `auth` directive to select the HMAC message digest algorithm. For this, `SHA256` is a good choice:

/etc/openvpn/server.conf

```
auth SHA256
```

Next, find the line containing a `dh` directive which defines the Diffie-Hellman parameters. Because of some recent changes made to EasyRSA, the filename for the Diffie-Hellman key may be different than what is listed in the example server configuration file. If necessary, change the file name listed here by removing the `2048` so it aligns with the key you generated in the previous step:

/etc/openvpn/server.conf

```
dh dh.pem
```

Finally, find the `user` and `group` settings and remove the ";" at the beginning of each to uncomment these lines:

/etc/openvpn/server.conf

```
user nobody
group nogroup
```

The changes you've made to the sample `server.conf` file up to this point are necessary in order for OpenVPN to function. The changes outlined below are optional, though they too are needed for many common use cases.

(Optional) Push DNS Changes to Redirect All Traffic Through the VPN

The settings above will create the VPN connection between the two machines, but will not force any connections to use the tunnel. If you wish to use the VPN to route all of your traffic, you will likely want to push the DNS settings to the client computers.

There are a few directives in the `server.conf` file which you must change in order to enable this functionality. Find the `redirect-gateway` section and remove the semicolon ";" from the beginning of the `redirect-gateway` line to uncomment it:

/etc/openvpn/server.conf

```
push "redirect-gateway def1 bypass-dhcp"
```

Just below this, find the `dhcp-option` section. Again, remove the ";" from in front of both of the lines to uncomment them:

/etc/openvpn/server.conf

```
push "dhcp-option DNS 208.67.222.222"
push "dhcp-option DNS 208.67.220.220"
```

This will assist clients in reconfiguring their DNS settings to use the VPN tunnel for as the default gateway.

(Optional, not tested by HMS) Adjust the Port and Protocol

By default, the OpenVPN server uses port `1194` and the UDP protocol to accept client connections. If you need to use a different port because of restrictive network environments that your clients might be in, you can change the `port` option. If you are not hosting web content on your OpenVPN server, port `443` is a popular choice since it is usually allowed through firewall rules.

/etc/openvpn/server.conf

```
# Optional!
port 443
```

Oftentimes, the protocol is restricted to that port as well. If so, change `proto` from UDP to TCP:

/etc/openvpn/server.conf

```
# Optional!
proto tcp
```

If you do switch the protocol to TCP, you will need to change the `explicit-exit-notify` directive's value from `1` to `0`, as this directive is only used by UDP. Failing to do so while using TCP will cause errors when you start the OpenVPN service:

/etc/openvpn/server.conf

```
# Optional!
explicit-exit-notify 0
```

If you have no need to use a different port and protocol, it is best to leave these two settings as their defaults.

(Optional) Point to Non-Default Credentials

If you selected a different name during the `./build-key-server` command earlier, modify the `cert` and `key` lines that you see to point to the appropriate `.crt` and `.key` files. If you used the default name, "server", this is already set correctly:

/etc/openvpn/server.conf

```
cert server.crt
key server.key
```

When you are finished, save and close the file.

After going through and making whatever changes to your server's OpenVPN configuration are required for your specific use case, you can begin making some changes to your server's networking.

Step 6 — Adjusting the Server Networking Configuration (Not tested by HMS)

Step 6 would depend on the customers network setup. Customers are left to configure this portion of the system to meet their needs. For the sake of inclusion, step 6 has been left in from the Digital Ocean website import. When testing you should be able to go right to step 7.

There are some aspects of the server's networking configuration that need to be tweaked so that OpenVPN can correctly route traffic through the VPN. The first of these is *IP forwarding*, a method for determining where IP traffic should be routed. This is essential to the VPN functionality that your server will provide.
Adjust your server's default IP forwarding setting by modifying the `/etc/sysctl.conf` file:

```
$ sudo nano /etc/sysctl.conf
```

Inside, look for the commented line that sets `net.ipv4.ip_forward`. Remove the "#" character from the beginning of the line to uncomment this setting:

/etc/sysctl.conf

```
net.ipv4.ip_forward=1
```

Save and close the file when you are finished.
To read the file and adjust the values for the current session, type:

```
$ sudo sysctl -p
```

```
Output
net.ipv4.ip_forward = 1
```

If you followed the Ubuntu 18.04 initial server setup guide listed in the prerequisites, you should have a UFW firewall in place. Regardless of whether you use the firewall to block unwanted traffic (which you almost always should do), for this guide you need a firewall to manipulate some of the traffic coming into the server. Some of the firewall rules must be modified to enable masquerading, an iptables concept that provides on-the-fly dynamic network address translation (NAT) to correctly route client connections.
Before opening the firewall configuration file to add the masquerading rules, you must first find the public network interface of your machine. To do this, type:

```
$ ip route | grep default
```

Your public interface is the string found within this command's output that follows the word "dev". For example, this result shows the interface named `wlp11s0`, which is highlighted below:

```
Output
default via 203.0.113.1 dev wlp11s0 proto static
```

When you have the interface associated with your default route, open the `/etc/ufw/before.rules` file to add the relevant configuration:

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

UFW rules are typically added using the `ufw` command. Rules listed in the `before.rules` file, though, are read and put into place before the conventional UFW rules are loaded. Towards the top of the file, add the highlighted lines below. This will set the default policy for the `POSTROUTING` chain in the `nat` table and masquerade any traffic coming from the VPN. Remember to replace `wlp11s0` in the `-A POSTROUTING` line below with the interface you found in the above command:

/etc/ufw/before.rules

```
#
# rules.before
#
# Rules that should be run before the ufw command line added rules. Custom
# rules should be added to one of these chains:
#   ufw-before-input
#   ufw-before-output
#   ufw-before-forward
#

# START OPENVPN RULES
# NAT table rules
*nat
:POSTROUTING ACCEPT [0:0]
# Allow traffic from OpenVPN client to wlp11s0 (change to the interface
you discovered!)
-A POSTROUTING -s 10.8.0.0/8 -o wlp11s0 -j MASQUERADE
COMMIT
# END OPENVPN RULES

# Don't delete these required lines, otherwise there will be errors
*filter
. . .
```

Save and close the file when you are finished.

Next, you need to tell UFW to allow forwarded packets by default as well. To do this, open the `/etc/default/ufw` file:

```
$ sudo nano /etc/default/ufw
```

Inside, find the `DEFAULT_FORWARD_POLICY` directive and change the value from `DROP` to `ACCEPT`:

/etc/default/ufw

```
DEFAULT_FORWARD_POLICY="ACCEPT"
```

Save and close the file when you are finished.
Next, adjust the firewall itself to allow traffic to OpenVPN. If you did not change the port and protocol in the `/etc/openvpn/server.conf` file, you will need to open up UDP traffic to port `1194`. If you modified the port and/or protocol, substitute the values you selected here.
In case you forgot to add the SSH port when following the prerequisite tutorial, add it here as well:

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

```
$ sudo ufw allow OpenSSH
```

After adding those rules, disable and re-enable UFW to restart it and load the changes from all of the files you've modified:

```
$ sudo ufw disable
```

```
$ sudo ufw enable
```

Your server is now configured to correctly handle OpenVPN traffic.

Step 7 — Starting and Enabling the OpenVPN Service

You're finally ready to start the OpenVPN service on your server. This is done using the systemd utility `systemctl`.
Start the OpenVPN server by specifying your configuration file name as an instance variable after the systemd unit file name. The configuration file for your server is called `/etc/openvpn/server.conf`, so add `@server` to end of your unit file when calling it:

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

Double-check that the service has started successfully by typing:

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

If everything went well, your output will look something like this:

```
Output
openvpn@server.service - OpenVPN connection to server
   Loaded: loaded (/lib/systemd/system/openvpn@.service; disabled; vendor
preset: enabled)
   Active: active (running) since Tue 2016-05-03 15:30:05 EDT; 47s ago
     Docs: man:openvpn(8)
           https://community.openvpn.net/openvpn/wiki/Openvpn23ManPage
           https://community.openvpn.net/openvpn/wiki/HOWTO
```

```
  Process: 5852 ExecStart=/usr/sbin/openvpn --daemon ovpn-%i --status
/run/openvpn/%i.status 10 --cd /etc/openvpn --script-security 2 --config
/etc/openvpn/%i.conf --writepid /run/openvpn/%i.pid (code=exited, sta
 Main PID: 5856 (openvpn)
    Tasks: 1 (limit: 512)
   CGroup: /system.slice/system-openvpn.slice/openvpn@server.service
           └─5856 /usr/sbin/openvpn --daemon ovpn-server --status
/run/openvpn/server.status 10 --cd /etc/openvpn --script-security 2 --
config /etc/openvpn/server.conf --writepid /run/openvpn/server.pid
```

You can also check that the OpenVPN `tun0` interface is available by typing:

```
$ ip addr show tun0
```

This will output a configured interface:

```
Output
4: tun0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue
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
```

After starting the service, enable it so that it starts automatically at boot:

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

Your OpenVPN service is now up and running. Before you can start using it, though, you must first create a configuration file for the client machine. This tutorial already went over how to create certificate/key pairs for clients, and in the next step we will demonstrate how to create an infrastructure that will generate client configuration files easily.

For debugging the network connection you can ping the open vpn from a linux machine with the netcat command. The following is pinging an openvpn server at ip 172.16.0.45 on the port 1194 with the verbose flag and udp flag. If you are testing against a tcp Openvpn, remove the udp flag and it will ping with tcp.

```
$ nc -vu 172.16.0.45 1194
```

Commands to turn on and off the OpenVPN server are listed below. If you make any changes to your server configuration, restart OpenVPN. Note that the server name is what gets put after the @ symbol. If you changed it to something else during the setup, you will need to know that name. The below commands worked during HMS testing.

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

```
$ sudo systemctl restart openvpn@server
```

Another very useful tool to check out is the OpenVPN client to test for connectivity from your desktop during setup. To use this tool, download it from here

https://openvpn.net/community-downloads/

you would then load in your client1.ovpn configuration file with the following changes:
You will need to imbed your certificates directly into the file by switching out all file path references with the imbedded certificates like the below example.
These file paths:

```
ca /usr/ca.crt

cert /usr/client1.crt

key /usr/client1.key
```

would get switched to this format below. Copy and paste the contents of those certificates into the correct locations.

```
<ca>
-----BEGIN CERTIFICATE-----
  "Copy your certificate here"
-----END CERTIFICATE-----
</ca>

<cert>
-----BEGIN CERTIFICATE-----
-----END CERTIFICATE-----
</cert>

<key>
-----BEGIN PRIVATE KEY-----
-----END PRIVATE KEY-----
</key>
```

Import the client config by right clicking the icon on the bottom right of your screen and hitting import file, then import the correct file.

Now you can easily test for client connectivity by right clicking the icon on the bottom right of your screen and hitting connect. Right click and hit edit config to change your config file.

Step 8 — Creating the Client Configuration Infrastructure

Creating configuration files for OpenVPN clients can be somewhat involved, as every client must have its own config and each must align with the settings outlined in the server's

configuration file. Rather than writing a single configuration file that can only be used on one client, this step outlines a process for building a client configuration file for your flexy.

The GUI interface on your flexy can be used to configure a number of the OpenVPN Client setting but not all. As a result, the use of the GUI will be minimized, and many of the configuration will be done using configuration files. Note that when GUI configuration files are changes it may overwrite your uploaded files. You should double check this has not happened after changes are made.

In the Flexy GUI confirm the following settings:

1) Setup > System > Communications > Networking > VPN Connection > Main setup
    a. Ensure Establish outgoing VPN to server is selected
    b. No other options should be selected
2) Setup > System > Communications > Networking > VPN Connection > Global
    a. Internet connection proxy – No proxy
    b. Talk2M Account Name – Customers Account Name
    c. Talk2M Access Server – talk2m_pro
    d. Advanced settings Diagnosis level – High
    e. Advanced settings Port In – 0
    f. Advanced settings Port Out – This should be the port of the OpenVPN Server
    g. Advanced settings 'keep alive' interval – 120
    h. Advanced settings VPN Driver Mode – TUN
    i. Advanced settings VPN Protocol – UDP
3) Setup > System > Communications > Networking > VPN Connection > Incoming
    a. Leave as default
4) Setup > System > Communications > Networking > VPN Connection > Outgoing
    a. Establish VPN connection should be checked
    b. Primary server – IP address of OpenVPN Server
    c. Secondary server – IP address of backup OpenVPN Server
    d. Connect to…
        i. Select VPN Server in the drop down
        ii. The Private key, Certificate and CA certificate can be left unchanged or blank

Once the above GUI settings have been made reboot the Flexy. Using an FTP tool like FileZilla connect to the flexy once it has completed the reboot. The complete Flexy configuration is stored in multiple text files at the top level of the Flexy device. These file are overwrite by the GUI and should not be used to implement the VPN configuration. A link to a user configuration file is needed to create a user controlled file for configuration. This configuration link will be added to the top level comcfg.txt file.

1) FTP into the Flexy
2) Copy the /comcfg.txt file to a host PC
3) Edit the file and add this line and the end of the file

    VPNCfgFile:+/usr/client1.ovpn

4) Copy the new comcfg.txt back to the Flexy overwrite the old file
5) Create a file on the host PC called client1.ovpn

a. If you are connecting multiple flexy devices a unique certificate will be created for each flexy. This will result in the creation of a client1.crt, client2.crt, client3.crt…. There will also be unique client keys for each flexy. As a result, it is a good idea to number the client config file client1.ovpn, client2.ovpn. This will help prevent confusion when managing multiple Flexy devices.

6) The client1.ovpn file used in this app note is included in appendix B
   a. Different cipher files can be used for example auth SHA256 has been tested and will work
   b. It is critical that if auth SHA256 is used an identical change is made to the server.conf
   c. ns-cert-type server is not included in the settings because it is automatically set by the Flexy firmware. Do not add ns-cert-type server to the flexy client1.ovpn file.
   d. comp-lzo is a requirement alternative compressions are not supported
7) FTP the following files over to the Flexy /usr directory: ca.crt, client1.ovpn, client1.crt, client1.key.
8) Reboot the Flexy.

Useful Tools

During the development of this application note the following commands and tools were used to debug the system. End users may find these tools useful in debugging the system.

1) Checking a Key, Certification pair. If the output of these two functions match the CRT and Key are a pair.
```
$ openssl rsa -noout -modulus -in client1.key

$ openssl x509 -noout -modulus -in client1.crt
```

2) Determining the version of ssl
```
$ Openssl version -a
```

3) Command line reboot VirtualBox Linux box
```
$ sudo shutdown -r now
```

4) By default the OpenVPN server will startup automatically when Linux boots. It may be helpful to control the time when the OpenVPN server starts during debug. Open this file for editing /etc/default/openvpn. Uncomment #AUTOSTART=”none”. Save the file and restart Linux. This prevent auto start. Execute the following line to start the server.
```
$ Sudo systemctl start openvpn@servername
```

5) Journalctl can be used to dump status associated with the OpenVPN Sever
```
$ sudo journalctl –u openvpn@servername
$ sudo journalctl –xe openvpn@servername
```

6) The OpenVPN server.conf file support multiple log files by default they are defined as follows:
   a. status /var/log/openvpn/openvpn-status.log
   b. log /var/log/openvpn/openvpn.log
   c. Choose the truncated on startup option to eliminate confusion when reviewing the log

Error Messages and Typical causes

Debugging a VPN solution requires access to the real-time logs on the Flexy and the OpenVPN logs found in /var/log/openvpn on the server side. The server side files are as follows unless the names have been changed in the server.conf file.

Ipp.txt – list of IP addresses assigned by the server

openvpn.log – High level status log

openvpn-status.log – Detailed status log

Most of the debug work will involve checking the openvpn-status.log and the real-time logs on Flexy. During debug limit the real-time log messages to VPN only by selecting VPN in the dropdown menu in the upper left hand corner of the real-time log screen. In the line items below the log will be listed followed by the message or error.

- Real-time log: Status Message with "P_ CONTROL_HARD_RESET_CLIENT_V2"
    - This message indicated the Flexy is closing the port because the VPN server did not response
    - This indicates the VPN server is not receiving the hello message from Flexy
    - This indicates the server firewall is not configured properly
        - For Azure deployment verify the NSG is properly configuration for the OpenVPN Server VM
        - Ensure the NSG being configured is the NSG associated with the OpenVPN Sever
        - Verify the ACL rules will allow the Flexy device to send messages to the OpenVPN Server

Debugging a VPN solution requires access to the real-time logs on the Flexy and the OpenVPN logs found in /var/log/openvpn on the server side. The server side files are as follows unless the names have been changed in the server.conf file.

Ipp.txt – list of IP addresses assigned by the server

openvpn.log – High level status log

openvpn-status.log – Detailed status log

Most of the debug work will involve checking the openvpn-status.log and the real-time logs on Flexy. During debug limit the real-time log messages to VPN only by selecting VPN in the dropdown menu in the upper left hand corner of the real-time log screen. In the line items below the log will be listed followed by the message or error.

- Real-time log: Status Message with "P_ CONTROL_HARD_RESET_CLIENT_V2"
    - This message indicated the Flexy is closing the port because the VPN server did not response
    - This indicates the VPN server is not receiving the hello message from Flexy
    - This indicates the server firewall is not configured properly
        - For Azure deployment verify the NSG is properly configuration for the OpenVPN Server VM
        - Ensure the NSG being configured is the NSG associated with the OpenVPN Sever

- Verify the ACL rules will allow the Flexy device to send messages to the OpenVPN Server

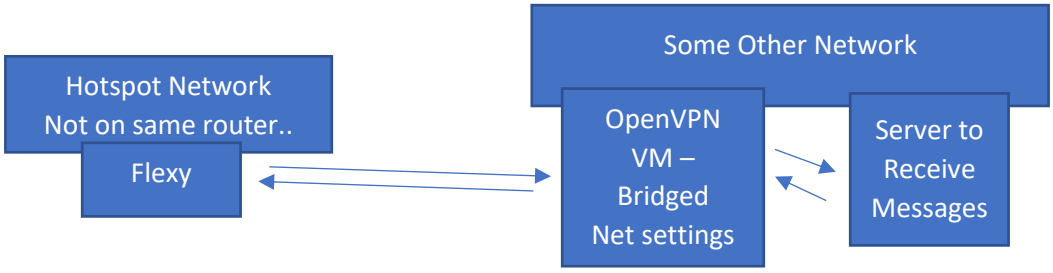
IMPORTANT FOR TESTING:

For testing purposes, if you are testing this all on the same network you will need to insure that the traffic leaves your network for it to go through the VPN. The easiest way to do this is by setting up a wifi hotspot and connecting your flexy to that hotspot

You could than connect your OpenVPN server to a network with portforwarding for the port 1194 to your OpenVPN machine. Flexy seems to not connect to NAT Virtual machines. Used bridged Network settings to give it its own IP address. Note that some routers do not like this for port forwarding… if that happens to you than you should install the server image on your hard drive.

Example Test Environment:

If the OpenVPN machine is a bridged virtual machine on 192.169.0.35, add a port forwarding rule in your router settings for that private IP and port. To access this from outside the network, you would then connect to your public IP address with that port which leads the traffic to your OpenVPN machine.



Appendix A: EasyRSA “vars”

```
# Easy-RSA 3 parameter settings

# NOTE: If you installed Easy-RSA from your distro's package manager, don't edit
# this file in place -- instead, you should copy the entire easy-rsa directory
# to another location so future upgrades don't wipe out your changes.

# HOW TO USE THIS FILE
#
# vars.example contains built-in examples to Easy-RSA settings. You MUST name
# this file 'vars' if you want it to be used as a configuration file. If you do
# not, it WILL NOT be automatically read when you call easyrsa commands.
#
# It is not necessary to use this config file unless you wish to change
# operational defaults. These defaults should be fine for many uses without the
# need to copy and edit the 'vars' file.
#
# All of the editable settings are shown commented and start with the command
# 'set_var' -- this means any set_var command that is uncommented has been
# modified by the user. If you're happy with a default, there is no need to
# define the value to its default.

# NOTES FOR WINDOWS USERS
#
# Paths for Windows  *MUST* use forward slashes, or optionally double-esscaped
# backslashes (single forward slashes are recommended.) This means your path to
# the openssl binary might look like this:
# "C:/Program Files/OpenSSL-Win32/bin/openssl.exe"

# A little housekeeping: DON'T EDIT THIS SECTION
#
# Easy-RSA 3.x doesn't source into the environment directly.
# Complain if a user tries to do this:
if [ -z "$EASYRSA_CALLER" ]; then
        echo "You appear to be sourcing an Easy-RSA 'vars' file." >&2
        echo "This is no longer necessary and is disallowed. See the section called" >&2
        echo "'How to use this file' near the top comments for more details." >&2
        return 1
fi

# DO YOUR EDITS BELOW THIS POINT

# This variable is used as the base location of configuration files needed by
# easyrsa.  More specific variables for specific files (e.g., EASYRSA_SSL_CONF)
# may override this default.
```

```
#
# The default value of this variable is the location of the easyrsa script
# itself, which is also where the configuration files are located in the
# easy-rsa tree.

#set_var EASYRSA   "${0%/*}"

# If your OpenSSL command is not in the system PATH, you will need to define the
# path to it here. Normally this means a full path to the executable, otherwise
# you could have left it undefined here and the shown default would be used.
#
# Windows users, remember to use paths with forward-slashes (or escaped
# back-slashes.) Windows users should declare the full path to the openssl
# binary here if it is not in their system PATH.

#set_var EASYRSA_OPENSSL        "openssl"
#
# This sample is in Windows syntax -- edit it for your path if not using PATH:
#set_var EASYRSA_OPENSSL        "C:/Program Files/OpenSSL-Win32/bin/openssl.exe"

# Edit this variable to point to your soon-to-be-created key directory.  By
# default, this will be "$PWD/pki" (i.e. the "pki" subdirectory of the
# directory you are currently in).
#
# WARNING: init-pki will do a rm -rf on this directory so make sure you define
# it correctly! (Interactive mode will prompt before acting.)

#set_var EASYRSA_PKI                "$PWD/pki"

# Define X509 DN mode.
# This is used to adjust what elements are included in the Subject field as the DN
# (this is the "Distinguished Name.")
# Note that in cn_only mode the Organizational fields further below aren't used.
#
# Choices are:
#   cn_only  - use just a CN value
#   org      - use the "traditional" Country/Province/City/Org/OU/email/CN format

#set_var EASYRSA_DN       "cn_only"

# Organizational fields (used with 'org' mode and ignored in 'cn_only' mode.)
# These are the default values for fields which will be placed in the
# certificate.  Don't leave any of these fields blank, although interactively
# you may omit any specific field by typing the "." symbol (not valid for
# email.)
```

```
set_var EASYRSA_REQ_COUNTRY     "US"
set_var EASYRSA_REQ_PROVINCE    "NH"
set_var EASYRSA_REQ_CITY        "Manchester"
set_var EASYRSA_REQ_ORG         "HMS"
set_var EASYRSA_REQ_EMAIL       "olwa@hms.se"
set_var EASYRSA_REQ_OU          "HMS Solutions Center"

# Choose a size in bits for your keypairs. The recommended value is 2048.  Using
# 2048-bit keys is considered more than sufficient for many years into the
# future. Larger keysizes will slow down TLS negotiation and make key/DH param
# generation take much longer. Values up to 4096 should be accepted by most
# software. Only used when the crypto alg is rsa (see below.)

set_var EASYRSA_KEY_SIZE        2048

# The default crypto mode is rsa; ec can enable elliptic curve support.
# Note that not all software supports ECC, so use care when enabling it.
# Choices for crypto alg are: (each in lower-case)
#  * rsa
#  * ec

#set_var EASYRSA_ALGO            rsa

# Define the named curve, used in ec mode only:

#set_var EASYRSA_CURVE           secp384r1

# In how many days should the root CA key expire?

#set_var EASYRSA_CA_EXPIRE    3650

# In how many days should certificates expire?

#set_var EASYRSA_CERT_EXPIRE          3650

# How many days until the next CRL publish date?  Note that the CRL can still be
# parsed after this timeframe passes. It is only used for an expected next
# publication date.

#set_var EASYRSA_CRL_DAYS     180

# Support deprecated "Netscape" extensions? (choices "yes" or "no".) The default
# is "no" to discourage use of deprecated extensions. If you require this
# feature to use with --ns-cert-type, set this to "yes" here. This support
# should be replaced with the more modern --remote-cert-tls feature.  If you do
# not use --ns-cert-type in your configs, it is safe (and recommended) to leave
```

```
# this defined to "no".  When set to "yes", server-signed certs get the
# nsCertType=server attribute, and also get any NS_COMMENT defined below in the
# nsComment field.

set_var EASYRSA_NS_SUPPORT   "yes"

# When NS_SUPPORT is set to "yes", this field is added as the nsComment field.
# Set this blank to omit it. With NS_SUPPORT set to "no" this field is ignored.

set_var EASYRSA_NS_COMMENT           "Easy-RSA Generated Certificate"

# A temp file used to stage cert extensions during signing. The default should
# be fine for most users; however, some users might want an alternative under a
# RAM-based FS, such as /dev/shm or /tmp on some systems.

#set_var EASYRSA_TEMP_FILE    "$EASYRSA_PKI/extensions.temp"

# !!
# NOTE: ADVANCED OPTIONS BELOW THIS POINT
# PLAY WITH THEM AT YOUR OWN RISK
# !!

# Broken shell command aliases: If you have a largely broken shell that is
# missing any of these POSIX-required commands used by Easy-RSA, you will need
# to define an alias to the proper path for the command.  The symptom will be
# some form of a 'command not found' error from your shell. This means your
# shell is BROKEN, but you can hack around it here if you really need. These
# shown values are not defaults: it is up to you to know what you're doing if
# you touch these.
#
#alias awk="/alt/bin/awk"
#alias cat="/alt/bin/cat"

# X509 extensions directory:
# If you want to customize the X509 extensions used, set the directory to look
# for extensions here. Each cert type you sign must have a matching filename,
# and an optional file named 'COMMON' is included first when present. Note that
# when undefined here, default behaviour is to look in $EASYRSA_PKI first, then
# fallback to $EASYRSA for the 'x509-types' dir.  You may override this
# detection with an explicit dir here.
#
#set_var EASYRSA_EXT_DIR        "$EASYRSA/x509-types"

# OpenSSL config file:
# If you need to use a specific openssl config file, you can reference it here.
# Normally this file is auto-detected from a file named openssl-easyrsa.cnf from the
```

```
# EASYRSA_PKI or EASYRSA dir (in that order.) NOTE that this file is Easy-RSA
# specific and you cannot just use a standard config file, so this is an
# advanced feature.

#set_var EASYRSA_SSL_CONF      "$EASYRSA/openssl-easyrsa.cnf"

# Default CN:
# This is best left alone. Interactively you will set this manually, and BATCH
# callers are expected to set this themselves.

#set_var EASYRSA_REQ_CN                "ChangeMe"

# Cryptographic digest to use.
# Do not change this default unless you understand the security implications.
# Valid choices include: md5, sha1, sha256, sha224, sha384, sha512

#set_var EASYRSA_DIGEST                "sha256"

# Batch mode. Leave this disabled unless you intend to call Easy-RSA explicitly
# in batch mode without any user input, confirmation on dangerous operations,
# or most output. Setting this to any non-blank string enables batch mode.

#set_var EASYRSA_BATCH                 ""
```

Appendix B: OpenVPN Server “server.conf”

```
#############################################
# Sample OpenVPN 2.0 config file for        #
# multi-client server.                      #
#                                           #
# This file is for the server side          #
# of a many-clients <-> one-server          #
# OpenVPN configuration.                    #
#                                           #
# OpenVPN also supports                     #
# single-machine <-> single-machine         #
# configurations (See the Examples page     #
# on the web site for more info).           #
#                                           #
# This config should work on Windows        #
# or Linux/BSD systems.  Remember on        #
# Windows to quote pathnames and use        #
# double backslashes, e.g.:                 #
# "C:\\Program Files\\OpenVPN\\config\\foo.key" #
#                                           #
# Comments are preceded with '#' or ';'     #
#############################################

# Which local IP address should OpenVPN
# listen on? (optional)
;local a.b.c.d

# Which TCP/UDP port should OpenVPN listen on?
# If you want to run multiple OpenVPN instances
# on the same machine, use a different port
# number for each one.  You will need to
# open up this port on your firewall.
port 1194

# TCP or UDP server?
;proto tcp
proto udp

# "dev tun" will create a routed IP tunnel,
# "dev tap" will create an ethernet tunnel.
# Use "dev tap0" if you are ethernet bridging
# and have precreated a tap0 virtual interface
# and bridged it with your ethernet interface.
# If you want to control access policies
# over the VPN, you must create firewall
# rules for the the TUN/TAP interface.
```

```
# On non-Windows systems, you can give
# an explicit unit number, such as tun0.
# On Windows, use "dev-node" for this.
# On most systems, the VPN will not function
# unless you partially or fully disable
# the firewall for the TUN/TAP interface.
;dev tap
dev tun

# Windows needs the TAP-Win32 adapter name
# from the Network Connections panel if you
# have more than one.  On XP SP2 or higher,
# you may need to selectively disable the
# Windows firewall for the TAP adapter.
# Non-Windows systems usually don't need this.
;dev-node MyTap

# SSL/TLS root certificate (ca), certificate
# (cert), and private key (key).  Each client
# and the server must have their own cert and
# key file.  The server and all clients will
# use the same ca file.
#
# See the "easy-rsa" directory for a series
# of scripts for generating RSA certificates
# and private keys.  Remember to use
# a unique Common Name for the server
# and each of the client certificates.
#
# Any X509 key management system can be used.
# OpenVPN can also use a PKCS #12 formatted key file
# (see "pkcs12" directive in man page).
# HMS explict call out of path, rename according to your server name
ca /etc/openvpn/server/ca.crt
cert /etc/openvpn/server/server.crt
key /etc/openvpn/server/server.key  # This file should be kept secret

# Diffie hellman parameters.
# Generate your own with:
#   openssl dhparam -out dh2048.pem 2048
# HMS use key generated during setup
dh /etc/openvpn/server/dh.pem

# Network topology
# Should be subnet (addressing via IP)
# unless Windows clients v2.0.9 and lower have to
```

```
# be supported (then net30, i.e. a /30 per client)
# Defaults to net30 (not recommended)
;topology subnet

# Configure server mode and supply a VPN subnet
# for OpenVPN to draw client addresses from.
# The server will take 10.8.0.1 for itself,
# the rest will be made available to clients.
# Each client will be able to reach the server
# on 10.8.0.1. Comment this line out if you are
# ethernet bridging. See the man page for more info.
server 10.8.0.0 255.255.255.0

# Maintain a record of client <-> virtual IP address
# associations in this file.  If OpenVPN goes down or
# is restarted, reconnecting clients can be assigned
# the same virtual IP address from the pool that was
# previously assigned.
ifconfig-pool-persist /var/log/openvpn/ipp.txt

# Configure server mode for ethernet bridging.
# You must first use your OS's bridging capability
# to bridge the TAP interface with the ethernet
# NIC interface.  Then you must manually set the
# IP/netmask on the bridge interface, here we
# assume 10.8.0.4/255.255.255.0.  Finally we
# must set aside an IP range in this subnet
# (start=10.8.0.50 end=10.8.0.100) to allocate
# to connecting clients.  Leave this line commented
# out unless you are ethernet bridging.
;server-bridge 10.8.0.4 255.255.255.0 10.8.0.50 10.8.0.100

# Configure server mode for ethernet bridging
# using a DHCP-proxy, where clients talk
# to the OpenVPN server-side DHCP server
# to receive their IP address allocation
# and DNS server addresses.  You must first use
# your OS's bridging capability to bridge the TAP
# interface with the ethernet NIC interface.
# Note: this mode only works on clients (such as
# Windows), where the client-side TAP adapter is
# bound to a DHCP client.
;server-bridge

# Push routes to the client to allow it
# to reach other private subnets behind
```

```
# the server.  Remember that these
# private subnets will also need
# to know to route the OpenVPN client
# address pool (10.8.0.0/255.255.255.0)
# back to the OpenVPN server.
;push "route 192.168.10.0 255.255.255.0"
;push "route 192.168.20.0 255.255.255.0"

# To assign specific IP addresses to specific
# clients or if a connecting client has a private
# subnet behind it that should also have VPN access,
# use the subdirectory "ccd" for client-specific
# configuration files (see man page for more info).

# EXAMPLE: Suppose the client
# having the certificate common name "Thelonious"
# also has a small subnet behind his connecting
# machine, such as 192.168.40.128/255.255.255.248.
# First, uncomment out these lines:
;client-config-dir ccd
;route 192.168.40.128 255.255.255.248
# Then create a file ccd/Thelonious with this line:
#   iroute 192.168.40.128 255.255.255.248
# This will allow Thelonious' private subnet to
# access the VPN.  This example will only work
# if you are routing, not bridging, i.e. you are
# using "dev tun" and "server" directives.

# EXAMPLE: Suppose you want to give
# Thelonious a fixed VPN IP address of 10.9.0.1.
# First uncomment out these lines:
;client-config-dir ccd
;route 10.9.0.0 255.255.255.252
# Then add this line to ccd/Thelonious:
#   ifconfig-push 10.9.0.1 10.9.0.2

# Suppose that you want to enable different
# firewall access policies for different groups
# of clients.  There are two methods:
# (1) Run multiple OpenVPN daemons, one for each
#     group, and firewall the TUN/TAP interface
#     for each group/daemon appropriately.
# (2) (Advanced) Create a script to dynamically
#     modify the firewall in response to access
#     from different clients.  See man
#     page for more info on learn-address script.
```

```
;learn-address ./script

# If enabled, this directive will configure
# all clients to redirect their default
# network gateway through the VPN, causing
# all IP traffic such as web browsing and
# and DNS lookups to go through the VPN
# (The OpenVPN server machine may need to NAT
# or bridge the TUN/TAP interface to the internet
# in order for this to work properly).
;push "redirect-gateway def1 bypass-dhcp"

# Certain Windows-specific network settings
# can be pushed to clients, such as DNS
# or WINS server addresses.  CAVEAT:
# http://openvpn.net/faq.html#dhcpcaveats
# The addresses below refer to the public
# DNS servers provided by opendns.com.
;push "dhcp-option DNS 208.67.222.222"
;push "dhcp-option DNS 208.67.220.220"

# Uncomment this directive to allow different
# clients to be able to "see" each other.
# By default, clients will only see the server.
# To force clients to only see the server, you
# will also need to appropriately firewall the
# server's TUN/TAP interface.
;client-to-client

# Uncomment this directive if multiple clients
# might connect with the same certificate/key
# files or common names.  This is recommended
# only for testing purposes.  For production use,
# each client should have its own certificate/key
# pair.
#
# IF YOU HAVE NOT GENERATED INDIVIDUAL
# CERTIFICATE/KEY PAIRS FOR EACH CLIENT,
# EACH HAVING ITS OWN UNIQUE "COMMON NAME",
# UNCOMMENT THIS LINE OUT.
;duplicate-cn

# The keepalive directive causes ping-like
# messages to be sent back and forth over
# the link so that each side knows when
# the other side has gone down.
```

```
# Ping every 10 seconds, assume that remote
# peer is down if no ping received during
# a 120 second time period.
keepalive 10 120

# For extra security beyond that provided
# by SSL/TLS, create an "HMAC firewall"
# to help block DoS attacks and UDP port flooding.
#
# Generate with:
#   openvpn --genkey --secret ta.key
#
# The server and each client must have
# a copy of this key.
# The second parameter should be '0'
# on the server and '1' on the clients.
#tls-auth ta.key 0 # This file is secret -- HMS comment this line

# Select a cryptographic cipher.
# This config item must be copied to
# the client config file as well.
# Note that v2.4 client/server will automatically
# negotiate AES-256-GCM in TLS mode.
# See also the ncp-cipher option in the manpage
cipher AES-256-CBC
auth SHA1

# Enable compression on the VPN link and push the
# option to the client (v2.4+ only, for earlier
# versions see below)
;compress lz4-v2
;push "compress lz4-v2"

# For compression compatible with older clients use comp-lzo
# If you enable it here, you must also
# enable it in the client config file.
# HMS uncomment this line
comp-lzo

# The maximum number of concurrently connected
# clients we want to allow.
;max-clients 100

# It's a good idea to reduce the OpenVPN
# daemon's privileges after initialization.
#
```

```
# You can uncomment this out on
# non-Windows systems.
;user nobody
;group nogroup

# The persist options will try to avoid
# accessing certain resources on restart
# that may no longer be accessible because
# of the privilege downgrade.
persist-key
persist-tun

# Output a short status file showing
# current connections, truncated
# and rewritten every minute.
status /var/log/openvpn/openvpn-status.log

# By default, log messages will go to the syslog (or
# on Windows, if running as a service, they will go to
# the "\Program Files\OpenVPN\log" directory).
# Use log or log-append to override this default.
# "log" will truncate the log file on OpenVPN startup,
# while "log-append" will append to it.  Use one
# or the other (but not both).
# HMS uncomment for shorter log file
log         /var/log/openvpn/openvpn.log
;log-append  /var/log/openvpn/openvpn.log

# Set the appropriate level of log
# file verbosity.
#
# 0 is silent, except for fatal errors
# 4 is reasonable for general usage
# 5 and 6 can help to debug connection problems
# 9 is extremely verbose
# HMS increase to 5 to help debug if there is an issue
verb 5

# Silence repeating messages.  At most 20
# sequential messages of the same message
# category will be output to the log.
;mute 20

# Notify the client that when the server restarts so it
# can automatically reconnect.
explicit-exit-notify 1
```

```
# HMS client only certificate requirement
# Certificate is generated for a client only, legacy requirement
ns-cert-type client
```

Appendix C: Flexy “client1.ovpn”

```
client

dev tun

proto udp

remote 10.10.35.115

port 1194

resolv-retry infinite

comp-lzo

nobind

persist-key

persist-tun

keepalive 10 60

ca /usr/ca.crt

cert /usr/client1.crt

key /usr/client1.key

cipher AES-256-CBC

auth SHA1

#mute-replay-warnings

verb 1
```