Advanced OpenVPN Configuration

by Markus Feilner

This article deals with few features of advanced OpenVPN configuration like protecting clients through a firewall behind a tunnel, distributed compilation through VPN tunnels with **distcc**, and Authentication methods.

Individual Firewall Rules for Connecting Clients

One striking possibility OpenVPN offers is a setup where:

- An OpenVPN machine acts as a server that protects the company's network, admitting access for OpenVPN clients.
- The clients are automatically assigned IPs by the server.
- The clients are equipped with certificates, and identified and authorized by these certificates.

The scripting parameter `learn-address` in the server's OpenVPN configuration file will have the server execute a script whenever an authorized client connects to the VPN and is assigned an address. This parameter takes the full path to a script as an option:

```
learn-address /etc/openvpn/scripts/openvpnFW
```

In this example, the script `openvpnFW` will be executed each time a client is assigned an IP address and will be passed three variables by the OpenVPN server process:

1. `$1`: The action taken; this may be one of add, delete, update
2. `$2`: The IP assigned to the client connecting
3. `$3`: The common name in the subject line of the client's certificate

Add the line `learn-address` /etc/openvpn/scripts/openvpnFW to your OpenVPN server configuration file and edit the file /etc/openvpn/scripts/openvpnFW to be like the following. These lines will show how to make use of these parameters in a short Linux shell script:

```
#!/bin/sh
LOGFILE=
DATE=`/bin/date`
echo $DATE $1 $2 $3 >> $LOGFILE
```

This script will only export the variables passed to the logfile, including a timestamp that is added by the command `date`. Stop and start your tunnel a few times. Now let's have a look at the file /var/log/openvpn/connections.log:

```
Mi Feb 1 04:33:53 CET 2006 update 10.99.0.3 mfeilner
Do Feb 2 04:34:33 CET 2006 update 10.99.0.3 mfeilner
Fr Feb 3 04:34:14 CET 2006 update 10.99.0.3 mfeilner
Sa Feb 4 04:34:53 CET 2006 update 10.99.0.3 mfeilner
So Feb 5 04:34:43 CET 2006 update 10.99.0.3 mfeilner
```

This example shows a VPN client reconnecting every day. This alone might yet be an interesting feature, if you want to keep track of your users and their VPN connections. However, we can do more. Let's add some more lines to our `openvpnFW` script:

```
if [ $1 = add ]
then
/etc/openvpn/scripts/$2.FW_connect.sh
fi
if [ $1 = delete ]
```

```
then
/etc/openvpn/scripts/$2.FW_disconnect.sh
fi
```

Two simple tests are run and, depending on the content of the variable `$1`, different firewall scripts are executed. Let's express this in brief. If the first variable passed is `add`, then the script `/etc/openvpn/scripts/$2.FW_connect.sh` is run, where `$2` will be replaced by the IP of the client connecting. If for example a client `mfeilner` connects and is assigned the IP `10.99.0.3`, then the variables passed to this script `openvpnFW` will be:

```
add 10.99.0.3 mfeilner
```

And the script run will be called: `/etc/openvpn/scripts/10.99.0.3.FW_connect.sh`

However, if the variables passed to `openvpnFW` are the following:

```
delete 10.99.0.3
```

then the script `/etc/openvpn/scripts/10.99.0.3.FW_disconnect.sh` will be executed.

You would have already guessed that these two scripts contain firewall rules (like `iptables` statements) for the client with the certificate `mfeilner`. Even though all of this could be done within one single script, it is preferable to have the tests and firewall rules split up in several scripts.

This setup can become very powerful and fairly complex. A client that has its default route set through the tunnel can be allowed selective Internet access, simply by enabling or disabling, routing or forwarding. And access to the local servers can also be easily managed: e.g. A SAP server might only be available for roadwarriors from 7 am to 6 pm, whereas during the night firewall rules protect the server.

Using a Client Configuration Directory with Per-Client Configurations

Another striking feature of OpenVPN is the fact that we can have client configurations pushed through the tunnel on creation and use client-specific configurations, which are simply set by the subject line of the client's certificate. An appropriate server configuration file may look like the following:

```
port 443
dev tun0FIT
ca /etc/openvpn/certs/ca.crt
cert /etc/openvpn/certs/firewall.crt
key /etc/openvpn/certs/firewall.key
dh /etc/openvpn/certs/dh2048.pem
tls-auth /etc/openvpn/certs/ta.key 0
auth SHA1
cipher AES-256-CBC
tls-cipher DHE-RSA-AES256-SHA
server 10.179.0.0 255.255.0.0
ifconfig-pool-persist /etc/openvpn/ipp.txt
client-config-dir clients
keepalive 10 120
resolv-retry 86400
comp-lzo
status /var/log/openvpn/status.log
log /var/log/openvpn/main.log
tls-server
verb 3
```

There are three lines that are relevant in this context:

1. `server 10.179.0.0 255.255.0.0`: This tells OpenVPN on this machine to act as a server and automatically distribute IP addresses to clients connecting.
2. `ifconfig-pool-persist /etc/openvpn/ipp.txt`: This makes OpenVPN keep a list of certificate to IP relationships, so that a client connecting will (probably) always have the same IP.

3. `client-config-dir clients`: This has OpenVPN look in the directory "clients" for a client-specific configuration file when a client connects.

A client configuration file must have a name matching the `CN` in the `Subject` line of the certificate. If a client connects with a certificate containing the following subject:

```
(...)
Subject: C=DE, ST=Bayern, L=Regensburg, O=Feilner-IT,
CN=mfeilner/emailAddress=mfeilner@feilner-it.net
(...)
```

Then the server will look if the directory clients contain a configuration file named `mfeilner`. This file may contain push options like the following:

```
ifconfig-push 10.179.0.3 10.179.0.4
push "route 10.1.0.0 255.255.0.0"
```

In this scenario, this client will always have the IP address `10.179.0.3` and is told about a network (10.1.0.0) behind the tunnel. Thus, if we use different client configurations, we can control the routing and network configuration for every client. It's simple to grant access to the network by activating or deactivating a client's routing on connecting, but we must always remember that this offers no real protection, because every local administrator could also activate this routing on the client.

On the client configuration, the parameter `client` must be present. If we want to have the client redirect its default gateway through the tunnel, we simply need to add the parameter `redirect-gateway`.

Redirecting the client's default gateway is another excellent feature of OpenVPN, especially when combined with HTTP-proxy tunneling. The parameter `redirect-gateway` causes three steps:

1. A static route to the other tunnel partner is created.
2. The old default gateway is deleted.
3. A new entry for the default gateway is created (pointing to the IP address of the other tunnel endpoint).

Of course we can enter these steps manually, if we like. The `route` command will help us here:

```
debian01:~# route add 172.16.103.2 gw 172.16.247.1
debian01:~# route del default
debian01:~# route add default gw 10.179.10.2
debian01:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
172.16.103.2    172.16.247.1    255.255.255.255 UGH   0      0        0 eth0
10.179.10.2     0.0.0.0         255.255.255.255 UH    0      0        0 tunVPN0
172.16.247.0    0.0.0.0         255.255.255.0   U     0      0        0 eth0
172.16.76.0     10.179.10.2     255.255.255.0   UG    0      0        0 tunVPN0
192.168.250.0   0.0.0.0         255.255.255.0   U     0      0        0 eth1
0.0.0.0         10.179.10.2     0.0.0.0         UG    0      0        0 tunVPN0
debian01:~#
```

First, we added a static route to the VPN partner (`route add 172.16.103.2 gw 172.16.247.1`). Then we deleted the old default route (`route del default`), and as a last step we created the new default route with `route add default gw 10.179.10.2`. From this moment on, all traffic not destined to the VPN partner's public IP will be routed through the tunnel, as the output of `route -n` will show. Because the routing entries will be useless when the VPN partner's IP changes, it is a good idea to have OpenVPN set the routing for us.

Using Authentication Methods

OpenVPN can be used with authentication based on shared secrets (static keys) and X.509 certificates. Another useful option for authentication is authentication plug-ins called with the configuration parameter `auth-user-pass-verify`, which can be used together with both methods mentioned before. For example, in a certificate-based VPN, we can use an authentication plug-in to make sure that only a user knowing the appropriate username/password combination can start the tunnel. This may be a convenient additional level of security for laptops or other roadwarrior machines.

While certificates in this context tend to protect and authenticate machines rather than users, username/password combinations are useful for VPNs that are started by a human. The Windows GUI will pop up a small authentication window where the user must enter a username and password. The VPN client takes these values and sends them to the VPN server, which starts the plug-in program (as configured in `auth-user-pass-verify`) to validate the combination. If the authentication program returns an OK, authentication was successful, and the tunnel is created. The tunnel will only be established if the password is correct.

For this purpose, the following configuration parameters must be added: In the server configuration file, add `auth-user-pass-verify /path/to/your/auth/script` to your server configuration and `auth-user-pass` to your client's configuration. The following table shows the usage of these parameters:

Parameter	Allowed options	Usage	Function
`--auth-user-pass-verify`	`<script> <method>`	Server configuration	Activates server's authentication and defines the name of the authentication script and the method to use for username/password handling
`--auth-user-pass`	`<file>`	Client configuration	Activates client's authentication and optionally defines a file where username and password are stored

On SuSE systems there are some example scripts (like `auth_pam.pl`) provided with OpenVPN, which can be found in `/usr/share/doc/packages/openvpn/sample-scripts`. But a typical scenario for such an authentication may be a local LDAP server. LDAP is the system-independent state of the art for all modern directory services both in open-source servers and also in Microsoft's Active Directory Service. The following overview will give you some hints on how to create an authentication plug-in using your own LDAP authentication for OpenVPN.

On a Linux system with the LDAP client tools installed, the command `ldapwhoami` can be used for testing username/password pairs against an LDAP server. In the following examples the LDAP server is `10.10.10.1`, the user `mfeilner`, and the password is `correct_password`. The string `uid=mfeilner,ou=Feilner-it_Users,dc=feilner-it,dc=home` must be adapted to the settings on your LDAP server. Here is the output of the `ldapwhoami` command:

```
suse01:/var/log # ldapwhoami -x -h 10.10.10.1 -D uid=mfeilner,ou=Feilner-
it_Users,dc=feilner-it,dc=home -w correct_password
dn:uid=mfeilner,ou=Feilner-it_Users,dc=feilner-it,dc=home

suse01: # ldapwhoami -x -h 10.10.10.1 -D uid=mfeilner,ou=Feilner-it_Users,dc=feilner-
it,dc=home -w wrong_password
ldap_bind: Invalid credentials (49)
```

The first command will give a return code of "0", whereas the second command, resulting in a failed authentication returned a value of "1".

This article is written by Markus Feilner, author of the book OpenVPN: Building and Integrating Virtual Private Networks by Packt Publishing.

For further details please visit http://www.packtpub.com/openvpn/book

Markus Feilner is a Linux author, trainer, and consultant from Regensburg, Germany, and has been working with open-source software since the mid 1990s. His first contact with UNIX was a SUN cluster and SPARC workstations at Regensburg University (during his studies of geography). Since the year 2000, he has published several documents used in Linux training all over Germany. In 2001, he founded his own Linux consulting and training company, Feilner IT (http://www.feilner-it.net). Furthermore, he is an author, currently working as a trainer, consultant, and systems engineer at Millenux, Munich, where he focuses on groupware, collaboration, and virtualization with Linux-based systems and networks. He is interested in anything about geography, traveling, photography, philosophy (especially that of open-source software), global politics, and literature, but always has too little time for these hobbies.