Linux VPNs with OpenVPN, Part III

MICK BAUER

Secure remote networking, the OpenVPN way.

In my previous two columns, I began a series on building Linux-based Virtual Private Network (VPN) solutions using OpenVPN. When I left off last time, I had gotten as far through the OpenVPN server configuration process as creating a simple Public Key Infrastructure (PKI), using it to generate server and client certificates, and creating a few other "support" files involved in building OpenVPN tunnels. In so doing, I worked my way down just the first third or so of the example OpenVPN server configuration, but those PKI/crypto-related configuration parameters represent the most complicated part of OpenVPN configuration tasks.

This month, I describe the rest of that server configuration file and show a corresponding OpenVPN client configuration file (which I'll dissect next month). I also show how to start both server and client processes, although debugging, firewall considerations and other finer points also will need to wait until my next column.

Have no fear—I think you'll find this installment to be plenty action-packed in its own right. Let's get to 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  # This file should be kept 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
```

OpenVPN Server Configuration, Continued

Normally at this point in a multipart series, I'd review at least some details from the prior month's column, but that won't work this time. Last month's article covered a lot of ground, and this month's needs to cover still more. Suffice it to say that I began dissecting an example OpenVPN server configuration file, /etc/openvpn/server.ovpn (Listing 1).

I got as far as generating the files referenced in the `ca`, `cert`, `key`, `dh` and `tls-auth` lines, using a combination of OpenVPN "easy-rsa" helper scripts (located in /usr/share/doc/openvpn/examples/easy-rsa/2.0) and the commands openvpn and openssl. I'm going to continue describing Listing 1's parameters, assuming that the aforementioned certificate, key and other helper files are in place.

So, having set up the basic port/protocol/device settings and cryptography-related settings, let's move on to settings that will determine what happens once a client successfully establishes an authenticated, encrypted tunnel. The first such setting is `server`.

`server` actually is a helper directive. It expands to an entire block of other parameters. Rather than slogging through all those additional parameters, let's just say that the server directive takes two parameters: a network address and a netmask. For each tunnel established by clients on the port we specified earlier, the OpenVPN server process will carve a little 30-bit subnet from the specified IP space, assign itself the first host IP address in that subrange as its local tunnel endpoint and assign the other host IP in the 30-bit subnet to the connecting client as its remote tunnel endpoint.

In the example, I've specified the network address 10.31.33.0 with a netmask of 255.255.255.0,

which translates to the range of IP addresses from 10.31.33.1 to 10.31.33.254. When the first tunnel is established, the server will use 10.31.33.1 as its local tunnel endpoint address and assign 10.31.33.2 to the client to use as the remote tunnel endpoint address. (10.31.33.0 is that subnet's network address, and 10.31.33.3 is its broadcast address.)

For the next client to connect, the server will use 10.31.33.5 as its tunnel endpoint and will assign 10.31.33.6 as the client's tunnel endpoint (with 10.31.33.4 and 10.31.33.7 as network and broadcast addresses, respectively). Get it?

This isn't the most efficient use of an IP range. The server needs a different local IP address for *each* tunnel it builds, and for each tunnel, the server essentially wastes two others (for network and broadcast addresses). Preceding the server directive with the line `topology subnet` will cause the server to use the first IP in its `server [network address] [netmask]` range for its local tunnel IP for *all* tunnels and for client tunnel-endpoint IPs to be allocated from the *entire* remainder of possible IPs in that range, as though all remote tunnel endpoints were IP addresses on the same LAN.

This isn't the default behavior, because it's new to OpenVPN 2.1. The "subnet" topology isn't supported by earlier versions or by Windows clients using version 8.1 or lower of the TAP-Win32 driver. Note that if undeclared (as in Listing 1), the `topology` parameter has a default value of `net30`, which results in server's specified IP range being split up into 30-bit subnets as described above.

Continuing on in Listing 1, next comes `ifconfig-pool-persist`, which specifies a file in which to store associations between tunnel clients' Common Names (usually their hostnames, as specified in their respective client certificates) and the IP addresses the server assigns to their tunnels. Although this doesn't guarantee a given client will receive the same tunnel IP every time it connects, it does allow clients to use the `--persist-tun` option in their client configurations, which keeps tunnel sessions open across disruptions in service (OpenVPN server dæmon restarts, network problems and so forth).

Next comes the statement `push "redirect-gateway def1 bypass-dhcp"`. The `push` directive causes the subsequent double-quotation-mark-enclosed string to be run on the client as though it was part of the client's local configuration file. In this case, the server will push the `redirect-gateway` parameter to all clients, with the effect that each time a client connects, the client's local default gateway, DNS servers and other network parameters normally provided by DHCP will be overridden by the server's settings for those things.

This effectively enforces a VPN policy of "local-subnet-only split tunneling". For those of you new to VPNs, a split tunnel configuration is one in which clients can use their VPN tunnels to connect to some things and their local (non-tunneled) Internet connection to connect to other things.

As I've said in previous columns though, forcing clients to use the *remote* network's infrastructure (DNS servers, Internet uplink and so forth) makes it much harder for attackers connected to a client's local network, which might be an untrusted environment like a coffee-shop wireless hotspot, to perform various kinds of eavesdropping, session-hijacking and man-in-the-middle attacks.

Even with this setting, a client still will be *able* to connect to some things on the local network. It just won't be able to use it as a *route* for anything but connecting back to your OpenVPN server. Again, it's good policy to configure clients to leverage as much of your trusted network's infrastructure as possible.

After the `push "redirect-gateway..."` directive comes `keepalive 10 120`. Like `server`, `keepalive` is a helper directive that expands to a list of other parameters. Again for the sake of brevity, let me summarize the effect of the example line: every ten seconds, the server will check to see that each client is still connected, and if no reply is received from a given client over any 120-second period, it will assume that client is unreachable at its last known IP address.

For example, if the server sends a query to a particular tunnel client at 9:00:00 and gets a reply, but another at 9:00:10 to which there's no reply, and also receives no reply to any of 11 more queries sent out at 9:00:20, 9:00:30 and so on until 9:02:00, then at 9:02:00 (after 120 seconds of no replies), the server will conclude the client system is unreachable.

At this point, the server will attempt to re-resolve the remote client's name, on the assumption that its IP address may have changed (due to DHCP lease renewal, for example) and, thus, re-establish the tunnel session.

The aforementioned infrastructure settings, such as DNS servers, by the way, will be read by the server's openvpn process from /etc/resolv.conf, the server's running routing table and so forth—no OpenVPN configuration parameters are necessary for those settings unless you want them to be different from the server's. (For now, let's assume you don't!)

I just spent a fair amount of ink on only a handful of settings. But I think this is warranted given that `server` and `keepalive` are helper directives that expand to many more settings and given that we're now done with the network configuration portion of our server configuration.

The next parameter is a simple one: `cipher BF-CBC`, which specifies that each tunnel's data payload will be encrypted with the Blowfish cipher, using 128-bit keys, in Cipher Block Chaining mode (CBC mode makes it harder for an attacker to brute-force-decrypt isolated parts of a given session). BF-CBC is the default setting for `cipher`, so technically, I don't need to specify this, but it's an important setting. You can use the command `openvpn --show-ciphers` to see a list of all supported cipher values and their default key sizes.

`comp-lzo` is even simpler. It tells OpenVPN to compress all session data using the LZO compression algorithm, unless a given portion of data appears to be compressed already (for example, if a JPEG image or a ZIP file is being transferred), in which case OpenVPN won't compress until it detects a return to noncompressed session content. This adaptive behavior helps minimize the data padding that results from trying to compress already-compressed data. Because LZO is a fast algorithm, this is a good setting. Its cost in CPU overhead is generally more than compensated for by the amount of network bandwidth (and, thus, other CPU cycles) it conserves.

The next setting, `max-clients 2`, specifies that a maximum of two tunnels may be active at one time. If you have only one or two users, there's no good reason to allow more than one or two concurrent tunnels. In my own testing, however, I've found that setting this all the way down to 1 can cause problems even if you have only one user, probably due to how OpenVPN handles tunnel persistence (see `keepalive` above).

The next four settings are interrelated. `user` and `group` specify the names of an unprivileged user account and group (`nobody` and `nogroup`, respectively), for the OpenVPN server dæmon to demote itself to after opening necessary tun/tap devices, reading its configuration file, certificates and keys, and other root-only startup actions.

For this to work properly, you also need to set `persist-key` and `persist-tun`. `persist-key` causes OpenVPN to keep key file contents cached in memory across dæmon interruptions (like those caused by tunnels being broken and re-established). `persist-tun` causes OpenVPN to keep any tun/tap devices that were opened on startup, open across the same kinds of dæmon restarts.

With `user` and `group` set to unprivileged user and group, if you were to skip declaring `persist-key` or `persist-tun`, the OpenVPN dæmon would lack the necessary privileges to re-read protected key files or re-open the tun or tap device.

You could, of course, skip the user and group settings. Those settings, however, lessen the impact of some unforeseen buffer-overflow vulnerability. It can make the difference from an attacker gaining an unprivileged shell and gaining a root shell. Unfortunately, you can't assume that just because OpenVPN has had a good track record so far with respect to lacking many significant security vulnerabilities, that it never will have any!

The last three settings in Listing 1 concern logging. `status` specifies a file to which OpenVPN will write dæmon status updates periodically, regardless of actual activity. Unlike most log files, each time this file is updated, OpenVPN will overwrite the previous message. This is what the file /etc/openvpn/openvpn-status.log on my OpenVPN server says right now:

```
OpenVPN CLIENT LIST
Updated,Fri Jan  1 21:55:11 2010
Common Name,Real Address,Bytes Received,Bytes Sent,Connected Since
minion2,192.168.20.1:36491,125761,103329,Fri Jan  1 17:56:21 2010
ROUTING TABLE
Virtual Address,Common Name,Real Address,Last Ref
10.31.33.6,minion2,192.168.20.1:36491,Fri Jan  1 20:54:03 2010
GLOBAL STATS
Max bcast/mcast queue length,0
END
```

As you can see, there's only one client currently connected (`minion2`), with one corresponding route table entry.

Moving on back in Listing 1, `verb 3` sets the overall logging-verbosity level to 3 out of a possible range of 0 (no logging except major errors) and 11 (the most verbose debugging output possible). The default value is 1, but 3 is much more useful for getting things set up and working properly, without presenting any particular danger of log files growing too huge too quickly.

This is especially true with `mute 20` set, which tells OpenVPN never to log the same message (in a given event category) more than 20 times in a row.

On my Ubuntu system, OpenVPN writes all its messages to /var/log/daemon if the `openvpn` command is executed with the `--daemon` flag, which causes it to run as a background (dæmon) process. If you run `openvpn` without `--daemon`, it runs in the foreground and logs all messages to the console or terminal window you started it in (tying up that console in the process, but this is a very handy way to run OpenVPN during initial setup and testing).

Running OpenVPN as a Server Dæmon

Now that I've covered a sample server configuration file in depth, let's fire up our OpenVPN dæmon in server mode! This, as you'll see, is the easy part.

OpenVPN uses a single command, `openvpn`, for everything. Precisely what any given OpenVPN instance does depends on how you start it. As

you've already seen, some startup parameters, like `--show-ciphers`, cause the openvpn command to give certain information and then exit. Other parameters tell it to remain active, listening for incoming client connections (`--mode server`) or attempting to establish and maintain a tunnel to some server, as a client (`--mode client`).

If you execute `openvpn` with the `--config` parameter followed by the name of a configuration file, OpenVPN will start itself configured with all parameters in that file. For example, you could create a configuration file containing just the parameter `show-ciphers` (parameters must start with a -- if specified in a command line, but the -- is omitted for all parameters within configuration files).

More commonly, as with Listing 1, we use configuration files for server-mode and client-mode startup. I mentioned that the `server` helper directive expands into a list of other parameters; the first of these is `mode server`.

Thus, to start OpenVPN as a persistent server dæmon running the configuration file /etc/openvpn/server.ovpn, shown in Listing 1, use this command:

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

Note the relative path for the file server.ovpn. If that file resides in /etc/openvpn, you'd need to run the above command from within that directory. Note also the use of sudo. On non-Ubuntu systems, you might instead `su` to root before running this command. Regardless, OpenVPN must be run as root in order to read its server key file, to open the tun device and so forth, even though as configured in Listing 1 it subsequently will demote itself to user nobody and group ID nogroup.

Did you notice I omitted the `--daemon` flag on that command line? Again, you can use that flag to tell OpenVPN to run in the background (like a quiet, well-behaved dæmon) and log its messages to /var/log/daemon.log, but you first may want to make sure everything's working properly.

Configuring the Client

At this point, I had hoped I'd be able to give you a detailed walk-through of client configuration, but I'm out of space for now, so that will need to wait until next time. But, I won't leave you completely hanging. Listing 2 shows a sample client configuration file, client.ovpn, that corresponds to Listing 1's server.ovpn file.

Much of this should be familiar. Other parts you can figure out via the openvpn(8) man page. In the meantime, feel free to experiment. To run OpenVPN in client mode on a client computer,

Listing 2. Client's iwazaru.ovpn File

```
client
dev tun
proto udp

remote 1.2.3.4 1194

resolv-retry infinite
nobind

user nobody
group nogroup
persist-key
persist-tun

mute-replay-warnings

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

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

cipher BF-CBC
comp-lzo

verb 3
mute 20
```

use this command:

```
sudo openvpn --config ./iwazaru.ovpn --daemon openvpn-client
```

One parting tip for you experimenters: you'll need to disable or reconfigure any local iptables (firewall) rules you've got running on either your server or client systems. I'll discuss iptables considerations in the next column in this series, and I'll continue where we left off this time. Until then, 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**