openvpn examples

Secure IP tunnel daemon

Manual section: 5
Manual group: Configuration files

INTRODUCTION

This man page gives a few simple examples to create OpenVPN setups and configuration files.

Small OpenVPN setup with peer-fingerprint

This section consists of instructions how to build a small OpenVPN setup with the `peer-fingerprint` option. This has the advantage of being easy to setup and should be suitable for most small lab and home setups without the need for a PKI. For bigger scale setup setting up a PKI (e.g. via easy-rsa) is still recommended.

Both server and client configuration can be further modified to customise the setup.

Server setup

1. Install openvpn

   Compile from source-code (see *INSTALL* file) or install via a distribution (apt/yum/ports) or via installer (Windows).

2. Generate a self-signed certificate for the server:

   ```
   openssl req -x509 -newkey ec:<(openssl ecparam -name secp384r1) -keyout server.key -out server.crt -nodes -sha256 -days 3650 -sub
   ```

3. Generate SHA256 fingerprint of the server certificate

   Use the OpenSSL command line utility to view the fingerprint of just created certificate:

   ```
   openssl x509 -fingerprint -sha256 -in server.crt -noout
   ```

   This output something similar to:

   ```
   SHA256 Fingerprint=00:11:22:33:44:55:66:77:88:99:aa:bb:cc:dd:ee:ff:00:11:22:33:44:55:66:77:88:99:aa:bb:cc:dd:ee:ff
   ```

4. Write a server configuration (*server.conf*):

   ```
   # The server certificate we created in step 1
   cert server.crt
   key server.key

   dh none
   dev tun

   # Listen on IPv6+IPv4 simultaneously
   proto udp6

   # The ip address the server will distribute
   server 10.8.0.0 255.255.255.0
   server-ipv6 fd00:6f76:706e::/64

   # A tun-mtu of 1400 avoids problems of too big packets after VPN encapsulation
   tun-mtu 1400

   # The fingerprints of your clients. After adding/removing one here restart the
   # server
   <peer-fingerprint>
   </peer-fingerprint>

   # Notify clients when you restart the server to reconnect quickly
   explicit-exit-notify 1

   # Ping every 60s, restart if no data received for 5 minutes
   keepalive 60 300
   ```

5. Add at least one client as described in the client section.

6. Start the server.
   - On systemd based distributions move *server.crt*, *server.key* and *server.conf* to `/etc/openvpn/server` and start it via systemctl

     ```
     sudo mv server.conf server.key server.crt /etc/openvpn/server

     sudo systemctl start openvpn-server@server
     ```

Adding a client

1. Install OpenVPN

2. Generate a self-signed certificate for the client. In this example the client name is alice. Each client should have a unique name. Replace alice with a different name for each client.

   ```
   openssl req -x509 -newkey ec:<(openssl ecparam -name secp384r1) -nodes -sha256 -days 3650 -subj '/CN=alice'
   ```

   This generate a certificate and a key for the client. The output of the command will look something like this:

   ```
   -----BEGIN PRIVATE KEY-----
   [base64 content]
   -----END PRIVATE KEY-----
   -----
   -----BEGIN CERTIFICATE-----
   [base 64 content]
   -----END CERTIFICATE-----
   ```

3. Create a new client configuration file. In this example we will name the file *alice.ovpn*:

   ```
   # The name of your server to connect to
   remote yourserver.example.net
   client
   # use a random source port instead the fixed 1194
   nobind

   # Uncomment the following line if you want to route
   # all traffic via the VPN
   # redirect-gateway def1 ipv6

   # To set a DNS server
   # dhcp-option DNS 192.168.234.1

   <key>
   -----BEGIN PRIVATE KEY-----
   [Insert here the key created in step 2]
   -----END PRIVATE KEY-----
   </key>
   <cert>
   -----BEGIN CERTIFICATE-----
   [Insert here the certificate created in step 2]
   -----END CERTIFICATE-----
   </cert>

   # This is the fingerprint of the server that we trust. We generated this fingerprint
   # in step 2 of the server setup
   peer-fingerprint 00:11:22:33:44:55:66:77:88:99:aa:bb:cc:dd:ee:ff:00:11:22:33:44:55:66:77:88:99:aa:bb:cc:dd:ee:ff

   # The tun-mtu of the client should match the server MTU
   tun-mtu 1400
   dev tun
   ```

4. Generate the fingerprint of the client certificate. For that we will let OpenSSL read the client configuration file as the x509 command will ignore anything that is not between the begin and end markers of the certificate:

   ```
   openssl x509 -fingerprint -sha256 -noout -in alice.ovpn
   ```

   This will again output something like

   ```
   SHA256 Fingerprint=ff:ee:dd:cc:bb:aa:99:88:77:66:55:44:33:22:11:00:ff:ee:dd:cc:bb:aa:99:88:77:66:55:44:33:22:11:00
   ```

5. Edit the *server.conf* configuration file and add this new client fingerprint as additional line between `<peer-fingerprint>` and `</peer-fingerprint>`

   After adding *two* clients the part of configuration would look like this:

   ```
   <peer-fingerprint>
   ff:ee:dd:cc:bb:aa:99:88:77:66:55:44:33:22:11:00:ff:ee:dd:cc:bb:aa:99:88:77:66:55:44:33:22:11:00
   99:88:77:66:55:44:33:22:11:00:ff:ee:dd:cc:bb:aa:99:88:77:66:55:44:33:22:11:00:88:77:66:55:44:33
   </peer-fingperint>
   ```

6. (optional) if the client is an older client that does not support the `peer-fingerprint` (e.g. OpenVPN 2.5 and older, OpenVPN Connect 3.3 and older), the client config *alice.ovpn* can be modified to still work with these clients.

   Remove the line starting with `peer-fingerprint`. Then add a new `<ca>` section at the end of the configuration file with the contents of the `server.crt` created in step 2 of the server setup. The end of *alice.ovpn* file should like:

   ```
   [...]  # Beginning of the file skipped
   </cert>

   # The tun-mtu of the client should match the server MTU
   tun-mtu 1400
   dev tun

   <ca>
   [contents of the server.crt]
   </ca>
   ```

   Note that we put the `<ca>` section after the `<cert>` section to make the fingerprint generation from step 4 still work since it will only use the first certificate it finds.

7. Import the file into the OpenVPN client or just use the `openvpn alice.ovpn` to start the VPN.

EXAMPLES

Prior to running these examples, you should have OpenVPN installed on two machines with network connectivity between them. If you have not yet installed OpenVPN, consult the INSTALL file included in the OpenVPN distribution.

Firewall Setup:

If firewalls exist between the two machines, they should be set to forward the port OpenVPN is configured to use, in both directions. The default for OpenVPN is 1194/udp. If you do not have control over the firewalls between the two machines, you may still be able to use OpenVPN by adding `--ping 15` to each of the `openvpn` commands used below in the examples (this will cause each peer to send out a UDP ping to its remote peer once every 15 seconds which will cause many stateful firewalls to forward packets in both directions without an explicit firewall rule).

Please see your operating system guides for how to configure the firewall on your systems.

VPN Address Setup:

For purposes of our example, our two machines will be called `bob.example.com` and `alice.example.com`. If you are constructing a VPN over the internet, then replace `bob.example.com` and `alice.example.com` with the internet hostname or IP address that each machine will use to contact the other over the internet.

Now we will choose the tunnel endpoints. Tunnel endpoints are private IP addresses that only have meaning in the context of the VPN. Each machine will use the tunnel endpoint of the other machine to access it over the VPN. In our example, the tunnel endpoint for bob.example.com will be 10.4.0.1 and for alice.example.com, 10.4.0.2.

Once the VPN is established, you have essentially created a secure alternate path between the two hosts which is addressed by using the tunnel endpoints. You can control which network traffic passes between the hosts (a) over the VPN or (b) independently of the VPN, by choosing whether to use (a) the VPN endpoint address or (b) the public internet address, to access the remote host. For example if you are on bob.example.com and you wish to connect to `alice.example.com` via `ssh` without using the VPN (since **ssh** has its own built-in security) you would use the command `ssh alice.example.com`. However in the same scenario, you could also use the command `telnet 10.4.0.2` to create a telnet session with alice.example.com over the VPN, that would use the VPN to secure the session rather than `ssh`.

You can use any address you wish for the tunnel endpoints but make sure that they are private addresses (such as those that begin with 10 or 192.168) and that they are not part of any existing subnet on the networks of either peer, unless you are bridging. If you use an address that is part of your local subnet for either of the tunnel endpoints, you will get a weird feedback loop.

Example 1: A simple tunnel without security (not recommended)

On bob:

```
openvpn --remote alice.example.com --dev tun1 \
         --ifconfig 10.4.0.1 10.4.0.2 --verb 9
```

On alice:

```
openvpn --remote bob.example.com --dev tun1 \
         --ifconfig 10.4.0.2 10.4.0.1 --verb 9
```

Now verify the tunnel is working by pinging across the tunnel.

On bob:

```
ping 10.4.0.2
```

On alice:

```
ping 10.4.0.1
```

The `--verb 9` option will produce verbose output, similar to the `tcpdump`(8) program. Omit the `--verb 9` option to have OpenVPN run quietly.

Example 2: A tunnel with self-signed certificates and fingerprint

First build a self-signed certificate on bob and display its fingerprint.

```
openssl req -x509 -newkey ec:<(openssl ecparam -name secp384r1) -keyout bob.pem -out bob.pem -nodes -sha256 -days 3650 -subj '/CN=bob'
openssl x509 -noout -sha256 -fingerprint -in bob.pem
```

and the same on alice:

```
openssl req -x509 -newkey ec:<(openssl ecparam -name secp384r1) -keyout alice.pem -out alice.pem -nodes -sha256 -days 3650 -subj '/CN=
openssl x509 -noout -sha256 -fingerprint -in alice.pem
```

These commands will build a text file called `bob.pem` or `alice.pem` (in ascii format) that contain both self-signed certificate and key and show the fingerprint of the certificates. Transfer the fingerprints over a secure medium such as by using the `scp`(1) or `ssh`(1) program.

On bob:

```
openvpn --ifconfig 10.4.0.1 10.4.0.2 --tls-server --dev tun --dh none \
        --cert bob.pem --key bob.pem --cipher AES-256-GCM \
        --peer-fingerprint "$fingerprint_of_alices_cert"
```

On alice:

```
openvpn --remote bob.example.com --tls-client --dev tun1   \
        --ifconfig 10.4.0.2 10.4.0.1 --cipher AES-256-GCM  \
        --cert alice.pem --key alice.pem                   \
        --peer-fingerprint "$fingerprint_of_bobs_cert"
```

Now verify the tunnel is working by pinging across the tunnel.

On bob:

```
ping 10.4.0.2
```

On alice:

```
ping 10.4.0.1
```

Note: This example use a elliptic curve (*secp384*), which allows `--dh` to be set to `none`.

Example 3: A tunnel with full PKI and TLS-based security

For this test, we will designate `bob` as the TLS client and `alice` as the TLS server.

Note:
The client or server designation only has meaning for the TLS subsystem. It has no bearing on OpenVPN's peer-to-peer, UDP-based communication model.*

First, build a separate certificate/key pair for both bob and alice (see above where `--cert` is discussed for more info). Then construct Diffie Hellman parameters (see above where `--dh` is discussed for more info). You can also use the included test files `client.crt`, `client.key`, `server.crt`, `server.key` and `ca.crt`. The `.crt` files are certificates/public-keys, the `.key` files are private keys, and `ca.crt` is a certification authority who has signed both `client.crt` and `server.crt`. For Diffie Hellman parameters you can use the included file `dh2048.pem`.

WARNING:
All client, server, and certificate authority certificates and keys included in the OpenVPN distribution are totally insecure and should be used for testing only.

On bob:

```
openvpn --remote alice.example.com --dev tun1    \
        --ifconfig 10.4.0.1 10.4.0.2             \
        --tls-client --ca ca.crt                 \
        --cert client.crt --key client.key       \
        --reneg-sec 60 --verb 5
```

On alice:

```
openvpn --remote bob.example.com --dev tun1      \
        --ifconfig 10.4.0.2 10.4.0.1             \
        --tls-server --dh dh1024.pem --ca ca.crt \
        --cert server.crt --key server.key       \
        --reneg-sec 60 --verb 5
```

Now verify the tunnel is working by pinging across the tunnel.

On bob:

```
ping 10.4.0.2
```

On alice:

```
ping 10.4.0.1
```

Notice the `--reneg-sec 60` option we used above. That tells OpenVPN to renegotiate the data channel keys every minute. Since we used `--verb 5` above, you will see status information on each new key negotiation.

For production operations, a key renegotiation interval of 60 seconds is probably too frequent. Omit the `--reneg-sec 60` option to use OpenVPN's default key renegotiation interval of one hour.

Routing:

Assuming you can ping across the tunnel, the next step is to route a real subnet over the secure tunnel. Suppose that bob and alice have two network interfaces each, one connected to the internet, and the other to a private network. Our goal is to securely connect both private networks. We will assume that bob's private subnet is *10.0.0.0/24* and alice's is *10.0.1.0/24*.

First, ensure that IP forwarding is enabled on both peers. On Linux, enable routing:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

This setting is not persistent. Please see your operating systems documentation how to properly configure IP forwarding, which is also persistent through system boots.

If your system is configured with a firewall. Please see your operating systems guide on how to configure the firewall. You typically want to allow traffic coming from and going to the tun/tap adapter OpenVPN is configured to use.

On bob:

```
route add -net 10.0.1.0 netmask 255.255.255.0 gw 10.4.0.2
```

On alice:

```
route add -net 10.0.0.0 netmask 255.255.255.0 gw 10.4.0.1
```

Now any machine on the *10.0.0.0/24* subnet can access any machine on the *10.0.1.0/24* subnet over the secure tunnel (or vice versa).

In a production environment, you could put the route command(s) in a script and execute with the `--up` option.