Miscellaneous

Introduction

This HOWTO is mainly relevant for setting up single-client or static site-to-site VPNs and is oriented more towards OpenVPN 1.x than 2.0. To take advantage of the OpenVPN 2.0 client/server capability, see the OpenVPN 2.0 HOWTO.

This document describes setting up OpenVPN in a typical Home to Office telecommuting configuration. While this HOWTO presents in-depth configuration examples, simpler examples are shown in the examples section of the man page.

Additional Articles and Documentation

Many excellent articles and HOWTOs exist for configuring OpenVPN in different environments.

Basic Tunnel Types

There are two basic types of tunnels that one can create with OpenVPN:

- **Routed IP tunnels** -- best used to route point-to-point IP traffic without broadcasts. Slightly more efficient than bridged ethernet tunnels and easier to configure. This HOWTO (below) covers routed IP tunnels.
- **Bridged Ethernet Tunnels** -- can be used to tunnel both IP and non-IP protocols. This type of tunnel is appropriate for applications which communicate via broadcasts, such as Windows file and print sharing (without a WINS server) and LAN games. Slightly more complex to configure. A Mini-HOWTO for bridged ethernet tunnels.

Routed IP tunnel HOWTO

Given the interrelated issues involved in configuring firewalls, VPNs, and NAT, we will attempt to describe a complete system configuration rather than isolating the VPN component of the setup.

In our example, both Home and Office private networks are linked to the internet via two gateway machines which each have a public IP address. Each gateway machine contains two NICs, one connected to the private network, the other connected to the internet. The gateway machines provide NAT, firewall, and VPN services for the machines on the private networks. The Home and Office sides of the configuration are roughly symmetrical except the Office gateway machine has a fixed IP address while the Home machine has a DHCP dynamic address.

In the following examples, all configuration files shown are also available in the OpenVPN distribution.

Home and Office IP Networking Parameters

	Home	Office
Local Ethernet Subnet (Private Address)	10.0.1.0/24	10.0.0.0/24
Tunnel Endpoint (Private Address)	10.1.0.2	10.1.0.1
OpenVPN Gateway (Public Address)	DHCP client, need not be explicitly specified	1.2.3.4

Installing OpenVPN

If your system doesn't have the OpenSSL Library, you should download and install it.

If you want to take advantage of compression on the VPN link, or you want to install OpenVPN as an

RPM package, install the LZO Library.

If you are using Linux 2.2 or earlier, download the TUN/TAP driver. Users of Linux 2.4.7 or greater should find the TUN/TAP driver already bundled with their kernel. Users of Linux 2.4.0 -> 2.4.6 should note the caveat at the end of the INSTALL file.

Now download the latest release of OpenVPN.

Install from tarball

Unzip the distribution:

```
gzip -dc openvpn-1.6.0.tar.gz | tar xvf -
```

Build OpenVPN:

```
cd openvpn-1.6.0
./configure
make
make install
```

If you didn't download the LZO Library, add **--disable-lzo** to the **configure** command. Other options can be enabled such as **pthread** support (**./configure --enable-pthread**) to improve latency during SSL/TLS dynamic key exchanges. The command

```
./configure --help
```

will show all configuration options.

Install from RPM

First build the RPM file. This will require that the OpenSSL, pthread, and LZO libraries are present. Normally only the LZO library requires an explicit download and install; the other libraries are present by default on most Linux distributions.

```
rpmbuild -tb openvpn-1.6.0.tar.gz
```

The RPM build process will generate a lot of output. If the build succeeds, there should be a note near the end of the output stating the name of the binary RPM file which was written. Install the binary RPM with the command:

```
rpm -Uvh binary-RPM-file
```

Configuring the TUN/TAP driver

One-time Configuration Steps

If you are using Linux 2.4.7 or higher, chances are good that the TUN/TAP driver is already bundled with your kernel. You can confirm this with the command

```
locate if_tun.h
```

which should show a file such as **/usr/include/linux/if_tun.h**.

For Linux 2.4.7 or higher, if you installed from the tarball, enter the following command to configure the TUN/TAP device node (you can omit this step if you installed from RPM, as the RPM install will do it automatically for you):

```
mknod /dev/net/tun c 10 200
```

If you are using Linux 2.2, you should obtain Version 1.1 of the TUN/TAP kernel module and follow the installation instructions.

Once-per-reboot Configuration Steps

On Linux, prior to using OpenVPN or any other program which uses TUN/TAP devices, you should load the TUN/TAP kernel module:

```
modprobe tun
```

and enable IP forwarding:

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

Configure Firewall and NAT

This section assumes you are using Linux 2.4 with an **iptables** firewall. Here is a sample firewall configuration that provides NAT for machines on a private network to access the internet, stateful outgoing connection tracking, and OpenVPN support:

sample-config-files/firewall.sh

```
#!/bin/bash

# A Sample OpenVPN-aware firewall.

# eth0 is connected to the internet.
# eth1 is connected to a private subnet.

# Change this subnet to correspond to your private
# ethernet subnet.  Home will use 10.0.1.0/24 and
# Office will use 10.0.0.0/24.
PRIVATE=10.0.0.0/24

# Loopback address
LOOP=127.0.0.1

# Delete old iptables rules
# and temporarily block all traffic.
iptables -P OUTPUT DROP
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -F

# Set default policies
iptables -P OUTPUT ACCEPT
iptables -P INPUT DROP
iptables -P FORWARD DROP

# Prevent external packets from using loopback addr
iptables -A INPUT -i eth0 -s $LOOP -j DROP
iptables -A FORWARD -i eth0 -s $LOOP -j DROP
iptables -A INPUT -i eth0 -d $LOOP -j DROP
iptables -A FORWARD -i eth0 -d $LOOP -j DROP

# Anything coming from the Internet should have a real Internet address
iptables -A FORWARD -i eth0 -s 192.168.0.0/16 -j DROP
iptables -A FORWARD -i eth0 -s 172.16.0.0/12 -j DROP
iptables -A FORWARD -i eth0 -s 10.0.0.0/8 -j DROP
iptables -A INPUT -i eth0 -s 192.168.0.0/16 -j DROP
iptables -A INPUT -i eth0 -s 172.16.0.0/12 -j DROP
iptables -A INPUT -i eth0 -s 10.0.0.0/8 -j DROP

# Block outgoing NetBios (if you have windows machines running
# on the private subnet).  This will not affect any NetBios
# traffic that flows over the VPN tunnel, but it will stop
# local windows machines from broadcasting themselves to
# the internet.
iptables -A FORWARD -p tcp --sport 137:139 -o eth0 -j DROP
iptables -A FORWARD -p udp --sport 137:139 -o eth0 -j DROP
iptables -A OUTPUT -p tcp --sport 137:139 -o eth0 -j DROP
iptables -A OUTPUT -p udp --sport 137:139 -o eth0 -j DROP

# Check source address validity on packets going out to internet
iptables -A FORWARD -s ! $PRIVATE -i eth1 -j DROP

# Allow local loopback
iptables -A INPUT -s $LOOP -j ACCEPT
iptables -A INPUT -d $LOOP -j ACCEPT

# Allow incoming pings (can be disabled)
iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT

# Allow services such as www and ssh (can be disabled)
iptables -A INPUT -p tcp --dport http -j ACCEPT
iptables -A INPUT -p tcp --dport ssh -j ACCEPT

# Allow incoming OpenVPN packets
# Duplicate the line below for each
# OpenVPN tunnel, changing --dport n
# to match the OpenVPN UDP port.
#
# In OpenVPN, the port number is
# controlled by the --port n option.
# If you put this option in the config
# file, you can remove the leading '--'
#
# If you taking the stateful firewall
```

```
# approach (see the OpenVPN HOWTO),
# then comment out the line below.

iptables -A INPUT -p udp --dport 1194 -j ACCEPT

# Allow packets from TUN/TAP devices.
# When OpenVPN is run in a secure mode,
# it will authenticate packets prior
# to their arriving on a tun or tap
# interface.  Therefore, it is not
# necessary to add any filters here,
# unless you want to restrict the
# type of packets which can flow over
# the tunnel.

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

# Allow packets from private subnets
iptables -A INPUT -i eth1 -j ACCEPT
iptables -A FORWARD -i eth1 -j ACCEPT

# Keep state of connections from local machine and private subnets
iptables -A OUTPUT -m state --state NEW -o eth0 -j ACCEPT
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -m state --state NEW -o eth0 -j ACCEPT
iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT

# Masquerade local subnet
iptables -t nat -A POSTROUTING -s $PRIVATE -o eth0 -j MASQUERADE
```

OpenVPN offers a few additional options on firewall setup:

- If both OpenVPN peers reference the other with an explicit **--remote** option, and stateful firewalls that provide UDP connection tracking (such as **iptables**) exist between the peers, it is possible to run OpenVPN without any explicit firewall rules, if both peers originate regular pings to each other to keep the connection alive. To do this, simply run OpenVPN with the **--remote *peer*** option, and specify **--ping 15** to ensure that packets flow over the tunnel at least once every 15 seconds.
- The above option is less convenient if one of the peers changes its IP address frequently such as a DHCP or a dial-in peer. For these cases, the sample firewall configuration above will allow incoming packets on UDP port 1194 (OpenVPN's default UDP port) from any IP address. This should be considered safe in any of OpenVPN's secure modes, since all incoming tunnel packets must pass an authentication test or they will be dropped.
- If you choose to fully open OpenVPN's incoming UDP port as in the sample firewall configuration above, you might want to take advantage of the **--tls-auth** option to do double authentication on the TLS control channel, using both the RSA key and a pre-shared secret passphrase as a second line of defense against DoS or active attacks. For more information on **--tls-auth**, see the openvpn man page.

Build RSA Certificates and Keys

OpenVPN has two secure modes, one based on SSL/TLS security using RSA certificates and keys, the other using a pre-shared static key. While SSL/TLS + RSA keys is arguably the most secure option, static keys have the benefit of simplicity. If you want to use RSA keys, read on. For static keys, jump forward to the Build Pre-Shared Static Key section.

We will build RSA certificates and keys using the **openssl** command, included in the OpenSSL library distribution.

RSA certificates are public keys that also have other secure fields embedded in them such as the **Common Name** or **email address** of the certificate holder. OpenVPN provides the ability to write scripts to test these fields prior to authentication. For more information, see the **--tls-verify** option in the openvpn man page.

In our example we will follow the **apache** convention of using the **.crt** file extension to denote certificate files and the **.key** file extension to denote private key files. Private key files must always be kept secure. Certificate files can be freely published or shared.

Select one machine such as Office to be the key management machine.

First edit the **/usr/share/ssl/openssl.cnf** file (this file may exist in a different place, so use **locate openssl.cnf** to find it).

You may want to make some changes to this file:

- Make a directory to serve as your key working area and change **dir** to point to it.
- Consider increasing **default_days** so your VPN doesn't mysteriously stop working after exactly one year.
- Set **certificate** and **private_key** to point to your master certificate authority certificate and private

key files which we will presently generate. In the examples below, we will assume that your certificate authority certificate is named **my-ca.crt** and your certificate authority private key is named **my-ca.key**.
- Note the files **index.txt** and **serial**. Initialize **index.txt** to be empty and **serial** to contain an initial serial number such as **01**.
- If you are paranoid about key sizes, increase **default_bits** to 2048. OpenVPN will have no problem handling a 2048 bit RSA key if you have built OpenVPN with **pthread** support, to enable background processing of RSA keys. You can still use large keys even without **pthread** support, but you will see some latency degradation on the tunnel during SSL/TLS key negotiations. For a good article on choosing an RSA key size, see the April 2002 issue of Bruce Schneier's Crypto-Gram Newsletter.

After **openssl.cnf** has been edited, create your master certificate authority certificate/private-key pair:

```
openssl req -nodes -new -x509 -keyout my-ca.key -out my-ca.crt -days 3650
```

This will create a master certificate authority certificate/private-key pair valid for 10 years.

Now create certificate/private-key pairs for both Home and Office. When prompted for the common name, make sure to use a different name for Home and Office.

```
openssl req -nodes -new -keyout office.key -out office.csr
openssl ca -out office.crt -in office.csr
openssl req -nodes -new -keyout home.key -out home.csr
openssl ca -out home.crt -in home.csr
```

Now copy **home.crt**, **home.key**, and **my-ca.crt** to Home over a secure channel, though actually only **.key** files should be considered non-public.

Now create Diffie Hellman parameters on Office with the following command:

```
openssl dhparam -out dh1024.pem 1024
```

Increase the bit size from 1024 to 2048 if you also increased it in **openssl.cnf**.

For the paranoid, consider omitting the **-nodes** option on the **openssl** commands above. This will cause each private key to be encrypted with a password, making the keys secure even if someone broke onto your server and stole your private key files. The downside of this approach is that every time you run OpenVPN, you will need to type in the password. For more information see the **--askpass** option in the openvpn man page.

If you find manual RSA key management confusing, note that OpenVPN will interoperate with any X509 certificate management tool or service including the commercial CAs such as Thawte or Verisign. Check out the OpenCA project for an example of what's being done with certificate/key management in the Open Source realm.

In addition, the OpenVPN distribution contains a small set of scripts which can be used to simplify RSA certificate and key management.

Important Note on the use of commercial certificate authorities (CAs) with OpenVPN

It should be noted that OpenVPN's security model in SSL/TLS mode is oriented toward users who will generate their own root certificate, and hence be their own CA. In SSL/TLS mode, OpenVPN authenticates its peer by checking that the peer-supplied certificate was signed by the CA certificate specified in the **--ca** option. Like the SSL-based secure web, the security of OpenVPN's SSL/TLS mode rests on the infeasibility of forging a root certificate signature.

This authentication procedure works perfectly well if you have generated your own root certificate, but presents a problem if you wish to use the root certificate of a commercial CA such as Thawte. If, for example, you specified Thawte's root certificate in the **--ca** option, any certificate signed by Thawte would now be able to authenticate with your OpenVPN peer -- certainly not what you would want.

Luckily there is a solution to this problem in the **--tls-verify** option. This option will allow you to execute a command to check the contents of a certificate, to fine-tune the selection of which certificate is allowed, and which is not. See the script **verify-cn** in the sample-scripts subdirectory for an example of how to do this, and also see the man page for the **--tls-verify** option.

Important Note on possible "Man-in-the-Middle" attack if clients do not verify the certificate of the server they are connecting to.

See discussion here.

Configuration file using SSL/TLS mode and RSA certificates/keys

In our example, we will use OpenVPN configuration files. OpenVPN allows options to be passed on either the command line or in one or more configuration files. Options in configuration files can omit the leading "--" that is required for command line options.

Set up the following configuration files:

sample-config-files/tls-office.conf

```
#
# Sample OpenVPN configuration file for
# office using SSL/TLS mode and RSA certificates/keys.
#
# '#' or ';' may be used to delimit comments.

# Use a dynamic tun device.
# For Linux 2.2 or non-Linux OSes,
# you may want to use an explicit
# unit number such as "tun1".
# OpenVPN also supports virtual
# ethernet "tap" devices.
dev tun

# 10.1.0.1 is our local VPN endpoint (office).
# 10.1.0.2 is our remote VPN endpoint (home).
ifconfig 10.1.0.1 10.1.0.2

# Our up script will establish routes
# once the VPN is alive.
up ./office.up

# In SSL/TLS key exchange, Office will
# assume server role and Home
# will assume client role.
tls-server

# Diffie-Hellman Parameters (tls-server only)
dh dh1024.pem

# Certificate Authority file
ca my-ca.crt

# Our certificate/public key
cert office.crt

# Our private key
key office.key

# OpenVPN 2.0 uses UDP port 1194 by default
# (official port assignment by iana.org 11/04).
# OpenVPN 1.x uses UDP port 5000 by default.
# Each OpenVPN tunnel must use
# a different port number.
# lport or rport can be used
# to denote different ports
# for local and remote.
; port 1194

# Downgrade UID and GID to
# "nobody" after initialization
# for extra security.
; user nobody
; group nobody

# If you built OpenVPN with
# LZO compression, uncomment
# out the following line.
; comp-lzo

# Send a UDP ping to remote once
# every 15 seconds to keep
# stateful firewall connection
# alive.  Uncomment this
# out if you are using a stateful
# firewall.
; ping 15
```

```
# Uncomment this section for a more reliable detection when a system
# loses its connection.  For example, dial-ups or laptops that
# travel to other locations.
; ping 15
; ping-restart 45
; ping-timer-rem
; persist-tun
; persist-key

# Verbosity level.
# 0 -- quiet except for fatal errors.
# 1 -- mostly quiet, but display non-fatal network errors.
# 3 -- medium output, good for normal operation.
# 9 -- verbose, good for troubleshooting
verb 3
```

sample-config-files/office.up

```
#!/bin/sh
route add -net 10.0.1.0 netmask 255.255.255.0 gw $5
```

sample-config-files/tls-home.conf

```
#
# Sample OpenVPN configuration file for
# home using SSL/TLS mode and RSA certificates/keys.
#
# '#' or ';' may be used to delimit comments.

# Use a dynamic tun device.
# For Linux 2.2 or non-Linux OSes,
# you may want to use an explicit
# unit number such as "tun1".
# OpenVPN also supports virtual
# ethernet "tap" devices.
dev tun

# Our OpenVPN peer is the office gateway.
remote 1.2.3.4

# 10.1.0.2 is our local VPN endpoint (home).
# 10.1.0.1 is our remote VPN endpoint (office).
ifconfig 10.1.0.2 10.1.0.1

# Our up script will establish routes
# once the VPN is alive.
up ./home.up

# In SSL/TLS key exchange, Office will
# assume server role and Home
# will assume client role.
tls-client

# Certificate Authority file
ca my-ca.crt

# Our certificate/public key
cert home.crt

# Our private key
key home.key

# OpenVPN 2.0 uses UDP port 1194 by default
# (official port assignment by iana.org 11/04).
# OpenVPN 1.x uses UDP port 5000 by default.
# Each OpenVPN tunnel must use
# a different port number.
# lport or rport can be used
# to denote different ports
# for local and remote.
; port 1194

# Downgrade UID and GID to
```

```
# "nobody" after initialization
# for extra security.
; user nobody
; group nobody

# If you built OpenVPN with
# LZO compression, uncomment
# out the following line.
; comp-lzo

# Send a UDP ping to remote once
# every 15 seconds to keep
# stateful firewall connection
# alive.  Uncomment this
# out if you are using a stateful
# firewall.
; ping 15

# Uncomment this section for a more reliable detection when a system
# loses its connection.  For example, dial-ups or laptops that
# travel to other locations.
; ping 15
; ping-restart 45
; ping-timer-rem
; persist-tun
; persist-key

# Verbosity level.
# 0 -- quiet except for fatal errors.
# 1 -- mostly quiet, but display non-fatal network errors.
# 3 -- medium output, good for normal operation.
# 9 -- verbose, good for troubleshooting
verb 3
```

sample-config-files/home.up

```
#!/bin/sh
route add -net 10.0.0.0 netmask 255.255.255.0 gw $5
```

Build A Pre-Shared Static Key

In contrast with RSA key management, using a pre-shared static key has the benefit of simplicity. The major downside of using static keys is that you give up the notion of *perfect forward secrecy*, meaning that if an attacker steals your static key, everything that was ever encrypted with it is compromised.

Generate a static key with the following command:

```
openvpn --genkey --secret static.key
```

The static key file is formatted in ascii and looks like this:

```
-----BEGIN OpenVPN Static key V1-----
e5e4d6af39289d53
171ecc237a8f996a
97743d146661405e
c724d5913c550a0c
30a48e52dfbeceb6
e2e7bd4a8357df78
4609fe35bbe99c32
bdf974952ade8fb9
71c204aaf4f256ba
eeda7aed4822ff98
fd66da2efa9bf8c5
e70996353e0f96a9
c94c9f9afb17637b
283da25cc99b37bf
6f7e15b38aedc3e8
e6adb40fca5c5463
-----END OpenVPN Static key V1-----
```

An OpenVPN static key file contains enough entropy to key both a 512 bit cipher key and a 512 bit HMAC key for authentication.

Copy **static.key** to the other peer via a secure medium such as **scp** or copy-paste in **ssh**.

Configuration File using a Pre-Shared Static Key

In our example, we will use OpenVPN configuration files. OpenVPN allows options to be passed on either the command line or in one or more configuration files. Options in configuration files can omit the leading "--" that is required for command line options.

Set up the following configuration files:

sample-config-files/static-office.conf

```
#
# Sample OpenVPN configuration file for
# office using a pre-shared static key.
#
# '#' or ';' may be used to delimit comments.

# Use a dynamic tun device.
# For Linux 2.2 or non-Linux OSes,
# you may want to use an explicit
# unit number such as "tun1".
# OpenVPN also supports virtual
# ethernet "tap" devices.
dev tun

# 10.1.0.1 is our local VPN endpoint (office).
# 10.1.0.2 is our remote VPN endpoint (home).
ifconfig 10.1.0.1 10.1.0.2

# Our up script will establish routes
# once the VPN is alive.
up ./office.up

# Our pre-shared static key
secret static.key

# OpenVPN 2.0 uses UDP port 1194 by default
# (official port assignment by iana.org 11/04).
# OpenVPN 1.x uses UDP port 5000 by default.
# Each OpenVPN tunnel must use
# a different port number.
# lport or rport can be used
# to denote different ports
# for local and remote.
; port 1194

# Downgrade UID and GID to
# "nobody" after initialization
# for extra security.
; user nobody
; group nobody

# If you built OpenVPN with
# LZO compression, uncomment
# out the following line.
; comp-lzo

# Send a UDP ping to remote once
# every 15 seconds to keep
# stateful firewall connection
# alive.  Uncomment this
# out if you are using a stateful
# firewall.
; ping 15

# Uncomment this section for a more reliable detection when a system
# loses its connection.  For example, dial-ups or laptops that
# travel to other locations.
; ping 15
; ping-restart 45
; ping-timer-rem
; persist-tun
; persist-key

# Verbosity level.
# 0 -- quiet except for fatal errors.
# 1 -- mostly quiet, but display non-fatal network errors.
# 3 -- medium output, good for normal operation.
# 9 -- verbose, good for troubleshooting
```

```
verb 3
```

sample-config-files/office.up

```
#!/bin/sh
route add -net 10.0.1.0 netmask 255.255.255.0 gw $5
```

sample-config-files/static-home.conf

```
#
# Sample OpenVPN configuration file for
# home using a pre-shared static key.
#
# '#' or ';' may be used to delimit comments.

# Use a dynamic tun device.
# For Linux 2.2 or non-Linux OSes,
# you may want to use an explicit
# unit number such as "tun1".
# OpenVPN also supports virtual
# ethernet "tap" devices.
dev tun

# Our OpenVPN peer is the office gateway.
remote 1.2.3.4

# 10.1.0.2 is our local VPN endpoint (home).
# 10.1.0.1 is our remote VPN endpoint (office).
ifconfig 10.1.0.2 10.1.0.1

# Our up script will establish routes
# once the VPN is alive.
up ./home.up

# Our pre-shared static key
secret static.key

# OpenVPN 2.0 uses UDP port 1194 by default
# (official port assignment by iana.org 11/04).
# OpenVPN 1.x uses UDP port 5000 by default.
# Each OpenVPN tunnel must use
# a different port number.
# lport or rport can be used
# to denote different ports
# for local and remote.
; port 1194

# Downgrade UID and GID to
# "nobody" after initialization
# for extra security.
; user nobody
; group nobody

# If you built OpenVPN with
# LZO compression, uncomment
# out the following line.
; comp-lzo

# Send a UDP ping to remote once
# every 15 seconds to keep
# stateful firewall connection
# alive.  Uncomment this
# out if you are using a stateful
# firewall.
; ping 15

# Uncomment this section for a more reliable detection when a system
# loses its connection.  For example, dial-ups or laptops that
# travel to other locations.
; ping 15
; ping-restart 45
; ping-timer-rem
; persist-tun
```

```
; persist-key

# Verbosity level.
# 0 -- quiet except for fatal errors.
# 1 -- mostly quiet, but display non-fatal network errors.
# 3 -- medium output, good for normal operation.
# 9 -- verbose, good for troubleshooting
verb 3
```

sample-config-files/home.up

```
#!/bin/sh
route add -net 10.0.0.0 netmask 255.255.255.0 gw $5
```

Starting the VPN in SSL/TLS mode

On Home, start the VPN with the command:

```
openvpn --config tls-home.conf
```

On Office, start the VPN with the command:

```
openvpn --config tls-office.conf
```

Starting the VPN in Static Key mode

On Home, start the VPN with the command:

```
openvpn --config static-home.conf
```

On Office, start the VPN with the command:

```
openvpn --config static-office.conf
```

Test the VPN

On Home, test the VPN by pinging Office through the tunnel:

```
ping 10.1.0.1
```

On Office, test the VPN by pinging Home through the tunnel:

```
ping 10.1.0.2
```

If these tests silently fail, you may want to re-edit the configuration files and set the verbosity level to 8 which will produce much more detailed debugging output. Also consult the FAQ for more information on troubleshooting.

If these tests succeed, now try pinging through the tunnel using machines on the private networks other than the OpenVPN gateway machines, to test the routing. Basically any machine on the **10.0.1.0/24** subnet should be able to access any machine on the **10.0.0.0/24** subnet and vice versa.

If that works, congratulations! If not, you might want to check out the OpenVPN Mailing List archives to see if anyone else has had a similar problem. If you don't find a resolution to your problem there, consider posting to the **openvpn-users** list.

Make the VPN DHCP-aware

If you recall, in our example network configuration, Home has a dynamic IP address which could change without warning. If you are using **dhcpcd** as your client daemon, it is easy to construct a script which will be run anytime the client's IP address changes. This script will be named something like **/etc/dhcpc/dhcpcd-eth0.exe**.

Basically, you should add a line to this script which will send a **SIGUSR1** or **SIGHUP** signal to the OpenVPN daemon such as:

```
killall -HUP openvpn
```

When OpenVPN receives this signal it will close and reopen the network connection to its peer, using the new IP address assigned by DHCP.

You should also use the **--float** option if you are connecting to a peer which may change its IP address

due to a DHCP reset.

It is also possible to handle DHCP resets with the **SIGUSR1** signal which is like **SIGHUP** except it offers more fine-grained control over which OpenVPN subsystems are reset. A SIGUSR1 signal can also be generated internally based on **--ping** and **--ping-restart**. The **--persist-tun** option allows a reset without closing and reopening the TUN device (which allows seamless connectivity through the tunnel across DHCP resets). The **--persist-remote-ip** option allows for preservation of remote IP address across DHCP resets. This allows both OpenVPN peers to be DHCP clients. The **--persist-key** option doesn't re-read key files on restart (which allows an OpenVPN daemon to be restarted even if its privileges were downgraded with **--user** or **--group**).

For more information on using OpenVPN in a dynamic IP address context, see the FAQ.

OpenVPN can also be used in cases where both ends of the connection are dynamic.

Start the VPN automatically on reboot

First make a directory to store OpenVPN keys and configuration files such as **/etc/openvpn**.

Decide whether you want to use TLS or Static Key mode and copy appropriate **.conf**, **.up**, **.key**, **.pem**, and **.crt** files to **/etc/openvpn**.

Protect your **.key** files:

```
chmod go-rwx /etc/openvpn/*.key
```

If you are using Linux **iptables**, edit the firewall configuration file **firewall.sh**, making changes appropriate to your site and copy to **/etc/openvpn**.

Make a startup script that looks something like this:

sample-config-files/openvpn-startup.sh

```
#!/bin/sh

# A sample OpenVPN startup script
# for Linux.

# openvpn config file directory
dir=/etc/openvpn

# load the firewall
$dir/firewall.sh

# load TUN/TAP kernel module
modprobe tun

# enable IP forwarding
echo 1 > /proc/sys/net/ipv4/ip_forward

# Invoke openvpn for each VPN tunnel
# in daemon mode.  Alternatively,
# you could remove "--daemon" from
# the command line and add "daemon"
# to the config file.
#
# Each tunnel should run on a separate
# UDP port.  Use the "port" option
# to control this.  Like all of
# OpenVPN's options, you can
# specify "--port 8000" on the command
# line or "port 8000" in the config
# file.

openvpn --cd $dir --daemon --config vpn1.conf
openvpn --cd $dir --daemon --config vpn2.conf
openvpn --cd $dir --daemon --config vpn2.conf
```

And make a shutdown script like this:

sample-config-files/openvpn-shutdown.sh

```
#!/bin/sh

# stop all openvpn processes

killall -TERM openvpn
```

Finally, add calls to **openvpn-startup.sh** and **openvpn-shutdown.sh** to your system startup and shutdown scripts or to your **/etc/init.d** directory.

Managing startup and shutdown of multiple OpenVPN tunnels

Here is a sample **/etc/init.d** script which will automatically create an OpenVPN tunnel for each **.conf** file in **/etc/openvpn**.

This script is installed by default if you install OpenVPN from an RPM package.

sample-scripts/openvpn.init

```
#!/bin/sh
#
# openvpn       This shell script takes care of starting and stopping
#               openvpn on RedHat or other chkconfig-based system.
#
# chkconfig: 345 24 76
#
# description: OpenVPN is a robust and highly flexible tunneling application that
#              uses all of the encryption, authentication, and certification features
#              of the OpenSSL library to securely tunnel IP networks over a single
#              UDP port.
#

# Contributed to the OpenVPN project by
# Douglas Keller <
 doug@voidstar.dyndns.org>
# 2002.05.15

# To install:
#   copy this file to /etc/rc.d/init.d/openvpn
#   shell> chkconfig --add openvpn
#   shell> mkdir /etc/openvpn
#   make .conf or .sh files in /etc/openvpn (see below)

# To uninstall:
#   run: chkconfig --del openvpn

# Author's Notes:
#
# I have created an /etc/init.d init script and enhanced openvpn.spec to
# automatically register the init script.  Once the RPM is installed you
# can start and stop OpenVPN with "service openvpn start" and "service
# openvpn stop".
#
# The init script does the following:
#
# - Starts an openvpn process for each .conf file it finds in
#   /etc/openvpn.
#
# - If /etc/openvpn/xxx.sh exists for a xxx.conf file then it executes
#   it before starting openvpn (useful for doing openvpn --mktun...).
#
# - In addition to start/stop you can do:
#
#   service openvpn reload - SIGHUP
#   service openvpn reopen - SIGUSR1
#   service openvpn status - SIGUSR2
#
# Modifications:
```

```
#
# 2003.05.02
#   * Changed == to = for sh compliance (Bishop Clark).
#   * If condrestart|reload|reopen|status, check that we were
#     actually started (James Yonan).
#   * Added lock, piddir, and work variables (James Yonan).
#   * If start is attempted twice, without an intervening stop, or
#     if start is attempted when previous start was not properly
#     shut down, then kill any previously started processes, before
#     commencing new start operation (James Yonan).
#   * Do a better job of flagging errors on start, and properly
#     returning success or failure status to caller (James Yonan).
#
# 2005.04.04
#   * Added openvpn-startup and openvpn-shutdown script calls
#     (James Yonan).
#

# Location of openvpn binary
openvpn=""
openvpn_locations="/usr/sbin/openvpn /usr/local/sbin/openvpn"
for location in $openvpn_locations
do
  if [ -f "$location" ]
  then
    openvpn=$location
  fi
done

# Lockfile
lock="/var/lock/subsys/openvpn"

# PID directory
piddir="/var/run/openvpn"

# Our working directory
work=/etc/openvpn

# Source function library.
. /etc/rc.d/init.d/functions

# Source networking configuration.
. /etc/sysconfig/network

# Check that networking is up.
if [ ${NETWORKING} = "no" ]
then
  echo "Networking is down"
  exit 0
fi

# Check that binary exists
if ! [ -f  $openvpn ]
then
  echo "openvpn binary not found"
  exit 0
fi

# See how we were called.
case "$1" in
  start)
        echo -n $"Starting openvpn: "

        /sbin/modprobe tun >/dev/null 2>&1

        # From a security perspective, I think it makes
        # sense to remove this, and have users who need
        # it explictly enable in their --up scripts or
        # firewall setups.

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

        # Run startup script, if defined
        if [ -f $work/openvpn-startup ]; then
            $work/openvpn-startup
        fi

        if [ ! -d  $piddir ]; then
            mkdir $piddir
        fi

        if [ -f $lock ]; then
            # we were not shut down correctly
            for pidf in `/bin/ls $piddir/*.pid 2>/dev/null`; do
```

```
            if [ -s $pidf ]; then
              kill `cat $pidf` >/dev/null 2>&1
            fi
            rm -f $pidf
          done
          rm -f $lock
          sleep 2
      fi

      rm -f $piddir/*.pid
      cd $work

      # Start every .conf in $work and run .sh if exists
      errors=0
      successes=0
      for c in `/bin/ls *.conf 2>/dev/null`; do
          bn=${c%%.conf}
          if [ -f "$bn.sh" ]; then
              . $bn.sh
          fi
          rm -f $piddir/$bn.pid
          $openvpn --daemon --writepid $piddir/$bn.pid --config $c --cd $work
          if [ $? = 0 ]; then
              successes=1
          else
              errors=1
          fi
      done

      if [ $errors = 1 ]; then
          failure; echo
      else
          success; echo
      fi

      if [ $successes = 1 ]; then
          touch $lock
      fi
      ;;
stop)
      echo -n $"Shutting down openvpn: "
      for pidf in `/bin/ls $piddir/*.pid 2>/dev/null`; do
        if [ -s $pidf ]; then
          kill `cat $pidf` >/dev/null 2>&1
        fi
        rm -f $pidf
      done

      # Run shutdown script, if defined
      if [ -f $work/openvpn-shutdown ]; then
          $work/openvpn-shutdown
      fi

      success; echo
      rm -f $lock
      ;;
restart)
      $0 stop
      sleep 2
      $0 start
      ;;
reload)
      if [ -f $lock ]; then
          for pidf in `/bin/ls $piddir/*.pid 2>/dev/null`; do
              if [ -s $pidf ]; then
                  kill -HUP `cat $pidf` >/dev/null 2>&1
              fi
          done
      else
          echo "openvpn: service not started"
          exit 1
      fi
      ;;
reopen)
      if [ -f $lock ]; then
          for pidf in `/bin/ls $piddir/*.pid 2>/dev/null`; do
              if [ -s $pidf ]; then
                  kill -USR1 `cat $pidf` >/dev/null 2>&1
              fi
          done
      else
          echo "openvpn: service not started"
          exit 1
      fi
```

```
        ;;
  condrestart)
        if [ -f $lock ]; then
            $0 stop
            # avoid race
            sleep 2
            $0 start
        fi
        ;;
  status)
        if [ -f $lock ]; then
            for pidf in `/bin/ls $piddir/*.pid 2>/dev/null`; do
                if [ -s $pidf ]; then
                    kill -USR2 `cat $pidf` >/dev/null 2>&1
                fi
            done
            echo "Status written to /var/log/messages"
        else
            echo "openvpn: service not started"
            exit 1
        fi
        ;;
  *)
        echo "Usage: openvpn {start|stop|restart|condrestart|reload|reopen|status}"
        exit 1
        ;;
esac
exit 0
```

Instantiate an OpenVPN daemon using **inetd** or **xinetd**

The common **xinetd** service can be used to automatically instantiate an OpenVPN daemon upon receipt of an initial datagram from a remote peer.

This xinetd configuration will cause xinetd to listen on UDP port 1194 for the first datagram of an incoming OpenVPN session (using a pre-shared key), at which time xinetd will automatically instantiate an OpenVPN daemon to handle the session. Note the use of the **--inactive** switch which will cause the OpenVPN daemon to time out and exit after 10 minutes of idle time. After the OpenVPN daemon exits for whatever reason, the xinetd service will resume listening on the port, and will again instantiate an OpenVPN daemon to handle additional incoming connections. Also note that xinetd will initially instantiate the OpenVPN daemon with *root* privileges, but OpenVPN will subsequently (after reading the protected key file) downgrade its privilege to *nobody*.

The key file can be generated with the following command:

```
openvpn --genkey --secret key
```

Note that each OpenVPN tunnel needs to run on its own separate port number, and needs its own xinetd configuration file. This is because OpenVPN needs specific information on each potential incoming connection, including key files, TUN/TAP devices, tunnel endpoints, and routing configuration. At this point in OpenVPN's development, it is not capable of handling any sort of *incoming connection template* that would allow a single configuration file to describe a large class of potential connecting clients. Since OpenVPN is implemented as a UDP server, it cannot take advantage of the infrastructure available to forking TCP servers which listen on a fixed port number, then dynamically fork off a new handling daemon for each client session. Nonetheless, incoming connection templates are on the wish list and may be implemented if there is sufficient interest and support from the developer and user community.

sample-config-files/xinetd-server-config

```
# An xinetd configuration file for OpenVPN.
#
# This file should be renamed to openvpn or something suitably
# descriptive and copied to the /etc/xinetd.d directory.
# xinetd can then be made aware of this file by restarting
# it or sending it a SIGHUP signal.
#
# For each potential incoming client, create a separate version
# of this configuration file on a unique port number.  Also note
# that the key file and ifconfig endpoints should be unique for
# each client.  This configuration assumes that the OpenVPN
# executable and key live in /root/openvpn.  Change this to fit
# your environment.

service openvpn_1
{
        type            = UNLISTED
```

```
        port            = 1194
        socket_type     = dgram
        protocol        = udp
        wait            = yes
        user            = root
        server          = /root/openvpn/openvpn
        server_args     = --inetd --dev tun --ifconfig 10.4.0.2 10.4.0.1 --secret /root/o
}
```

sample-config-files/xinetd-client-config

```
# This OpenVPN config file
# is the client side counterpart
# of xinetd-server-config

dev tun
ifconfig 10.4.0.1 10.4.0.2
remote my-server
port 1194
user nobody
secret /root/openvpn/key
inactive 600
```