Linking internal networks to provide large-scale virtual networks

Low-Budget VPNs

A large number of hardware and software solutions are now contending for the privilege of linking up company sites or the flat-sharing community in "Virtual Private Networks" (VPNs). If you have a low budget and requires a minimal amount of effort to configure, then Linux has the answer.

BY MICHAEL RENNER

With the Internet providing the means of remote access to shared machines from all over the world, the idea of a virtual local network makes sense for global projects, virtual enterprises, or even couples that live at different addresses. The borders of the LAN are no longer defined by the physical proximity of an office or flat-sharing community but by team membership.

However, "local" networks, where the Internet assumes the role of the wire, need to be secured to ensure the privacy of the traffic they carry. Privacy is provided by using a tunnel, where only the outer core of the tunnel is visible to the Internet and the data traffic is only visible to computers at the endpoints.

vpn-pppssh, from [1] or this month's subscription CD, provides a script based option for building a virtual private network (VPN). The advantage: This approach uses software that is available on almost any Linux computer. A single line in the script is all it takes to set up the VPN tunnel. The remaining code makes the VPN more convenient and takes care of **routing** the payload packages.

Secure Shell *ssh* [2], see the article in this issue, provides reliable encryption for the data as they cross the Internet; the "*P*oint to *P*oint *P*rotocol", PPP, ensures that various types of transports, such as TCP, UDP, and even ICMP can pass through the VPN tunnel (see Box 1).

Although this is a simple solution, it is not entirely restricted to connecting two sites. Some additional work will allow you to link up two or three individual networks or computers to form a single large-scale virtual network.

Of course, an extremely basic solution does have some disadvantages. For example, the connection will not be re-established automatically if it goes down. Additionally, the data transfer rate is less than one might expect from a DSL link. Each data packet transferred to another subnet is first encapsulated in another packet, and this increases the overhead. You should avoid connecting up too many networks using this technique. The reliability of the connection declines with every network you attach, and soon reaches a threshold where the number of dropped connections makes working on the VPN more or less impossible.

The Scenario

Looking at the scenario from a practical point of view reveals that it makes little difference whether your are linking up two flat-sharing communities with Windows and Linux computers and a DSL based Internet connection, or connecting two sites that belong to a single enterprise. The technology is the important thing.

The more powerful gateway computer will be assuming the role of the server at the tunnel endpoint; the less powerful machine will be used as the client. In the example of two flat-sharing communities shown in Figure 1, the computer in house 2 has a more powerful processor than its counterpart in house 5, so this makes the role assignments simple; house 2 will be hosting the server, while the other is used as a client.

The shell script, *vpn-pppssh*, calls a few programs that you will need to install additionally. These include Secure Shell, SSH, as already mentioned; Secure Shell will be used to establish the data tunnel and transport encrypted packets across the Internet. *pppd* will be used as the transport program. Both the SSH package and the PPP daemon need to be installed on both ends of the tunnel to allow communication with the other end of the tunnel to take place.

To provide a modicum of security for the server with respect to the attached clients, VPN requests should be handled by a non-privileged system account created specially for this purpose. But even if the client side VPN script does not log on to the server as *root*, some processes launched after logging on (*pppd* for one) will need superuser privileges. The ***sudo*** program provides a way out of this twist. If *ssh* and *pppd* are installed on both machines and *sudo* is additionally

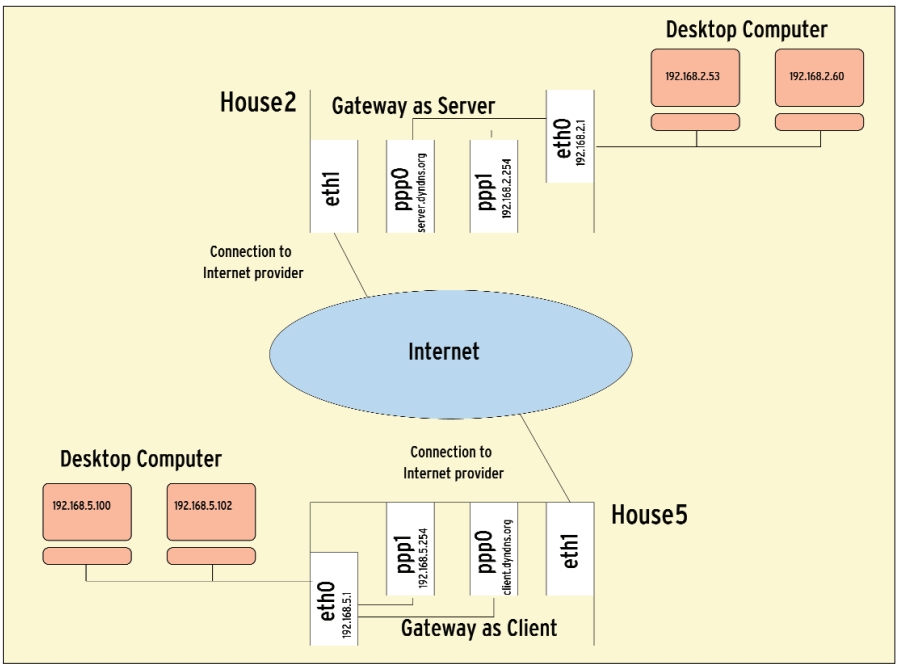

Figure 1: A simple VPN tunnel with a DSL link to the intranet

installed on the server, that is the Internet gateway machine in house 2, you can start configuring the VPN.

Donkey Work

Ensure that you are working as *root*, so you have the correct privileges and then type the following command to create the unprivileged *vpn* user account which the VPN script will then use to log on to the server:

```
adduser --system --shell /bin/bash --gecos "VPN Account" --disabled-password --group vpn
```

The server sysadmin from house 2 will need to add an entry to the *sudo* configuration file, */etc/sudoers*, to allow *vpn* to use *sudo* to call *pppd* in the script. The sysadmin can launch *visudo* to edit the script. Calling *visudo* will launch the editor specified by the *EDITOR* or *VISUAL* environment variable; this typically defaults to *vi*. You can then toggle vi to insert mode by pressing [i], type the following lines, and then press [Esc] *:wq* to save and quit:

```
vpn ALL=NOPASSWD:/usr/sbin/pppd
vpn ALL=NOPASSWD:/sbin/route
```

The second line allows *vpn* to call the */sbin/route* tool and change the routing table.

Having to type the login password for *vpn* manually every time a connection to the server is established would defeat the object of the exercise of automating the VPN. Instead of a password prompt, we will now be using an alternative authentication technique, the **public key method** provided by Secure Shell.

When a client attempts to log on to the server, the server side Secure Shell **Daemon** *sshd* reads the public key deposited on the server by the client. The server uses this key to encrypt a short message which it returns to the client.

If the public key deposited on the server matches the client's private key, the client will be capable of decrypting the "challenge". The client will then send proof of this ability to the server, and the server will in turn permit a controlled but automated login.

If client side */root/.ssh/id_dsa* and */root/.ssh/id_dsa.pub* files do not contain a keypair, you will need to run the

Box 1: The TCP/IP Protocol Family

If machines are to exchange information, a ruleset is required to specify how this exchange will take place. How should one machine inquire whether it is permitted to deliver data, how should the machine at the other end reply, in what format will information be transferred from A to B, and how can computer A know that computer A really has transferred what it intended to transfer? These rules are referred to as protocols and are comparable with agreements in diplomatic protocols.

Communication on the Internet is based on the TCP/IP protocol suite. The underlying "*I*nternet *P*rotocol" is responsible for forwarding data. It determines what route to use to send packets and fragments larger data packets. As IP is not interested in whether these packets actually arrive at their intended destination, there are extensions that provide a control structure. The "*T*ransmission *C*ontrol *P*rotocol" specifies that a machine will keep on re-transmitting a package until receipt and integrity of the packages is confirmed by the other end of the connection. But there is a price for this assurance in the form of non-trivial overheads.

UDP ("*U*ser *D*atagram *P*rotocol") is a minimalist protocol, based on IP but without control mechanisms. UDP is used for Internet services that do not depend on every single packet arriving, but require quick, low-overhead transfers, as in the case of streaming. The "*I*nternet *C*ontrol *M*essage *P*rotocol" ICMP is used to exchange management messages between the entities involved in the data transmission. The most popular ICMP application is the *ping* command.

The "*I*nternetwork *P*acket *E*xchange" protocol, IPX, has nothing in common with TCP/IP. It provides addressing and routing facilities for packets within and between LANs and has been more or less completely supplanted by TCP/IP family protocols.

GLOSSARY

Routing: *A routing table, which is managed by the operating system, defines the path data packages take between networks (the route).*

sudo: *This tool gives the sysadmin the ability to allow specific users to run specific programs with root privileges. The capabilities of this method are far greater than those of the well-known SUID bit [5].*

Public key method: *Asymmetric cryptography always involves a pair of interrelated keys: the public key and a secret or private key.*

Daemon: *A program that listens in the background for external requests. The program name is normally a combination of the protocol with a "d" attached to it, such as pppd or sshd.*

DSA: *"Digital Signature Algorithm", a secure algorithm used for signing messages.*

Listing 1: Generating an SSH Key

```
client:~# ssh-keygen -t dsa
Generating public/private dsa key pair.
Enter file in which to save the key (/root/.ssh/id_dsa):
Enter passphrase (empty for no passphrase): [Enter]
Enter same passphrase again: [Enter]
Your identification has been saved in /root/.ssh/id_dsa.
Your public key has been saved in /root/.ssh/id_dsa.pub.
```

ssh-keygen command to create one (see Listing 1). The *-t dsa* parameter specifies **DSA** as the encryption method. Do not hit [Enter] when prompted to enter a passphrase – if you do, you will be expected to enter a password every time a connection is established.

The public key belonging to the key-pair recently generated on the client must now be added to the key catalog for the *vpn* system account on the server. Although you might expect this to work, you cannot simply copy the */root/.ssh/id_dsa.pub* file from the client to */home/vpn/.ssh/authorized_keys* on the server!

authorized_keys might contain a large number of public keys belonging to other machines, especially if multiple computers need to login using the SSH public key method. For this reason, you should transfer the *id_dsa.pub* file with the new key to the server first (using a secure method, such as *scp*, an USB stick or even a floppy), and then using *cat* to append the file to *authorized_keys*:

```
cat /mnt/usb/id_dsa.pub >> ➥
/home/vpn/.ssh/authorized_keys
```

Swimming on Dry Land

After all this preparatory work, it's time for a few initial tests. Ensuring that you are *root*, use *ssh* to log on to the server from the client as the *vpn* user. Type *yes* (see Listing 2) if the *ssh* command prompts you to confirm that you trust the other machine.

Listing 2: Initial Tests

```
01 client# ssh server.dyndns.org -l vpn
02 The authenticity of host 'server.dyndns.org (217.81.158.48)' can't be
   established.
03 DSA key fingerprint is
   80:f5:44:74:f4:5e:d0:50:32:12:88:f7:f4:31:75:82.
04 Are you sure you want to continue connecting (yes/no)? yes
05 Warning: Permanently added 'server.dyndns.org' (DSA) to the list of
   known hosts.
06 Linux server 2.4.16 #6 Mon Apr 20 11:03:22 EST 2003 i686 unknown
07 No mail.
08 vpn@server:~$
09 vpn@server:~$ sudo /usr/sbin/pppd noauth
10 ~9}#ÀZ}!}!} }9}"}k} }r} }'}%}zt2-*}'}"}
```

Listing 3: Setting up the VPN script as an init script for Debian

```
ln -s /etc/init.d/vpn-pppssh /etc/rc0.d/K10vpn-pppssh
ln -s /etc/init.d/vpn-pppssh /etc/rc1.d/K10vpn-pppssh
ln -s /etc/init.d/vpn-pppssh /etc/rc2.d/S99vpn-pppssh
ln -s /etc/init.d/vpn-pppssh /etc/rc3.d/S99vpn-pppssh
ln -s /etc/init.d/vpn-pppssh /etc/rc4.d/S99vpn-pppssh
ln -s /etc/init.d/vpn-pppssh /etc/rc5.d/S99vpn-pppssh
ln -s /etc/init.d/vpn-pppssh /etc/rc6.d/K10vpn-pppssh
```

If you are able to log on without entering a password, launch *pppd* on the server. The unintelligible characters in the response shown in Listing 2 are quite normal. They show that *pppd* on the server computer is running, and is now attempting to talk to *pppd* at the other end. As this daemon is not yet running, you will want to terminate the process by pressing [Ctrl-C] and continue with other settings.

Nobody will want to launch the VPN manually every time they need access to it, so the best approach is to tell your computer to establish the VPN automatically on booting or when you access the Internet. If the computers involved use leased lines to attach to the Internet, you can simply copy the *vpn-pppssh* script to */etc/init.d/* on the client computer and create links to the script in the **runlevel** directories appropriate to your distribution. Listing 3 shows how to do this on Debian.

For computers with a DSL connection whose IP changes every 24 hours, it is preferable to establish the VPN on Internet access: Copy *vpn-pppssh* to */usr/local/bin/* and create an addition script (see Listing 4) in */etc/ppp/ip-up.d/* to launch *pppd* when ever the machine accesses the Internet, so establishing the VPN.

Practical Details

It would be impractical for the client in house 5 to need to know the IP assigned to the server the last time it accessed the Internet. The flat-sharing community in house 2 uses a dynamic DNS service to avoid this, mapping the dynamic IP assigned by the server's Internet provider to a static hostname as a result.

In our example, the server machine has the internal IP address *192.168.2.1* from the local network perspective. The gateway machine in house 5 has been assigned the internal IP address *192.168.5.1*. Additionally, the values of the *SERVER_HOSTNAME*, *SERVER_USERNAME*, *SERVER_IFIPADDR*, and *CLIENT_IFIPADDR* script variables have been modified to reflect the local scenario (see Listing 5).

In order to modify the routing table, you need to know the IP address blocks used at both ends of the tunnel. A few other details are also needed.

To ensure that packets address to the server side *192.168.2.0* network will be sent through the VPN tunnel, it is important to insert a command at the end of the startup section of the *vpn-pppssh* script to add an entry to the client side routing table:

```
route add -net $SERVER_NET gw ➥
$SERVER_IFIPADDR
```

Another command opens an *ssh* connection to the server and uses *sudo* to enter the correct route to the client:

```
ssh -o Batchmode=yes $SERVER_➥
HOSTNAME -l $SERVER_USERNAME ➥
sudo /sbin/route add -net ➥
$CLIENT_NET gw $CLIENT_IFIPADDR
```

The modifications explained in the "Swimming on Dry Land" section allow the client to launch the VPN script automatically after establishing a connection to the Internet provider. The two additional commands in the script modify the routing table. It should not take more than about one minute for connectivity to the computers at the other end of the tunnel to be established; you can easily test for this using *ping*. Web servers or printers in the other subnet can now be accessed like local devices by users in house 5. Of course the users in house 2 will have similar benefits.

For Perfectionists

There is still a lot of room for improvement with a VPN like the one described in our example. If machines are to be able to use intuitive hostnames rather than numeric IPs to access the computers at the opposite ends of the tunnel, the IP addresses and hostnames of the computers in both subnets will need to be added to the */etc/hosts* file on every machine. Instead, you might consider using NIS to propagate the names or even set up a **secondary nameserver**.

The issue of dropped connections referred to earlier could be resolved by allowing a script in */etc/ppp/ip-down.d/* to check whether the connection has gone down.

It would even be possible to use *vpn-pppssh* with a hardware based router. To do so, the former would simply need to forward SSH port 22 to a Linux machine. You could then configure the Linux machine as the gateway to the other subnet for the computers in the local network. To ensure that incoming packets really are forwarded, *root* needs to enable forwarding as follows:

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

However, if you need connectivity between a larger number of individual networks, you might like to consider more advanced VPN solutions, such as OpenVPN [3] and FreeS/WAN [4]. ■

Listing 4: Establishing the VPN on accessing the Internet

```
#!/bin/sh
if [ "$PPP_IPPARAM" = "vpn" ]; then
        # We do not want the VPN to start up automatically
        exit 0
fi
/usr/local/bin/vpn-pppssh start
```

Listing 5: Modifying vpn-pppssh

```
01 # The hostname or IP address of the server
02 SERVER_HOSTNAME=server.dyndns.org
03 # The name of the user, whose server account will be used to
   establish the VPN
04 SERVER_USERNAME=vpn
05
06 # The IP address of the VPN on the server:
07 SERVER_IFIPADDR=192.168.2.254
08
09 # The IP address of the VPN on the client:
10 CLIENT_IFIPADDR=192.168.5.254
11
12 # The address scope of the internal network server side:
13 SERVER_NET="192.168.2.0/24"
14
15 # The address scope of the internal network client side:
16 CLIENT_NET="192.168.5.0/24"
```

GLOSSARY

Runlevel: *Operating mode of the Linux system. The various runlevels are distinguished by the services that are launched on entering each runlevel. The /etc/inittab file specifies the runlevel used by default when the system boots.*

NIS: *"Network Information System". A NIS server distributes configuration information to the attached clients, for example user files in the form of a password file or configuration files for an auto-mounter that mounts replaceable media when needed. The /etc/nsswitch.conf file specifies the data sources (local files, NIS) a computer can access.*

Secondary nameserver: *A secondary nameserver polls DNS information from the primary nameserver authorized for the domain. The secondary may poll a full copy of the data on booting. DNS information provided by a nameserver of this type may correspond to the information known to the primary nameserver at times, but this is not typically a drawback.*

INFO

[1] vpn-pppssh and howto: *http://www.tldp.org/HOWTO/mini/ppp-ssh/*

[2] OpenSSH: *http://www.openssh.org/*

[3] OpenVPN: *http://openvpn.sourceforge.net/*

[4] FreeS/WAN: *http://www.freeswan.org/*

[5] Patricia Jung: "Talking Privileges", Linux Magazine, Issue 30 March 2003, p64