OpenVPN

Related articles

- OpenVPN (client) in Linux containers
- OpenVPN (server) in Linux containers
- Easy-RSA

This article describes a basic installation and configuration of **OpenVPN (http://openvpn.net)**, suitable for private and small business use. For more detailed information, please see the **OpenVPN 2.4 man page (https://community.openvpn.net/openvpn/wiki/Openvpn24ManPage)** and the **OpenVPN documentation (http://openvpn.net/index.php/open-source/documentation)**. OpenVPN is a robust and highly flexible **VPN** daemon. It supports **SSL/TLS** security, **Ethernet bridging**, **TCP** or **UDP tunnel transport** through **proxies** or **NAT**. Additionally it has support for dynamic IP addresses and **DHCP**, scalability to hundreds or thousands of users, and portability to most major OS platforms.

OpenVPN is tightly bound to the **OpenSSL (http://www.openssl.org)** library, and derives much of its crypto capabilities from it. It supports conventional encryption using a **pre-shared secret key** (Static Key mode) or **public key security** (**SSL/TLS** mode) using client & server certificates. Additionally it supports unencrypted TCP/UDP tunnels.

OpenVPN is designed to work with the **TUN/TAP** virtual networking interface that exists on most platforms. Overall, it aims to offer many of the key features of **IPSec** but with a relatively lightweight footprint. OpenVPN was written by James Yonan and is published

Contents

Install OpenVPN

Install the **`openvpn (https://www.archlinux.org/packages/?name=openvpn)`** package, which provides both server and client mode.

Kernel configuration

OpenVPN requires TUN/TAP support, which is already configured in the default kernel. Users of custom kernel should make sure to enable the `tun` module:

```
Kernel config file
```

```
Device Drivers
 --> Network device support
   [M] Universal TUN/TAP device driver support
```

Read **Kernel modules** for more information.

Connect to a VPN provided by a third party

To connect to a VPN service provided by a third party, most of the following can most likely be ignored, especially regarding server setup. Begin with **#The client config profile** and skip ahead to **#Starting OpenVPN** after that. One should use the provider certificates and instructions, see **Category:VPN providers** for examples that can be adapted to other providers. **OpenVPN (client) in Linux containers** also has general applicable instructions, while it goes a step further by isolating an OpenVPN client process into a container.

Note: Most free VPN providers will (often only) offer **PPTP**, which is drastically easier to setup and configure, but **not secure (http://poptop.sourceforge.net/dox/protocol-security.phtml)**.

Create a Public Key Infrastructure (PKI) from scratch

When setting up an OpenVPN server, users need to create a **Public Key Infrastructure (PKI)** which is detailed in the **Easy-RSA** article. Once the needed certificates, private keys, and associated files are created via following the steps in the separate article, one should have 5 files in `/etc/openvpn/server` at this point:

```
ca.crt
dh.pem
servername.crt
servername.key
ta.key
```

Alternatively, as of OpenVPN 2.4, one can use Easy-RSA to generate certificates and keys using elliptic curves. See the OpenVPN documentation for details.

A basic L3 IP routing configuration

Note: Unless otherwise explicitly stated, the rest of this article assumes a basic L3 IP routing configuration.

OpenVPN is an extremely versatile piece of software and many configurations are possible, in fact machines can be both servers and clients.

With the release of v2.4, server configurations are stored in `/etc/openvpn/server` and client configurations are stored in `/etc/openvpn/client` and each mode has its own respective systemd unit, namely, `openvpn-client@.service` and

`openvpn-server@.service`.

Example configuration

The OpenVPN package comes with a collection of example configuration files for different purposes. The sample server and client configuration files make an ideal starting point for a basic OpenVPN setup with the following features:

- Uses **Public Key Infrastructure (PKI)** for authentication.
- Creates a VPN using a virtual TUN network interface (OSI L3 IP routing).
- Listens for client connections on UDP port 1194 (OpenVPN's official IANA port number**[1] (https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml?search=openvpn)**).
- Distributes virtual addresses to connecting clients from the 10.8.0.0/24 subnet.

For more advanced configurations, please see the **`openvpn(8)` (https://jlk.fjfi.cvut.cz/arch/manpages/man/openvpn.8)** man page and the **OpenVPN documentation (http://openvpn.net/index.php/open-source/documentation)**.

The server configuration file

Note: Note that if the server is behind a firewall or a NAT translating router, the OpenVPN port must be forwarded on to the server.

Copy the example server configuration file `/usr/share/openvpn/examples/server.conf` to `/etc/openvpn/server/server.conf`.

Edit the file making a minimum of the following changes:

/etc/openvpn/server/server.conf

```
ca ca.crt
cert servername.crt
key servername.key  # This file should be kept secret
dh dh.pem
.
tls-crypt ta.key # Replaces tls-auth ta.key 0
.
user nobody
group nobody
```

Hardening the server

If security is a priority, additional configuration is recommended including: limiting the server to use a strong cipher/auth method and (optionally) limiting the set of enabled TLS ciphers to the newer ciphers.

Add the following to `/etc/openvpn/server/server.conf`:

/etc/openvpn/server/server.conf

```
.
cipher AES-256-CBC
auth SHA512
tls-version-min 1.2
tls-cipher TLS-DHE-RSA-WITH-AES-256-GCM-SHA384:TLS-DHE-RSA-WITH-AES-128-GCM-SHA256:TLS-DHE-RSA-WITH-AES-256-CBC-SHA:TLS-DHE-RSA-WITH-CAMELLIA-256-CB
```

```
C-SHA:TLS-DHE-RSA-WITH-AES-128-CBC-SHA:TLS-DHE-RSA-WITH-CAMELLIA-128-CBC-SHA
.
```

Note:

- The .ovpn client profile **must** contain a matching cipher and auth line to work properly (at least with the iOS and Android client).
- Using `tls-cipher` incorrectly may cause difficulty with debugging connections and may not be necessary. See **OpenVPN's community wiki (https://community.openvpn.net/openvpn/wiki/Hardening#Useof--tls-cipher)** for more information.

Enabling compression

Since OpenVPN v2.4 it is possible to use LZ4 compression over lzo. LZ4 generally offering the best performance with least CPU usage. For backwards compatibility with OpenVPN versions before v2.4, use lzo `comp-lzo`. Do **not** enable both compression options at the same time.

To do so, configure `/etc/openvpn/server/server.conf` as such:

/etc/openvpn/server/server.conf

```
.
compress lz4-v2
push "compress lz4-v2"
.
```

On the client set `--compress lz4` [2] (https://community.openvpn.net/openvpn/wiki/DeprecatedOptions), although this may be deprecated in the near future.

Deviating from the standard port and/or protocol

Some public/private network admins may not allow OpenVPN connections on its default port and/or protocol. One strategy to circumvent this is to mimic https/SSL traffic which is very likely unobstructed.

To do so, configure `/etc/openvpn/server/server.conf` as such:

```
/etc/openvpn/server/server.conf
```

```
.
port 443
proto tcp
.
```

Note: The .ovpn client profile **must** contain a matching port and proto line to work properly!

TCP vs UDP

There are subtle differences between TCP and UDP.

TCP

- So-called "stateful protocol."
- High reliability due to error correction (i.e. waits for packet acknowledgment).
- Potentially slower than UDP.

UDP

- So-called "stateless protocol."
- Less reliable than TCP as no error correction is in use.
- Potentially faster than TCP.

Note: It is generally a bad idea to use TCP for VPN unless your connection to the server is very stable. High reliability sounds great in theory but any disruption (packet drop, lag spikes, etc...) to the connection will potentially snowball into a **TCP Meltdown (http://sites.inka.de/bigred/devel/tcp-tcp.html)[3] (http://adsabs.harvard.edu/abs/2005SPIE.6011..138H)**.

The client config profile

Copy the example client configuration file `/usr/share/openvpn/examples/client.conf` to `/etc/openvpn/client/`.

Edit the following:

- The `remote` directive to reflect either the server's **Fully Qualified Domain Name**, hostname (as known to the client), or its IP address.
- Uncomment the `user` and `group` directives to drop privileges.
- The `ca`, `cert`, and `key` parameters to reflect the path and names of the keys and certificates.
- Enable the TLS HMAC handshake protection (`--tls-crypt` or `--tls-auth`).

/etc/openvpn/client/client.conf

```
client
remote elmer.acmecorp.org 1194
.
user nobody
group nobody
ca ca.crt
cert client.crt
key client.key
.
tls-crypt ta.key # Replaces tls-auth ta.key 1
```

Run as unprivileged user

Using the options `user nobody` and `group nobody` in the configuration file makes *OpenVPN* drop its `root` privileges after establishing the connection. The downside is that upon VPN disconnect the daemon is unable to delete its set network routes again. If one wants to limit transmitting traffic without the VPN connection, then lingering routes may be considered beneficial. It can also happen, however, that the OpenVPN server pushes updates to routes at runtime of the tunnel. A client with dropped privileges will be unable to perform the update and exit with an error.

As it could seem to require manual action to manage the routes, the options `user nobody` and `group nobody` might seem undesirable. Depending on setup, however, there are different ways to handle these situations:

- For errors of the unit, a simple way is to **edit** it and add a `Restart=on-failure` to the `[Service]` section. Though, this alone will not delete any obsoleted routes, so it may happen that the restarted tunnel is not routed properly.
- The package contains the `/usr/lib/openvpn/plugins/openvpn-plugin-down-root.so`, which can be used to let *openvpn* fork a process with root privileges with the only task to execute a custom script when receiving a down signal from the main process, which is handling the tunnel with dropped privileges (see also its **README (https://community.openvpn.net/openvpn/browser/plugin/down-root/README?rev=d02a86d37bed69ee3fb63d08913623a86c88da15)**).

The OpenVPN HowTo's linked below go further by creating a dedicated non-privileged user/group, instead of the already existing `nobody`. The advantage is that this avoids potential risks when sharing a user among daemons:

- The **OpenVPN HowTo (https://openvpn.net/index.php/open-source/documentation/howto.html#security)** explains another way how to create an unprivileged user mode and wrapper script to have the routes restored automatically.
- It is possible to let OpenVPN start as a non-privileged user in the first place, without ever running as root, see **this OpenVPN wiki (https://community.openvpn.net/openvpn/wik**

i/UnprivilegedUser) (howto). The howto assumes the presence of System V init, rather than Systemd and does not cover the handling of `--up` / `--down` scripts - those should be handled the same way as the *ip* command, with additional attention to access rights.

Note: Due to a bug (https://community.openvpn.net/openvpn/ticket/812) in OpenVPN 2.4.0, the `persist-tun` option mentioned in the howtos should **not** be used, otherwise new routes/IPs pushed on reconnect will be ignored by the client.

Tip: #openvpn-unroot describes a tool to automate above setup.

Converting certificates to encrypted .p12 format

Some software will only read VPN certificates that are stored in a password-encrypted .p12 file. These can be generated with the following command:

```
# openssl pkcs12 -export -inkey keys/bugs.key -in keys/bugs.crt -certfile keys/ca.crt -out keys/bugs.p12
```

Testing the OpenVPN configuration

Run `# openvpn /etc/openvpn/server/server.conf` on the server, and `# openvpn /etc/openvpn/client/client.conf` on the client. Example output should be similar to the following:

```
# openvpn /etc/openvpn/server/server.conf
```

```
Wed Dec 28 14:41:26 2011 OpenVPN 2.2.1 x86_64-unknown-linux-gnu [SSL] [LZO2] [EPOLL] [eurephia] built on Aug 13 2011
Wed Dec 28 14:41:26 2011 NOTE: OpenVPN 2.1 requires '--script-security 2' or higher to call user-defined scripts or executables
Wed Dec 28 14:41:26 2011 Diffie-Hellman initialized with 2048 bit key
.
.
Wed Dec 28 14:41:54 2011 bugs/95.126.136.73:48904 MULTI: primary virtual IP for bugs/95.126.136.73:48904: 10.8.0.6
Wed Dec 28 14:41:57 2011 bugs/95.126.136.73:48904 PUSH: Received control message: 'PUSH_REQUEST'
Wed Dec 28 14:41:57 2011 bugs/95.126.136.73:48904 SENT CONTROL [bugs]: 'PUSH_REPLY,route 10.8.0.1,topology net30,ping 10,ping-restart 120,ifconfig 1
0.8.0.6 10.8.0.5' (status=1)
```

```
# openvpn /etc/openvpn/client/client.conf
```

```
Wed Dec 28 14:41:50 2011 OpenVPN 2.2.1 i686-pc-linux-gnu [SSL] [LZO2] [EPOLL] [eurephia] built on Aug 13 2011
Wed Dec 28 14:41:50 2011 NOTE: OpenVPN 2.1 requires '--script-security 2' or higher to call user-defined scripts or executables
Wed Dec 28 14:41:50 2011 LZO compression initialized
.
.
Wed Dec 28 14:41:57 2011 GID set to nobody
Wed Dec 28 14:41:57 2011 UID set to nobody
Wed Dec 28 14:41:57 2011 Initialization Sequence Completed
```

On the server, find the IP address assigned to the tunX device:

```
# ip addr show
```

```
.
.
40: tun0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN qlen 100
    link/none
    inet 10.8.0.1 peer 10.8.0.2/32 scope global tun0
```

Here we see that the server end of the tunnel has been given the IP address 10.8.0.1.

Do the same on the client:

```
# ip addr show
```

```
.
.
37: tun0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN qlen 100
    link/none
    inet 10.8.0.6 peer 10.8.0.5/32 scope global tun0
```

And the client side has been given the IP address 10.8.0.6.

Now try pinging the interfaces.

On the server:

```
# ping -c3 10.8.0.6
```

```
PING 10.8.0.6 (10.8.0.6) 56(84) bytes of data.
64 bytes from 10.8.0.6: icmp_req=1 ttl=64 time=238 ms
64 bytes from 10.8.0.6: icmp_req=2 ttl=64 time=237 ms
64 bytes from 10.8.0.6: icmp_req=3 ttl=64 time=205 ms

--- 10.8.0.6 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 205.862/227.266/238.788/15.160 ms
```

On the client:

```
# ping -c3 10.8.0.1
```

```
PING 10.8.0.1 (10.8.0.1) 56(84) bytes of data.
64 bytes from 10.8.0.1: icmp_req=1 ttl=64 time=158 ms
64 bytes from 10.8.0.1: icmp_req=2 ttl=64 time=158 ms
64 bytes from 10.8.0.1: icmp_req=3 ttl=64 time=157 ms

--- 10.8.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 157.426/158.278/158.940/0.711 ms
```

Note: If using a firewall, make sure that IP packets on the TUN device are not blocked.

Configure the MTU with Fragment and MSS

Note: If you do not configure MTU, then you will notice that small packets like ping and DNS will work, however web browsing will not work.

Now it is time to configure the maximum segment size (MSS). In order to do this we need to discover what is the smallest MTU along the path between the client and server. In order to do this you can ping the server and disable fragmentation. Then specify the max packet size.

```
# ping -c5 -M do -s 1500 elmer.acmecorp.org
```

```
PING elmer.acmecorp.org (99.88.77.66) 1500(1528) bytes of data.
From 1.2.3.4 (99.88.77.66) icmp_seq=1 Frag needed and DF set (mtu = 576)
From 1.2.3.4 (99.88.77.66) icmp_seq=1 Frag needed and DF set (mtu = 576)
From 1.2.3.4 (99.88.77.66) icmp_seq=1 Frag needed and DF set (mtu = 576)
From 1.2.3.4 (99.88.77.66) icmp_seq=1 Frag needed and DF set (mtu = 576)
From 1.2.3.4 (99.88.77.66) icmp_seq=1 Frag needed and DF set (mtu = 576)

--- core.myrelay.net ping statistics ---
0 packets transmitted, 0 received, +5 errors
```

We received an ICMP message telling us the MTU is 576 bytes. The means we need to fragment the UDP packets smaller then 576 bytes to allow for some UDP overhead.

```
# ping -c5 -M do -s 548 elmer.acmecorp.org
```

```
PING elmer.acmecorp.org (99.88.77.66) 548(576) bytes of data.
556 bytes from 99.88.77.66: icmp_seq=1 ttl=48 time=206 ms
```

```
556 bytes from 99.88.77.66: icmp_seq=2 ttl=48 time=224 ms
556 bytes from 99.88.77.66: icmp_seq=3 ttl=48 time=206 ms
556 bytes from 99.88.77.66: icmp_seq=4 ttl=48 time=207 ms
556 bytes from 99.88.77.66: icmp_seq=5 ttl=48 time=208 ms

--- myrelay.net ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4001ms
rtt min/avg/max/mdev = 206.027/210.603/224.158/6.832 ms
```

After some trial and error..., we discover that we need to fragment packets on 548 bytes. In order to do this we specify this fragment size in the configuration and instruct OpenVPN to fix the Maximum Segment Size (MSS).

/etc/openvpn/client/client.conf

```
remote elmer.acmecorp.org 1194
...
fragment 548
mssfix 548
...
```

We also need to tell the server about the fragmentation. Note that "mssfix" is NOT needed in the server configuration.

Note: Clients that do not support the 'fragment' directive (e.g. OpenELEC, **iOS app (https://forums.openvpn.net/topic13201.html#p31156)**) are not able to connect to a server that uses the 'fragment' directive. To support such clients, skip this section and configure the clients with the 'mtu-test' directive described below.

/etc/openvpn/server/server.conf

```
...
fragment 548
```

Note: The following will add about 3 minutes to OpenVPN start time. It is advisable to configure the fragment size unless your client is a laptop that will be connecting over many different networks and the bottle neck is on the client side.

You can also allow OpenVPN to do this for you by having OpenVPN do the ping testing every time the client connects to the VPN. Be patient, since your client may not inform you about the test being run and the connection may appear as nonfunctional until finished.

/etc/openvpn/client/client.conf

```
remote elmer.acmecorp.org 1194
...
mtu-test
...
```

IPv6

Connect to the server via IPv6

In order to enable Dual Stack for OpenVPN, you have to change `proto udp` to `proto udp6` in both server.conf and client.conf. Afterwards both IPv4 and IPv6 are enabled.

Provide IPv6 inside the tunnel

In order to provide IPv6 inside the tunnel, you need to have a IPv6 prefix routed to your OpenVPN server. Either set up a static route on your gateway (if you have a static block assigned), or use a DHCPv6 client to get a prefix with DHCPv6 Prefix delegation (see **IPv6 Prefix delegation** for details). You can also use a unique local address from the address block fc00::/7. Both methods have advantages and disadvantages:

- Many ISPs only provide dynamically changing IPv6 prefixes. OpenVPN does not support prefix changes, so you need to change your server.conf every time the prefix is changed (Maybe can be automated with a script).
- ULA addresses are not routed to the Internet, and setting up NAT is not as straightforward as with IPv4. So you cannot route the entire traffic over the tunnel. If you only want to connect two sites via IPv6, without the need to connect to the Internet over the tunnel, the ULA addresses may be easier to use.

After you have received a prefix (a /64 is recommended), append the following to the server.conf:

```
server-ipv6 2001:db8:0:123::/64
```

This is the IPv6 equivalent to the default 10.8.0.0/24 network of OpenVPN and needs to be taken from the DHCPv6 client. Or use for example fd00:1234::/64.

If you want to push a route to your home network (192.168.1.0/24 equivalent), also append:

```
push "route-ipv6 2001:db8:0:abc::/64"
```

OpenVPN does not yet include DHCPv6, so there is no method to e.g. push DNS server over IPv6. This needs to be done with IPv4. The **OpenVPN Wiki (https://community.openvpn.net/openvpn/wiki/IPv6)** provides some other configuration options.

Starting OpenVPN

Manual startup

To troubleshoot a VPN connection, start the client's daemon manually with `openvpn /etc/openvpn/client/client.conf` as root. The server can be started the same way using its own configuration file (e.g., `openvpn /etc/openvpn/server/server.conf`).

systemd service configuration

To start the OpenVPN server automatically at system boot, **enable** `openvpn-server@<configuration>.service` on the applicable machine. For a client, **enable** `openvpn-client@<configuration>.service` instead. (Leave `.conf` out of the `<configuration>` string.)

For example, if the client configuration file is `/etc/openvpn/client/client.conf`, the service name is `openvpn-client@client.service`. Or, if the server configuration file is `/etc/openvpn/server/server.conf`, the service name is `openvpn-server@server.service`.

Letting NetworkManager start a connection

On a client you might not always need to run a VPN tunnel and/or only want to establish it for a specific NetworkManager connection. This can be done by adding a script to `/etc/NetworkManager/dispatcher.d/`. In the following example "Provider" is the name of the NetworkManager connection:

/etc/NetworkManager/dispatcher.d/10-openvpn

```
#!/bin/bash

case "$2" in
  up)
    if [ "$CONNECTION_ID" == "Provider" ]; then
      systemctl start openvpn-client@<configuration>
    fi
  ;;
  down)
    systemctl stop openvpn-client@<configuration>
  ;;
esac
```

See **NetworkManager#Network services with NetworkManager dispatcher** for more details.

Gnome configuration

If you would like to connect a client to an OpenVPN server through Gnome's built-in network configuration do the following. First, install **networkmanager-openvpn (https://www.archlinux.org/packages/?name=networkmanager-openvpn)**. Then go to the Settings menu and choose Network. Click the plus sign to add a new connection and choose VPN. From there you can choose OpenVPN and manually enter the settings. You can also choose to import **#The client config profile**, if you have already created one. Yet, be aware NetworkManager does not show error messages for options it does not import. To connect to the VPN simply turn the connection on and check the options are applied as you configured (e.g. via `journalctl -b -u NetworkManager`).

Routing all client traffic through the server

Note: There are potential pitfalls when routing all traffic through a VPN server. Refer to **the OpenVPN documentation on this topic (http://openvpn.net/index.php/open-source/documentation/howto.html#redirect)** for more information.

By default only traffic directly to and from an OpenVPN server passes through the VPN. To have all traffic, including web traffic, pass through the VPN do the following. First add the following to your server's configuration file (i.e., `/etc/openvpn/server/server.conf`) **[4] (http://openvpn.net/index.php/open-source/documentation/howto.html#redirect)**:

```
push "redirect-gateway def1 bypass-dhcp"
push "dhcp-option DNS 8.8.8.8"
```

Change `8.8.8.8` to your preferred DNS IP address if configured to run on the same box as the server or else leave it at 8.8.8.8 to use google's DNS.

If you have problems with non responsive DNS after connecting to server, install **BIND** as simple DNS forwarder and push the IP address of the OpenVPN server as DNS to clients.

After setting up the configuration file, one must **enable packet forwarding** on the server. Additionally, the server's firewall will need to be set up to allow VPN traffic through it, which is described below for both **ufw** and **iptables**.

To allow clients to be able to reach other (private) subnets behind the server, you may want to use the `push "route <address pool> <subnet>"` option:

```
push "route 172.10.142.0 255.255.255.0"
push "route 172.20.142.0 255.255.255.0"
```

Firewall configuration

ufw

In order to configure your ufw settings for VPN traffic first add the following to `/etc/default/ufw`:

```
/etc/default/ufw
```

```
DEFAULT_FORWARD_POLICY="ACCEPT"
```

Now change `/etc/ufw/before.rules`, and add the following code after the header and before the "*filter" line. Do not forget to change the IP/subnet mask to match the one in `/etc/openvpn/server/server.conf`. The adapter ID in the example is generically called `eth0` so edit it for your system accordingly.

```
/etc/ufw/before.rules
```

```
# NAT (Network Address Translation) table rules
*nat
:POSTROUTING ACCEPT [0:0]

# Allow traffic from clients to eth0
-A POSTROUTING -s 10.8.0.0/24 -o eth0 -j MASQUERADE

# do not delete the "COMMIT" line or the NAT table rules above will not be processed
COMMIT
```

Open OpenVPN port 1194:

```
# ufw allow 1194
```

Lastly, reload UFW:

```
# ufw reload
```

iptables

In order to allow VPN traffic through your iptables firewall of your server, first create an iptables rule for NAT forwarding **[5] (http://openvpn.net/index.php/open-source/documentation/howto.html#redirect)** on the server, assuming the interface you want to forward to is named `eth0`:

```
iptables -t nat -A POSTROUTING -s 10.8.0.0/24 -o eth0 -j MASQUERADE
```

If you have difficulty pinging the server through the VPN, you may need to add explicit rules to open up TUN/TAP interfaces to all traffic. If that is the case, do the following **[6] (https://community.openvpn.net/openvpn/wiki/255-qconnection-initiated-with-xxxxq-but-i-cannot-ping-the-server-through-the-vpn)**:

Warning: There are security implications for the following rules if you do not trust all clients which connect to the server. Refer to the **OpenVPN documentation on this topic (https://community.openvpn.net/openvpn/wiki/255-qconnection-initiated-with-xxxxq-but-i-cannot-ping-the-server-through-the-vpn)** for more details.

```
iptables -A INPUT -i tun+ -j ACCEPT
iptables -A FORWARD -i tun+ -j ACCEPT
iptables -A INPUT -i tap+ -j ACCEPT
iptables -A FORWARD -i tap+ -j ACCEPT
```

Additionally be sure to accept connections from the OpenVPN port (default 1194) and through the physical interface.

When you are satisfied make the changes permanent as shown in **iptables#Configuration and usage**.

If you have multiple `tun` or `tap` interfaces, or more than one VPN configuration, you can "pin" the name of your interface by specifying it in the OpenVPN config file, e.g. `tun22` instead of `tun`. This is advantageous if you have different firewall rules for different interfaces or OpenVPN configurations.

Prevent leaks if VPN goes down

The idea is simple: prevent all traffic through our default interface (enp3s0 for example) and only allow tun0. If the OpenVPN connection drops, your computer will lose its internet access and therefore, avoid your programs to continue connecting through an insecure network adapter.

Be sure to set up a script to restart OpenVPN if it goes down if you do not want to manually restart it.

ufw

```
# Default policies
ufw default deny incoming
ufw default deny outgoing

# Openvpn interface (adjust interface accordingly to your configuration)
ufw allow in on tun0
ufw allow out on tun0

# Local Network (adjust ip accordingly to your configuration)
ufw allow in on enp3s0 from 192.168.1.0/24
ufw allow out on enp3s0 to 192.168.1.0/24

# Openvpn (adjust port accordingly to your configuration)
ufw allow in on enp3s0 from any port 1194
ufw allow out on enp3s0 to any port 1194
```

Warning: DNS **will not** work **unless** you run your own DNS server like **BIND**

Otherwise, you will need to allow dns leak. **Be sure to trust your DNS server!**

```
# DNS
ufw allow in from any to any port 53
ufw allow out from any to any port 53
```

vpnfailsafe

Alternatively, the **vpnfailsafe (https://github.com/wknapik/vpnfailsafe)** (**`vpnfailsafe-git (https://aur.archlinux.org/packages/vpnfailsafe-git/)`**[AUR]) script can be used by the client to prevent DNS leaks and ensure that all traffic to the internet goes over the VPN. If the

VPN tunnel goes down, internet access will be cut off, except for connections to the VPN server(s). The script contains the functionality of **update-resolv-conf**, so the two do not need to be combined.

L3 IPv4 routing

This section describes how to connect client/server LANs to each other using L3 IPv4 routing.

Prerequisites for routing a LAN

For a host to be able to forward IPv4 packets between the LAN and VPN, it must be able to forward the packets between its NIC and its tun/tap device. See **Internet sharing#Enable packet forwarding** for configuration details.

Routing tables

By default, all IP packets on a LAN addressed to a different subnet get sent to the default gateway. If the LAN/VPN gateway is also the default gateway, there is no problem and the packets get properly forwarded. If not, the gateway has no way of knowing where to send the packets. There are a couple of solutions to this problem.

- Add a static route to the default gateway routing the VPN subnet to the LAN/VPN gateway's IP address.
- Add a static route on each host on the LAN that needs to send IP packets back to the VPN.
- Use **iptables**' NAT feature on the LAN/VPN gateway to masquerade the incoming VPN IP packets.

Connect the server LAN to a client

The server is on a LAN using the 10.66.0.0/24 subnet. To inform the client about the available subnet, add a push directive to the server configuration file:

```
/etc/openvpn/server/server.conf
```

```
push "route 10.66.0.0 255.255.255.0"
```

Note: To route more LANs from the server to the client, add more push directives to the server configuration file, but keep in mind that the server side LANs will need to know how to route to the client.

Connect the client LAN to a server

Prerequisites:

- Any subnets used on the client side, must be unique and not in use on the server or by any other client. In this example we will use 192.168.4.0/24 for the clients LAN.
- Each client's certificate has a unique Common Name, in this case bugs.
- The server may not use the duplicate-cn directive in its config file.

Create a client configuration directory on the server. It will be searched for a file named the same as the client's common name, and the directives will be applied to the client when it connects.

```
# mkdir -p /etc/openvpn/ccd
```

Create a file in the client configuration directory called bugs, containing the `iroute 192.168.4.0 255.255.255.0` directive. It tells the server what subnet should be routed to the client:

```
/etc/openvpn/ccd/bugs
```

```
iroute 192.168.4.0 255.255.255.0
```

Add the client-config-dir and the `route 192.168.4.0 255.255.255.0` directive to the server configuration file. It tells the server what subnet should be routed from the tun device to the server LAN:

```
/etc/openvpn/server/server.conf
```

```
client-config-dir ccd
route 192.168.4.0 255.255.255.0
```

Note: To route more LANs from the client to the server, add more iroute and route directives to the appropriate configuration files, but keep in mind that the client side LANs will need to know how to route to the server.

Connect both the client and server LANs

Combine the two previous sections:

/etc/openvpn/server/server.conf

```
push "route 10.66.0.0 255.255.255.0"
.
.
client-config-dir ccd
route 192.168.4.0 255.255.255.0
```

/etc/openvpn/ccd/bugs

```
iroute 192.168.4.0 255.255.255.0
```

Note: Remember to make sure that all the LANs or the needed hosts can route to all the destinations.

Connect clients and client LANs

By default clients will not see each other. To allow IP packets to flow between clients and/or client LANs, add a client-to-client directive to the server configuration file:

/etc/openvpn/server/server.conf

```
client-to-client
```

In order for another client or client LAN to see a specific client LAN, you will need to add a push directive for each client subnet to the server configuration file (this will make the server announce the available subnet(s) to other clients):

/etc/openvpn/server/server.conf

```
client-to-client
push "route 192.168.4.0 255.255.255.0"
push "route 192.168.5.0 255.255.255.0"
.
.
```

Note: As always, make sure that the routing is properly configured.

DNS

The DNS servers used by the system are defined in `/etc/resolv.conf`. Traditionally, this file is the responsibility of whichever program deals with connecting the system to the network (e.g. Wicd, NetworkManager, etc.). However, OpenVPN will need to modify this file if you want to be able to resolve names on the remote side. To achieve this in a sensible way,

install **openresolv (https://www.archlinux.org/packages/?name=openresolv)**, which makes it possible for more than one program to modify `resolv.conf` without stepping on each-other's toes.

Before continuing, test openresolv by restarting your network connection and ensuring that `resolv.conf` states that it was generated by *resolvconf*, and that your DNS resolution still works as before. You should not need to configure openresolv; it should be automatically detected and used by your network system.

For Linux, OpenVPN can send DNS host information, but expects an external process to act on it. This can be done with the `client.up` and `client.down` scripts packaged in `/usr/share/openvpn/contrib/pull-resolv-conf/`. See their comments on how to install them to `/etc/openvpn/client/`. The following is an excerpt of a resulting client configuration using the scripts in conjunction with *resolvconf* and options to **#Run as unprivileged user**:

/etc/openvpn/client/*clienttunnel*.conf

```
user nobody
group nobody
# Optional, choose a suitable path to chroot into for your system
chroot /srv
script-security 2
up /etc/openvpn/client/client.up
plugin /usr/lib/openvpn/plugins/openvpn-plugin-down-root.so "/etc/openvpn/client/client.down tun0"
```

Update resolv-conf script

The **openvpn-update-resolv-conf (https://github.com/masterkorp/openvpn-update-resolv-conf)** script is available as an alternative to packaged scripts. It needs to be saved for example at `/etc/openvpn/update-resolv-conf` and made executable with **chmod**.

Once the script is installed add lines like the following into your OpenVPN client configuration file:

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

Note: If manually placing the script on the filesystem, be sure to have **openresolv (https://www.archlinux.org/packages/?name=openresolv)** installed.

Now, when your launch your OpenVPN connection, you should find that your resolv.conf file is updated accordingly, and also returns to normal when your close the connection.

Note: When using `openresolv` with the *-p* or *-x* options in a script (as both the included `client.up` and `update-resolv-conf` scripts currently do), a DNS resolver like **dnsmasq (https://www.archlinux.org/packages/?name=dnsmasq)** or **unbound (https://www.archlinux.org/packages/?name=unbound)** is required for `openresolv` to correctly update `/etc/resolv.conf`. In contrast, when using the default DNS resolution from `libc` the *-p* and *-x* options must be removed in order for `/etc/resolv.conf` to be correctly updated by `openresolv`. For example, if the script contains a command like `resolvconf -p -a` and

the default DNS resolver from `libc` is being used, change the command in the script to be `resolvconf -a`.

Update systemd-resolved script

Since systemd-229, **systemd-networkd**'s `systemd-resolved.service` has exposed an API through DBus allowing management of DNS configuration on a per-link basis. Tools such as **openresolv (https://www.archlinux.org/packages/?name=openresolv)** may not work reliably when `/etc/resolv.conf` is managed by `systemd-resolved`, and will not work at all if you are using `resolve` instead of `dns` in your `/etc/nsswitch.conf` file. The **update-systemd-resolved (https://github.com/jonathanio/update-systemd-resolved)** script is another alternative and links OpenVPN with `systemd-resolved` via DBus to update the DNS records.

If you copy the script into `/etc/openvpn` and mark as executable with **chmod**, or install it via the AUR package (**openvpn-update-systemd-resolved (https://aur.archlinux.org/packages/openvpn-update-systemd-resolved/)**[AUR]), you can add lines like the following into your OpenVPN client configuration file:

```
script-security 2
setenv PATH /usr/bin
up /etc/openvpn/scripts/update-systemd-resolved
down /etc/openvpn/scripts/update-systemd-resolved
down-pre
```

L2 Ethernet bridging

For now see: **OpenVPN Bridge**

Config generators

Warning: Users are highly recommended to pass through the manual configuration described above to gain knowledge about options and usage before using any additional automation scripts.

ovpngen

The `ovpngen (https://aur.archlinux.org/packages/ovpngen/)`[AUR] package provides a simple shell script that creates OpenVPN compatible tunnel profiles in the unified file format suitable for the iOS version of OpenVPN Connect as well as for the Android app.

Simply invoke the script with 5 tokens:

1. Server Fully Qualified Domain Name of the OpenVPN server (or IP address).
2. Full path to the CA cert.
3. Full path to the client cert.
4. Full path to the client private key.

5. Full path to the server TLS shared secret key.
6. Optionally a port number.
7. Optionally a protocol (udp or tcp).

Example:

```
# ovpngen example.org /etc/openvpn/server/ca.crt /etc/easy-rsa/pki/signed/client1.crt /etc/easy-rsa/pki/private/client1.key /etc/openvpn/server/ta.k
ey > iphone.ovpn
```

The resulting `iphone.ovpn` can be edited if desired as the script does insert some commented lines.

The client expects this file to be located in `/etc/openvpn/client/iphone.conf`. Note the change in file extension from 'ovpn' to 'conf' in this case.

Tip: If the server.conf contains a specified cipher and/or auth line, it is highly recommended that users manually edit the generated .ovpn file adding matching lines for cipher and auth. Failure to do so may results in connection errors!

openvpn-unroot

The steps necessary for OpenVPN to **#Run as unprivileged user**, can be performed automatically using **openvpn-unroot (https://github.com/wknapik/openvpn-unroot)** (**openvpn-unroot-git (https://aur.archlinux.org/packages/openvpn-unroot-**

git/)[AUR]).

It automates the actions required for the **OpenVPN howto (https://community.openvpn.net/openvpn/wiki/UnprivilegedUser)** by adapting it to systemd, and also working around the bug for persistent tun devices mentioned in the note.

Troubleshooting

Client daemon not reconnecting after suspend

openvpn-reconnect (https://aur.archlinux.org/packages/openvpn-reconnect/)[AUR], available on the AUR, solves this problem by sending a SIGHUP to openvpn after waking up from suspend.

Alternatively, you can kill and restart OpenVPN after suspend by creating the folowing systemd service:

/etc/systemd/system/openvpn-reconnect.service

```
[Unit]
Description=Restart OpenVPN after suspend

[Service]
ExecStart=/usr/bin/pkill --signal SIGHUP --exact openvpn

[Install]
WantedBy=sleep.target
```

Enable this service for it to take effect.

Connection drops out after some time of inactivity

If the VPN-Connection drops some seconds after it stopped transmitting data and, even though it states it is connected, no data can be transmitted through the tunnel, try adding a `keepalive` directive to the server's configuration:

/etc/openvpn/server/server.conf

```
.
.
keepalive 10 120
.
.
```

In this case the server will send ping-like messages to all of its clients every 10 seconds, thus keeping the tunnel up. If the server does not receive a response within 120 seconds from a specific client, it will assume this client is down.

A small ping-interval can increase the stability of the tunnel, but will also cause slightly higher traffic. Depending on your connection, also try lower intervals than 10 seconds.

PID files not present

The default systemd service file for openvpn-client does not have the --writepid flag enabled, despite creating /var/run/openvpn-client. If this breaks a config (such as an i3bar VPN indicator), simply change `openvpn-client@.service` using a **drop-in snippet**:

```
[Service]
ExecStart=
ExecStart=/usr/sbin/openvpn --suppress-timestamps --nobind --config %i.conf --writepid /var/run/openvpn-client/%i.pid
```

Route configuration fails with systemd-networkd

When using **systemd-networkd** to manage network connections and attempting to tunnel all outgoing traffic through the VPN, OpenVPN may fail to add routes. This is a result of systemd-networkd attempting to manage the tun interface before OpenVPN finishes configuring the routes. When this happens, the following message will appear in the OpenVPN log.

```
openvpn[458]: RTNETLINK answers: Network is unreachable
openvpn[458]: ERROR: Linux route add command failed: external program exited with error status: 2
```

With systemd-233 (currently in **testing**), systemd-networkd can be configured to ignore the tun connections and allow OpenVPN to manage them. To do this, create the following file:

```
/etc/systemd/network/90-tun-ignore.network
```

```
[Match]
Name=tun*
```

```
[Link]
Unmanaged=true
```

Restart `systemd-networkd.service` to apply the changes. To verify that the changes took effect, start the previously problematic OpenVPN connection and run `networkctl`. The output should have a line similar to the following:

```
7 tun0             none              routable    unmanaged
```

See also

- **Wikipedia:OpenVPN**

Retrieved from "https://wiki.archlinux.org/index.php?title=OpenVPN&oldid=510533"
