# Linux VPNs with OpenVPN, Part IV

MICK BAUER

**Use dangerous local-area networks without fear with OpenVPN.**

**For the past** few months, I've been describing how to build a Virtual Private Network (VPN) server using OpenVPN, a free, multiplatform, TLS/SSL-based VPN dæmon. My example usage scenario involves the common "road warrior" setup where remote users connect back to a VPN server on their "home network" by establishing an encrypted VPN tunnel over the Internet.

Last month, in Part III, I finished a line-by-line walk-through of an example OpenVPN server configuration file (server.ovpn) shown here for your reference (Listing 1).

I then talked about running OpenVPN as a server process (the same executable can be run either as a dæmon/listener or as a client process), either running in the foreground, with all log messages printed to the console:

```
bash-$ sudo openvpn --config ./server.ovpn
```

or in the background, with all log messages being written to /var/log/daemon.log:

```
bash-$ sudo openvpn --daemon --config ./server.ovpn
```

While in the early stages of getting OpenVPN working on both server and clients, you'll definitely want to run the server dæmon in the foreground, because you'll probably have to stop and restart it

**Listing 1. Server's server.ovpn File**

```
port 1194
proto udp
dev tun

ca 2.0/keys/ca.crt
cert 2.0/keys/server.crt
key 2.0/keys/server.key  # Keep this file secret
dh 2.0/keys/dh1024.pem
tls-auth 2.0/keys/ta.key 0

server 10.31.33.0 255.255.255.0
ifconfig-pool-persist ipp.txt
push "redirect-gateway def1 bypass-dhcp"

keepalive 10 120

cipher BF-CBC        # Blowfish (default)
comp-lzo
max-clients 2

user nobody
group nogroup
persist-key
persist-tun

status openvpn-status.log
verb 3
mute 20
```

**Listing 2. Client's client.ovpn File**

```
client
proto udp
dev tun

remote 1.2.3.4 1194

nobind

ca ca.crt
cert minion.crt
key minion.key

ns-cert-type server
tls-auth ta.key 1

cipher BF-CBC
comp-lzo

user nobody
group nogroup
persist-key
persist-tun

mute-replay-warnings

verb 3
mute 20
```

through configuration tweaks anyhow. Once everything's working, you can put an init-script into your server's /etc/init.d directory that starts OpenVPN in dæmon mode automatically at startup time.

## OpenVPN Client Configuration

This brings us to client configuration. Listing 2 shows a sample client configuration file, client.ovpn. Let's dissect it!

First is the `client` directive. Like `server`, which we covered last time, client is actually a helper directive that, when read by the openvpn command, expands to two other directives: pull, which instructs OpenVPN to accept options pushed to it by the OpenVPN server it connects to, and tls-client, which enables TLS (SSL) encryption and tells OpenVPN to assume the role of client any time it initiates a TLS transaction.

Next comes `proto udp`, which tells OpenVPN to use UDP packets to build its VPN tunnel. This setting needs to be the same as what's specified on the server to which you wish to connect.

Next comes `dev tun`, which tells OpenVPN to encapsulate IP packets via a /dev/tun interface, rather than Ethernet frames via a /dev/tap device. I'm sticking to IP encapsulation in my examples, and besides, this setting has to be the same as on the server to which you wish to connect.

And, to which server do you wish to connect? The one specified in the `remote` directive, which has two parameters, IP address (or hostname) and port. In Listing 2, these are set to 1.2.3.4 1194, specifically UDP port 1194. (If earlier I had set `proto` to `tcp-client`, OpenVPN would assume you mean TCP port 1194 here.)

The IP address of my example server is 1.2.3.4, which may strike you as improbable, but this address is, at least, Internet-routable. If you're going to connect to your OpenVPN server from across the Internet, you'll need to target an Internet-routable IP address. In my home setup, this is actually the address of my DSL router, which I've configured to redirect UDP 1194 connections to the same port on my OpenVPN server, whose real IP address is a non-Internet-routable 192.168.0.0 address.

After `remote` comes `nobind`, which tells OpenVPN to allow the local IP stack (your Linux kernel's TCP/IP modules) to assign a local port from which to send and receive OpenVPN packets dynamically, rather than have the OpenVPN dæmon "bind" to (listen on) a specific port like a server process would. This setting, therefore, is suitable only for VPN client systems.

## Certificate/Key-Related Directives

Next, there are five directives related to helper files that OpenVPN will need to read in order to build a tunnel. `ca` specifies a file containing at least one Certificate Authority certificate, specifically, the certificate of whatever CA will have signed the OpenVPN's server certificate. If you try to connect to a server whose server certificate was *not* signed by a CA key/certificate specified here, your OpenVPN client process will terminate the connection.

`cert` specifies a file containing a client certificate for your OpenVPN client process to present to the OpenVPN server. This certificate needs to have been signed by a CA whose certificate resides in the *server's* ca file, or the server will reject connections from *you*.

In many if not most cases, the simplest way to handle these certificates is to use the same CA to create and sign both server and client certificates. In fact, if you remember Part II in this series (*LJ*, March 2010), I *already* created a client certificate and key, right after I created the server credentials.

Because this process is both important and simple, let's take a minute to review it. I'm skipping the process of setting up and creating the Certificate Authority itself, as that applies only to server setup (you should do that only once, on the server). So, assuming you've got a working CA set up on your OpenVPN server as described in Part II of this article, follow these steps to use OpenVPN's pkitool script to create a new client certificate:

1) su to root:

```
bash-$ su
```

2) Change your working directory to /etc/openvpn/2.0:

```
bash-# cd /etc/openvpn/2.0
```

3) Declare some PKI-related environment variables stored in the file vars:

```
bash-# source ./vars
```

4) Create the new certificate:

```
bash-# ./pkitool --pass minion
```

In step 4, the string `minion` is the name (the "common name", in x.509 parlance) of the host or user whose certificate you're creating. After issuing this command, you'll be prompted twice to type the certificate's passphrase.

The output of this command takes the form of three files: ./keys/minion.csr, ./keys/minion.crt and ./keys/minion.key. Only the last two are important for the purposes of this article: the new client certificate and client key, respectively.

Of the helper files related to crypto that were created when you set up the OpenVPN server, your client certificate and key are the only two unique to the client; ca.crt and ta.key are used on the server and on all clients that connect to it. Note also that although the client certificate (minion.crt) contains no private data, the client key minion.key and the TLS authentication key ta.key both should be kept secret through local file permissions and by handling them carefully.

For example, you should never e-mail any client key or TA key in clear text (note that using an https:// URL for Webmail access doesn't count as message encryption). You should use S/MIME or PGP e-mail encryption if you need to mail keys to users.

If you use a USB drive or other physical media to distribute keys, you should either deliver it in person or use a trusted courier to deliver it, and users should be instructed either to destroy or erase the media after installing their keys, or keep the media under lock and key. Although having a passphrase-protected client key *should* make it hard for an attacker to use an intercepted key file, it's no guarantee! Under no circumstances should you issue blank-passphrase client certificates for any VPN scenario.

Speaking of the client key's passphrase, you also should take care in how you transmit that passphrase to the key's user. Because it isn't good policy for any system administrator to know users' passphrases in any context, users may afterward want to change the key's passphrase. The simplest way for users to do so is via the openssl command, like so:

```
bash-$ openssl rsa -in minion.key -out minion.key -aes192
```

The user does *not* need to be root to do this, provided the proper file permissions are set on minion.key (users should have read/write access on their own keys). After entering this command, users will be prompted for the key file's old passphrase and then twice for the new passphrase.

Once users have copied the files ca.crt, client.crt, client.key and ta.key over to their client system's /etc/openvpn/ directory, they should make sure they have the correct file permissions set. The two .crt files should be world-readable, but only owner-writable (that is, `-rw-r--r--`). The two .key files, however, should be only owner-readable/writable (that is, `-rw-------`).

All four files should be owned by root, assuming your users have root on their own Linux systems. (Setting up OpenVPN for nonroot users and the security challenges of doing so are beyond this article's scope.)

Now that you're clear on how to generate and manage client certificate/key pairs, let's continue working our way down Listing 2. The `ca`, `cert` and key directives specify the paths of your CA key file, client certificate file and client key file, respectively. In Listing 2 the values for these parameters are all just filenames, without their full paths specified. This implies that those three files are in the same directory as the client configuration file itself.

So, unlike on the server, where I left all the certificates and keys in /etc/openvpn/2.0/keys and, therefore, specified a CA certificate path of 2.0/keys/ca.crt, on the client system, you can get by with simply `ca.crt` if the file ca.crt, like the configuration file client.ovpn, is kept in the directory /etc/openvpn/.

The `ns-cert-type server` directive says what type of certificate your client should accept. Because in this example I'm dealing with multiple clients connecting back to a server, in Listing 2, it's set to `server`. This will prevent some other client from impersonating the server in a man-in-the-middle attack; the server's certificate, while signed by the same CA as its clients, has attributes that identify it as a server certificate, not another client certificate.

The last directive in the certificate/key-file portion of Listing 2 is `tls-auth ta.key 1`, which tells OpenVPN to use the file ta.key to add an extra layer of authentication on your VPN tunnel by requiring all packets in the TLS handshake phase at the beginning of each tunnel session to be signed with the specified TLS Authentication key. After the name of the TLS Authentication key (ta.key), specify a number telling in which "direction" to use this file: "0" for the server and "1" for clients.

## Other Client Settings

Nearly all of the rest of the directives in Listing 2 are ones I already covered in Parts II and III of this series when dissecting Listing 1, the server configuration file. The client-side settings for those directives are even the same as specified on the server.

In the case of `user nobody` and `group nogroup`, which tell the openvpn command to run with unprivileged user and group identities after initialization, make sure the user account nobody and the group nogroup exist on your client system. Because both typically exist on most Linux systems, they probably do already. If not, you either can create them or change the directives to point to some other, existing dæmon account or group name. In no event should you change either to any user or group name used for actual human-use login accounts!

The only other directive worth mentioning here is `mute-replay-warnings`, which I didn't include in the server.ovpn file. Declaring this directive (without any argument) tells OpenVPN not to log events associated with its anti-packet-replay mechanism, which tends to trigger false alarms if you connect to a wireless network. It won't turn off the actual anti-replay protection; it just suppresses associated local log entries.

## Initiating a Tunnel

You've created a client configuration file and put it into your client system's /etc/openvpn directory. You've copied over your CA certificate file, client certificate and key files, and your TA key file. It's time to connect to the server.

Assuming your client system is running Linux (specifically Ubuntu), and assuming that, as in Listing 2, your client configuration file is called client.ovpn, follow these steps the first time you connect to your server:

1) Change your working directory to /etc/openvpn:

```
bash-$ cd /etc/openvpn
```

2) Run OpenVPN like this:

```
bash-$ sudo openvpn --config ./client.ovpn
```

3) When prompted, enter your client key's passphrase.

```
Enter Private Key Password: Your passphrase here
```

Note that in step 2 you started OpenVPN *without* the `--daemon` directive, leaving it running in the foreground. If everything works without a hitch, the next time you start OpenVPN, you can use `sudo openvpn --daemon --config ./client.ovpn`, in which case OpenVPN will run silently after asking you for your client key passphrase. On my Ubuntu client system, OpenVPN logs to the file /var/log/syslog when running in dæmon mode.

## Troubleshooting

Hopefully, at this point you've got a working VPN tunnel back to your server. If you don't though, two different mistakes have caused the majority of my own problems using OpenVPN.

First, your tunnel will fail if you fail to copy all necessary files into /etc/openvpn: client configuration file, CA certificate, client certificate, client key and TLS Authentication key. These files also must have appropriate permissions set, and you must run the openvpn command with the appropriate level of privilege (that is, root privilege).

Second, firewall rules on both server and clients must be either disabled (left open) or reconfigured to allow OpenVPN traffic to pass. Telling you how to write such rules easily could occupy half or more of an entire article, especially if I were to cover the art of forcing certain types of traffic to use the tunnel. Suffice it to say, for now, check your iptables rules before you even bother running OpenVPN in server or client mode.

## Cross-Platform Notes

In writing this article, I tested both an Ubuntu client and a Windows XP client. Getting the Windows XP client to connect properly was no more difficult than on Ubuntu. It was a simple matter of placing the correct files in the correct directory and tweaking my Windows firewall configuration a little.

On Windows clients, the `user` and `group` directives have no effect, as OpenVPN's self-demotion feature is not supported in Windows. Other than that, however, I found the Windows version of OpenVPN to work in a very similar manner as the Linux version.

## Conclusion

And that, dear readers, is how you configure OpenVPN to allow yourself to connect back to your home network from an untrusted remote site. This process is, in practice, much easier than my taking four months to describe it implies. Hopefully, my line-by-line dissections of the two configuration files have given you a strong enough understanding of how OpenVPN works for you to explore other usage scenarios.

I may devote one more column to this topic, because Virtual Private Networks are such a powerful tool, and these four installments have covered only a small subset of its potential. Regardless, I hope you've found this series useful and that you have success in your own Virtual Private Linux endeavors. Until next time, be safe!■

**Mick Bauer (darth.elmo@wiremonkeys.org) is Network Security Architect for one of the US's largest banks. He is the author of the O'Reilly book *Linux Server Security*, 2nd edition (formerly called *Building Secure Servers With Linux*), an occasional presenter at information security conferences and composer of the "Network Engineering Polka".**

## Resources

Official OpenVPN Home Page: **www.openvpn.net**

Ubuntu Community OpenVPN Page: **https://help.ubuntu.com/community/OpenVPN**