Building SSH VPNs with OpenSSH

Very Painless Networks

OpenSSH VPN technology, installed by default on most Linux, BSD, and Unix systems, lets you mix and match different clients and servers easily. *By Kurt Seifried*

When it comes to VPN performance, the problems used to revolve around having enough CPU power to encrypt and decrypt network traffic (watching a Pentium III struggle with 100Mb traffic was always fun). Nowadays, CPU power is cheap, and now with Intel AES-NI and other vendors supporting AES with native instructions, the good news is that most systems can easily keep up with a gigabyte or more of encrypted traffic. One of the biggest performance problems faced by VPNs now is ISPs that either shape (that is to say they slow down) VPN traffic or block it completely.

Some VPN Options

On Linux, you have a number of popular VPN options: IPsec, OpenVPN, PPTP, and SSH. All four will get you a secure network between a combination of hosts and networks. As far as authentication, virtually all of them support the use of pre-shared keys or secrets, and they can all be configured to use PAM, LDAP, and a variety of other authentication back ends. However, they have some important differences. The biggest difference is the transport mechanisms used. IPsec does basically everything at the network layer – except the mutable headers (those that might be changed in transit, e.g., the TTL and checksum). OpenVPN, PPTP, and SSH basically all operate at the application layer and simply encrypt data and then send it as the payload within a data packet. The main difference is that OpenVPN can use either TCP or UDP, whereas PPTP and SSH are restricted to TCP, only without making major modifications or using additional software to transfer the TCP packets over UDP.

KURT SEIFRIED

Kurt Seifried is an Information Security Consultant specializing in Linux and networks since 1996. He often wonders how it is that technology works on a large scale but often fails on a small scale.

The Trouble with Standards

Why aren't VPN options like IPsec more popular? IPsec provides authentication and encryption of network traffic. IPsec is supported on virtually every operating system and most network devices. Most IPsec servers support a variety of strong encryption options.

So, why don't we use IPsec more? Although IPsec has a well-defined standard, it is implemented by many different vendors, some of which have added extensions or chosen slightly different ways to implement certain things. In my experience (between OpenBSD, Linux, Windows, Cisco, and a few other network equipment makers), configuration and management of different clients and servers can be challenging.

Generally, if you stick to a heterogeneous environment (e.g., all OpenBSD, all Windows, or all Cisco servers and Cisco client software on endpoints), you will be mostly fine, but if you start mixing and matching, you will likely be in for a world of pain.

Why SSH?

Here, I will focus on building SSH VPNs. One of the more compelling advantages of SSH is that virtually all network platforms with a TCP/IP stack support SSH. Even better, about 99% of all systems running an SSH server/client are using OpenSSH [1], which ensures an extremely high degree of compatibility between them. About the only systems that don't ship with OpenSSH are Windows and lower end network equipment. Virtually all "proper" VPN setups require root-level access because you will be modifying your systems' network interfaces and routing tables. However, if you are only using OpenSSH port forwarding, you can do this as a normal user with no additional privileges.

OpenSSH on Windows

OpenSSH installation on Linux is generally not required because almost all vendors ship it by default and enable it for remote administration. To install OpenSSH on Windows, you'll need to download Cygwin [2] then, during installation, simply choose OpenSSH (in *All* | *Net*) for installation. Once installed on Windows, you need to enable OpenSSH: Right-click on the Cygwin menu item or icon, choose *Run as administrator* (running it as a normal user will not allow you enough access to configure OpenSSH), then simply run:

```
ssh-host-config
```

When asked to enable privilege separation, say yes. You'll then be asked to create a new local account, to which you also agree, then you will be asked whether you want to install OpenSSH as a service (you do). For the *value of the CYGWIN daemon*, enter *ntsec*. Finally, you will be prompted to create a user for OpenSSH to run as; follow the prompts and you should be finished. Once done, you can enable the SSH server as

```
net start sshd
```

from the Cygwin command line. To log in, you need to run the `ssh-user-config` script, which creates SSH keys and so on for the currently logged in user so that automated logins work (which, if you're using OpenSSH as a VPN, you probably want). Once the OpenSSH client and server are installed, you can configure it to do something useful.

OpenSSH Port Forwarding

VPNs are great, but often they are overkill. Not everyone has a corporate network with hundreds or thousands of servers a user might need access to. Often, all you need is access to a few internal resources, such as email, file sharing, and web services. In this case, you can either set up an OpenSSH server inside the network or allow direct access (e.g., to the mail server, web server, etc.). Generally, setting up a separate server is much safer and easier: Allow port 22 to an external IP then allow that DMZ host to connect to your trusted systems. Please note that for this to work, the `AllowTcpForwarding` option in `sshd_config` needs to be set to `yes`; however, this is the default setting (so it should work with no problems).

For example, you can allow users to connect to ports 25 (SMTP) and 143 (IMAP) from the DMZ host. By default, local port forwarding only binds to localhost (127.0.0.1), which is ideal for a laptop. To set up SSH port forwarding for localhost port 2500 to port 25 and port 14300 to port 143 on the mail server, you can use commands like:

```
ssh user@openssh.example.org ↩
   -L 2500:mail.example.org:25
ssh user@openssh.example.org ↩
   -L 14300:mail.example.org:143
```

In the above scenario, every user's laptop mail client would connect to localhost port 25 (to send email) and port 143 (to receive email).

It's important to note that, by default, when forwarding ports, OpenSSH only binds to localhost; thus, by extension, only local users can connect to the forwarded port. However, what if you want to set up a proxy server at a branch office so all clients can connect securely to your mail server through an SSH port forwarding connection? Simply enable `GatewayPorts` to `yes` so that ports will bind to external servers and not just localhost.

VPN System Config

What if you need a "real" VPN? On the OpenSSH server handling clients, you need to enable `tun` (tunnel) interfaces and packet forwarding and potentially modify firewall rules. Setting up a tun (tunnel) interface varies a lot.

On Fedora 16 and later, you will want `tunctl`, which allows you to assign the `tun` interface to a nonprivileged user. On other distros, you can simply `ifconfig tun0 up` or add it to the appropriate network startup scripts. You will also need to enable IP forwarding, by modifying the `/etc/sysctl.conf` file:

```
# Controls IP packet forwarding
net.ipv4.ip_forward = 1
```

and simply reboot. If you want to enable IP forwarding, you can run

```
sysctl -w net.ipv4.ip_forward = 1
```

You might also need to enable forwarding in your firewall rules, although many Linux distributions allow forwarding by default.

OpenSSH VPN Config

The use of OpenSSH for "real" VPNs has one critical configuration directive: `PermitTunnel`. For tunneling to work, you need to enable this by either specifying `ethernet` for Layer 2 tunneling or `yes` for Layer 3 and Layer 2 tunneling. However, you probably don't want to give users administrative access to your OpenSSH server so that they can log in and run commands to modify the networking configuration, for example. The easiest way to deal with this is to use either the `ForceCommand` directive, to run a set program or script once the user logs in, or to modify the `~/.ssh/.authorized_keys` file, to include the commands you want to run and any SSH options:

```
command="/opt/bin/vpn-setup.sh",↩
   no-port-forwarding,↩
   no-X11-forwarding,↩
   no-pty <ssh-rsa key-string-here>
```

Of course, you must set permissions on the `.authorized_keys` file so that the user cannot modify it (both the file and the parent directory). Alternatively, you can use `ForceCommand` to specify a command that is run, combined with `Match`. By specifying a group of users, you can easily set up accounts that are allowed to run only the VPN script.

OpenSSH is the only VPN technology installed by default on almost every Linux, BSD, and Unix system, and it is allowed through most host-based firewalls by default. Add to that all the work done by the OpenBSD project to secure it (privilege separation, code audits, etc.), and OpenSSH can't be beat. ■■■

INFO

[1] OpenSSH: *http://www.openssh.org/*
[2] Cygwin: *http://www.cygwin.com/*