VPN HOWTO

Table of Contents

Table of Contents

VPN HOWTO

Matthew D. Wilson, matthew@shinythings.com

v 1.0, Dec 1999

This HOWTO describes how to set up a Virtual Private Network with Linux.

1.Introduction

2.Theory

3.Server

4.Client

5.Implementation

6.Addenda

1.Introduction

1.1 Why I wrote this HOWTO

I work Real Networks, and we needed VPN service. This was my first real project, and I truly learned more about Linux with this than with any other task. I ended up using my experience with that project to write this document, to share with others what I learned, so that they can do ultra−nifty things with Linux too!

1.2 Acknowledgements and Thanks

I want to first and foremost thank my wife Julie, without her, I wouldn't be where I am today. I also want to thank Arpad Magosanyi, the author of the first VPN mini−howto and pty−redir, the utility that makes all of this possible. Jerry, Rod, Glen, Mark V., Mark W., and David, You guys rock! Thanks for all your help.

1.3 Format of this document

This document is broken down into 5 sections.

Section 1: Introduction

This section

Section 2: Theory

Basic VPN theory. What is a VPN, and how does it work. Read this if you are entirely new to VPN.

Section 3: Server

This section describes how a VPN server is set up.

Section 4: Client

This section describes how a VPN client is set up.

Section 5: Implementation

A step by step implementation of a sample VPN setup.

Section 6: Addenda

Other bits and pieces of info that you might find helpful.

1.4 Copyright and Disclaimer

The author assumes no responsibility for anything done with this document, nor does he make any warranty, implied or explicit. If you break it, it's not my fault. Remember, what you do here could make very large holes in the security model of your network. You've been warned.

1.5 Document History

The original VPN mini−HOWTO was written by Arpad Magosanyi in 1997. He has since allowed me to take up the document and extend it into a full HOWTO. All of this would not be possible without his original document. Thanks again Arpad. :)

Version 1.0 of this HOWTO was completed on December 10, 1999.

1.6 Related Documents

- Networking Overview HOWTO
- Networking HOWTO
- VPN−Masquerade HOWTO

2.Theory

2.1 What is a VPN?

VPN stands for Virtual Private Network. A VPN uses the Internet as it's transport mechanism, while maintaining the security of the data on the VPN.

But really, what IS a VPN?

Well, there are several answers to that question. It really depends on your network layout. The most common configuration is to have a single main internal network, with remote nodes using VPN to gain full access to the central net. The remote nodes are commonly remote offices or employees working from home. You can also link two smaller (or even larger!) networks to form an even larger single network.

So how does it work?

Put simply, to make a VPN, you create a secure tunnel between the two networks and route IP through it. If I've lost you already, you should read The Linux Networking Overview HOWTO to learn about networking with Linux.

Please bear with me, my ASCII art could use some work.

```
                                   \            \
                  --------         /            /       --------
Remote _______| Client |______\ Internet \_____| Server |______ Private
Network       | Router |       /            /      | Router |        Network
                  --------         \            \       --------
                                   /            /
```

```
                         Client Router
               ----------------------------------------------------
              |   /->    10.0.0.0/255.0.0.0   \                    |
  Remote      |  |-->  172.16.0.0/255.240.0.0  |--> Tunnel >---\   |
  Network >---|--|--> 192.168.0.0/255.255.0.0 /                 |--|----> Internet
 192.168.12.0 |  |                                              |  |
              |   \-----> 0.0.0.0/0.0.0.0 --> IP Masquerade >--/   |
               ----------------------------------------------------
```

```
                        Server Router
             ----------------------------------------------------
            |                   /->    10.0.0.0/255.0.0.0    \   |
            |   /--> Tunnel >--|-->  172.16.0.0/255.240.0.0   |--|----> Private
Internet >--|--|                \--> 192.168.0.0/255.255.0.0 /   |      Network
            |  |                                                 |     172.16.0.0/12
            |   \-----> 0.0.0.0/0.0.0.0 -----> /dev/null         |    192.168.0.0/16
             ----------------------------------------------------
```

The above diagram shows how the network might be set up. If you don't know what IP Masquerade is, you probably shouldn't be here. Go read the The Linux Networking Overview HOWTO and come back once you've figured it out.

The Client Router is a Linux box acting as the gateway/firewall for the remote network. As you can see, the remote network uses the local net 192.168.12.0. For the sake of a simple diagram, I let out the local routing information in the routers. The basic idea is to route traffic for all of the private networks (10.0.0.0, 172.16.0.0, and 192.168.0.0) through the tunnel. The setup shown here is one way. That is, while the remote network can see the private network, the private network cannot necessarily see the remote network. In order for that to happen, you must specify that the routes are bidirectional.

From the diagram you should also note that all of the traffic coming out of the client router appears to be from the client router, that is, it's all from one IP address. You could route real numbers from inside your network but that brings all sorts of security problems with it.

2.2 SSH and PPP

The system that I describe to implement VPN uses SSH and PPP. Basically I use ssh to create a tunnel connection, and then use pppd to run TCP/IP traffic though it. That's what makes up the tunnel.

The real trick to getting ssh and pppd to play well together is the utility written by Arpad Magosanyi that allows the redirection of standard in and standard out to a pseudo tty. This allows pppd to talk through ssh as if it were a serial line. On the server side, pppd is run as the users shell in the ssh session, completing the link. After that, all you need to do is the routing.

2.3 Alternative VPN Systems

There are of course other ways of setting up a VPN, here are a couple of other systems.

PPTP

PPTP is a Microsoft protocol for VPN. It is supported under Linux, but is known to have serious security issues. I do not describe how to use it here since it is covered by the Linux VPN Masquerade HOWTO.

IP Sec

IP Sec is a different set of protocols from SSH. I don't actually know all that much about it, so if someone wants to help me out with a description, I'd be most appreciative. Again, I do not describe how to use it here since it is covered by the Linux VPN Masquerade HOWTO.

CIPE

CIPE is a kernel level network encryption system that may be better suited to enterprise setups. You can find out more about it at the CIPE homepage. I plan on looking into it more, so maybe I'll have info about it here someday.

3.Server

This section tells you how to set up the server side of things, I figured that this should go first since without a server, your client is kind of useless.

3.1 Security – keeping people out

Security is very important for a VPN. That's why you're building one in the first place, isn't it? You need to keep a few things in mind while setting up your server.

Trim your daemons

Since this server is going to be on both sides of your firewall, and set up to forward traffic into your network, it's a good idea to secure the box as well as you possibly can. You can read up more on Linux security in the Linux Security HOWTO For my purposes, I've killed everything but sshd and a Roxen Web server. I use the web server to download a couple of files (my scripts, etc) for setting up new machines to access the VPN. I don't use an FTP server since it's harder to configure one to be secure than it is to just make a few files available with a web server. Plus, I only need to be able to download files. If you really want to run different servers on your gateway, you might want to think about restricting access to them to only those machines on your private network.

Don't allow passwords

Yes, it sounds kind of silly, but it got your attention, didn't it? No, you don't use passwords, you disable them completely. All authentication on this machine should be done via ssh's public key authentication system. This way, only those with keys can get in, and it's pretty much impossible to remember a binary key that's 530 characters long.

So how do you do that? It requires editing the /etc/passwd file. The second field contains either the password hash, or alternatively 'x' telling the authentication system to look in the /etc/shadow file. What you do is change that field to read '*' instead. This tells the authentication system that there is no password, and that none should be allowed.

Here's how a typical /etc/passwd file looks:

```
...
nobody:x:65534:100:nobody:/dev/null:
mwilson:x:1000:100:Matthew Wilson,,,:/home/mwilson:/bin/bash
joe:*:504:101:Joe Mode (home),,,:/home/vpn-users:/usr/sbin/pppd
bill:*:504:101:Bill Smith (home),,,:/home/vpn-users:/usr/sbin/pppd
frank:*:504:101:Frank Jones (home),,,:/home/vpn-users:/usr/sbin/pppd
...
```

Note that I've done more than just editing the second field. I'll say more about the other fields later on.

3.2 User Access – letting people in

User access is done via ssh's authentication scheme. As I've stated above, this is how users get access to the system, while maintaining a high level of security. If you're not familiar with ssh, check out http://www.ssh.org/ Note that I am using ssh version 1, not version 2. There is a big difference, notably that version 1 is free, and 2 isn't.

Configuring `sshd`

You'll need to configure sshd. The following options should be present. The idea is to disable password authentication and rhosts authentication. The following options should be present in your `/etc/sshd_config` file.

```
PermitRootLogin yes
IgnoreRhosts yes
StrictModes yes
QuietMode no
CheckMail no
IdleTimeout 3d
X11Forwarding no
PrintMotd no
KeepAlive yes
RhostsAuthentication no
RhostsRSAAuthentication no
RSAAuthentication yes
PasswordAuthentication no
PermitEmptyPasswords no
UseLogin no
```

3.3 Restricting Users

Now that you're keeping the bad people out, and only letting the good people in, you may need to make sure that the good people behave themselves. This is most easily done by not letting them do anything except run pppd. This may or may not be necessary. I restrict users because the system that I maintain is dedicated to VPN, users have no business doing anything else on it.

sudo or not sudo

There is this neat little program called sudo that allows the admin on a Unix system to grant certain users the ability to run certain programs as root. This is necessary in this case since pppd must be run as root. You'll need to use this method if you want to allow users shell access. Read up on how to setup and use sudo in the sudo man page. Using sudo is best on multi−use systems that typically host a small number of trusted users.

If you decide to not allow users to have shell access, then the best way to keep them from gaining it is to make their shell be pppd. This is done in the /etc/passwd file. You can see above that that is what I did for the last three users. The last field of the /etc/passwd file is the user's shell. You needn't do anything special to pppd in order to make it work. It gets executed as root when the user connects. This is certainly the simplest setup to be had, as well as the most secure. It's the ideal for large scale and corporate systems. I describe exactly what all needs to be done later in this document. You can jump ahead if you like.

3.4 Networking

Now that your users have access to the system, we need to make sure that they have access to the network. We do that by using the Linux kernel's firewalling rules and routing tables. Using the `route` and `ipfwadm` commands, we can set up the kernel to handle network traffic in the appropriate ways. For more info on `ipfwadm`, `ipchains` and `route` see the Linux Networking HOWTO.

The Kernel

In order for any of this to work, you must have your kernel configured correctly. If you don't know how to build your own kernel, then you should read the Kernel HOWTO. You'll need to make sure that the following kernel options are turned on in addition to basic networking. I use a 2.0.38 kernel in my system.

For 2.0 kernels:

- CONFIG_FIREWALL
- CONFIG_IP_FORWARD
- CONFIG_IP_FIREWALL
- CONFIG_IP_ROUTER
- CONFIG_IP_MASQUERADE (optional)
- CONFIG_IP_MASQUERADE_ICMP (optional)
- CONFIG_PPP

For 2.2 kernels:

- CONFIG_FIREWALL
- CONFIG_IP_ADVANCED_ROUTER
- CONFIG_IP_FIREWALL
- CONFIG_IP_ROUTER
- CONFIG_IP_MASQUERADE (optional)
- CONFIG_IP_MASQUERADE_ICMP (optional)
- CONFIG_PPP

Filter Rules

First, we write firewall filter rules that allow our users to access our internal nets, while restricting them from accessing the outside internet. If this sounds wierd, but think about it this way: they already have access to the internet, so why let them use the tunnel to access the net? It wastes both our bandwidth and processor.

The filter rules that we use depend upon which internal nets we use. But basically they say: "Allow traffic coming from our VPNs that is destined for our internal nets to go there." So how do we do that? As always, it depends. If you are running a 2.0 kernel, you use the tool called `ipfwadm`, if on the other hand you are using a 2.2 kernel, you use the utility called `ipchains`.

To set the rules with `ipfwadm`, run it with options similar to the following:

```
# /sbin/ipfwadm -F -f
# /sbin/ipfwadm -F -p deny
# /sbin/ipfwadm -F -a accept -S 192.168.13.0/24 -D 172.16.0.0/12
```

To set the rules with `ipchains`, run it with options similar to the following:

```
# /sbin/ipchains -F forward
# /sbin/ipchains -P forward DENY
# /sbin/ipchains -A forward -j ACCEPT -s 192.168.13.0/24 -d 172.16.0.0/12
```

For those using 2.2 kernels, please read this.

Routing

So, now our users are allowed to access our nets, now we need to tell the kernel where to send the packets. On my system, I have two ethernet cards, one is on the external network, while the other is on the internal network. This helps keep things secure, as outbound traffic is masqueraded by our gateway, and any incoming traffic is filtered and routed by our Cisco. For most setups, the routing should be simple.

What we do is route all traffic destined for the private networks out the internal interface, and all other traffic out the external interface. The specific routing commands depend on which internal nets you are using. Below is an example of what they might look like. These lines are of course in addition to your basic routes for your local nets. I also doubt that you are using all 3 groups of internal numbers.

```
Assuming that 172.16.254.254 is our internal gateway:

# /sbin/route add -net 10.0.0.0 netmask 255.0.0.0 gw 172.16.254.254 dev eth1
# /sbin/route add -net 172.16.0.0 netmask 255.240.0.0 gw 172.16.254.254 dev eth1
# /sbin/route add -net 192.168.0.0 netmask 255.255.0.0 gw 172.16.254.254 dev eth1
```

One additional note on routing. If you are using two way routing for say, a remote office, then you will need to do one more thing. You'll need to set up routes on the server that point back to the client. The easiest way to accomplish this is to run a cron job every minute that quietly sets back routes. It's not a big deal if the client isn't connected, `route` will just spit out an error (that you've conveniently sent to `/dev/null`.)

4.Client

Now we examine the client end. In practice, when used to allow access to a remote network, this box can easily serve as a Samba (Windows Networking) server, DHCP server, and even an internal web server. The important thing to remember is that this box should be as secure as possible, as it runs your whole remote network.

4.1 The Kernel

First things first, you must have ppp available in your kernel. If you are going to allow multiple machines to use the tunnel, then you need to have firewalling and forwarding available too. If the client is going to be a single machine, ppp is enough.

4.2 Bring up the link

The link is created by running `pppd` through a pseudo terminal that is created by `pty-redir` and connected to `ssh`. This is done with something similar to the following sequence of commands:

```
# /usr/sbin/pty-redir /usr/bin/ssh -t -e none -o 'Batchmode yes' -c blowfish -i /root/.ssh/identity.vpn -l joe > /tmp/vpn-device
# sleep 10

# /usr/sbin/pppd `cat /tmp/vpn-device`
# sleep 15

# /sbin/route add -net 172.16.0.0 gw vpn-internal.mycompany.com netmask 255.240.0.0
# /sbin/route add -net 192.168.0.0 gw vpn-internal.mycompany.com netmask 255.255.0.0
```

Simply, what this does is run ssh, redirecting it's input and output to pppd. The options passed to ssh configure it to run without escape characters (−e), using the blowfish crypto algorithm (−c), using the identity file specified (−i), in terminal mode (−t), with the options 'Batchmode yes' (−o). The sleep commands are used to space out the executions of the commands so that each can complete their startup before the next is run.

4.3 scripting

Of course you don't want to have to type those commands in every time that you want to get the tunnel running. I've written a set of bash scripts that keep the tunnel up and running. You can download the package from here. Just download that an uncompress it into /usr/local/vpn. Inside you'll find three files:

- vpnd: The script that controls the tunnel connection.
- check−vpnd: a script to be run by cron to check that vpnd is still up.
- pty−redir: a small executable needed to initialize the tunnel.

You'll need to edit the `vpnd` script to set things like the client's username and the server's names. You may also need to modify the starttunnel section of the script to specify which networks you are using. Below is a copy of the script for your reading enjoyment. You'll note that you could put the script in a different directory, you just need to change the VPN_DIR variable.

```
#! /bin/bash
#
# vpnd: Monitor the tunnel, bring it up and down as necessary
#

USERNAME=vpn-username
IDENTITY=/root/.ssh/identity.vpn

VPN_DIR=/usr/local/vpn
LOCK_DIR=/var/run
VPN_EXTERNAL=vpn.mycompany.com
VPN_INTERNAL=vpn-internal.mycompany.com
PTY_REDIR=${VPN_DIR}/pty-redir
SSH=${VPN_DIR}/${VPN_EXTERNAL}
PPPD=/usr/sbin/pppd
ROUTE=/sbin/route
CRYPTO=blowfish
PPP_OPTIONS="noipdefault ipcp-accept-local ipcp-accept-remote local noauth nocrtscts lock nodefaultroute"
ORIG_SSH=/usr/bin/ssh


starttunnel () {
   $PTY_REDIR $SSH -t -e none -o 'Batchmode yes' -c $CRYPTO -i $IDENTITY -l $USERNAME > /tmp/vpn-device
   sleep 15

   $PPPD `cat /tmp/vpn-device` $PPP_OPTIONS
   sleep 15

   # Add routes (modify these lines as necessary)
   /sbin/route add -net 10.0.0.0 gw $VPN_INTERNAL netmask 255.0.0.0
   /sbin/route add -net 172.16.0.0 gw $VPN_INTERNAL netmask 255.240.0.0
   /sbin/route add -net 192.168.0.0 gw $VPN_INTERNAL netmask 255.255.0.0
}

stoptunnel () {
   kill `ps ax | grep $SSH | grep -v grep | awk '{print $1}'`
}

resettunnel () {
   echo "reseting tunnel."
   date >> ${VPN_DIR}/restart.log
   eval stoptunnel
   sleep 5
   eval starttunnel
}

checktunnel () {
   ping -c 4 $VPN_EXTERNAL 2>/dev/null 1>/dev/null

   if [ $? -eq 0 ]; then
      ping -c 4 $VPN_INTERNAL 2>/dev/null 1>/dev/null
      if [ $? -ne 0 ]; then
         eval resettunnel
      fi
   fi
}

settraps () {
   trap "eval stoptunnel; exit 0" INT TERM
   trap "eval resettunnel" HUP
   trap "eval checktunnel" USR1
```

```
}

runchecks () {
   if [ -f ${LOCK_DIR}/tunnel.pid ]; then
      OLD_PID=`cat ${LOCK_DIR}/vpnd.pid`
      if [ -d /proc/${OLD_PID} ]; then
         echo "vpnd is already running on process ${OLD_PID}."
         exit 1
      else
         echo "removing stale pid file."
         rm -rf ${LOCK_DIR}/vpnd.pid
         echo $$ > ${LOCK_DIR}/vpnd.pid
         echo "checking tunnel state."
         eval checktunnel
      fi
   else
      echo $$ > ${LOCK_DIR}/vpnd.pid
      eval starttunnel
   fi
}

case $1 in
    check)  if [ -d /proc/`cat ${LOCK_DIR}/vpnd.pid` ]; then
               kill -USR1 `cat ${LOCK_DIR}/vpnd.pid`
               exit 0
            else
               echo "vpnd is not running."
               exit 1
            fi ;;

    reset)  if [ -d /proc/`cat ${LOCK_DIR}/vpnd.pid` ]; then
               kill -HUP `cat ${LOCK_DIR}/vpnd.pid`
               exit 0
            else
               echo "vpnd is not running."
               exit 1
            fi ;;

   --help | -h)
            echo "Usage: vpnd [ check | reset ]"
            echo "Options:"
            echo "     check    Sends running vpnd a USR1 signal, telling it to check"
            echo "              the tunnel state, and restart if neccesary."
            echo "     reset    Sends running vpnd a HUP signal, telling it to reset"
            echo "              it's tunnel connection." ;;
esac

ln -sf $ORIG_SSH $SSH
settraps
runchecks

while true; do
   i=0
   while [ $i -lt 600 ]; do
      i=((i+1))
      sleep 1
   done
   eval checktunnel
done
```

4.4 LRP − Linux Router Project

I actually run this setup on pentium 90's running the LRP distribution of Linux. LRP is a distribution of Linux that fits in, and boots off of a single floppy disk. You can learn more about it at http://www.linuxrouter.org/ You can download my LRP package for the VPN client from here. You will also need both the ppp and ssh packages from the LRP site.

5.Implementation

In this section, I explain step by step how to set up your VPN system. I'll start with the server, and then move on to the client. For the purposes of an example, I will invent a situation that would require a couple of different kinds of VPN set up.

5.1 Planning

Let's imagine that we have a company, called mycompany.com. At our head office, we are using the 192.168.0.0 reserved network, breaking the class B into 256 class C networks to allow routing. We have just set up two small remote offices, and want to add them to our network. We also want to allow employees who work from home to be able to use their DSL and cable modem connections instead of making them use dialup. To start, we need to plan things out a little.

I decide that I want to give each remote office a class C network range to allow them to expand as necessary. So, I reserve the 192.168.10.0 and 192.168.11.0 nets. I also decide that for home users, I've got enough numbers that I don't need to masquerade them on the VPN server side. Each client gets it's own internal IP. So, I need to reserve another class C for that, say 192.168.40.0. The only thing that I must now do is to add these ranges to my router. Let's imagine that our company owns a small Cisco (192.168.254.254) that handles all of the traffic through our OC1. Just set routes on the Cisco such that traffic headed to these reserved nets goes to our VPN server (192.168.40.254). I put the VPN server into the home user's net for reasons that should become clear later. We'll name the external interface of the server vpn.mycompany.com, and the internal vpn−internal.mycompany.com.

As for external numbers, we don't need to know them explicitly. You should have your own numbers, supplied by your ISP.

5.2 Gather the tools

we're going to need a few pieces of software. Get the following packages, and install them where specified.

For the Server:

- pppd (version 2.3 or greater)
- ssh (version 1.2.26 or better)

For the Client:

- pppd (same version as server)
- ssh
- pty−redir

5.3 Server: Build the kernel

To start, you'll probably need to rebuild your kernel for the server. You need to make sure that the following kernel options are turned on in addition to basic networking and everything else that you might need. If you've never built your own kernel before, read the Kernel HOWTO.

For 2.0 kernels:

- CONFIG_FIREWALL
- CONFIG_IP_FORWARD
- CONFIG_IP_FIREWALL
- CONFIG_IP_ROUTER
- CONFIG_PPP

For 2.2 kernels:

- CONFIG_FIREWALL
- CONFIG_IP_ADVANCED_ROUTER
- CONFIG_IP_FIREWALL
- CONFIG_IP_ROUTER
- CONFIG_PPP

5.4 Server: Configure Networking

If you are building a server that has only one network card, I suggest that you think about buying another, and rewiring your network. The best way to keep your network private is to keep it on it's own wires. So if you do have two network cards, you'll need to know how to configure both of them. We'll use eth0 for the external interface, and eth1 for the internal interface.

Configuring the interfaces

We first should configure the external interface of the server. You should already know how to do this, and probably already have it done. If you don't, then do so now. If you don't know how, go back and read the Networking HOWTO

Now we bring up the internal interface. According to the numbers that we've chosen, the internal interface of the server is 192.168.40.254. so we have to configure that interface.

For 2.0 kernels, use the following:

```
# /sbin/ifconfig eth1 192.168.40.254 netmask 255.255.255.0 broadcast 192.168.40.255
# /sbin/route add -net 192.168.40.0 netmask 255.255.255.0 dev eth1
```

For 2.2 kernels, use the following:

```
# /sbin/ifconfig eth1 192.168.40.254 netmask 255.255.255.0 broadcast 192.168.40.255
```

That get's our basic interfaces up. You can now talk to machines on both local networks that are attached to the server.

Setting routes

We can now talk to machines on our local nets, but we can't get to the rest of our internal network. That requires a few more lines of code. In order to reach the other machines on other subnets, we need have a route that tells traffic to go to the Cisco router. Here's that line:

```
# /sbin/route add -net 192.168.0.0 gw 192.168.254.254 netmask 255.255.0.0 dev eth1
```

That line tells the kernel that any traffic destined for the 192.168.0.0 network should go out eth1, and that it should be handed off to the Cisco. Traffic for our local net still gets where it is supposed to because the routing tables are ordered by the size of the netmask. If we were to have other internal nets in our network, we would have a line like the above for each net.

Making filter rules

Ok, so we can reach every machine that we could need to. Now we need to write the firewall filtering rules that allow or deny access through the VPN server.

To set the rules with `ipfwadm`, run it like so:

```
# /sbin/ipfwadm -F -f
# /sbin/ipfwadm -F -p deny
# /sbin/ipfwadm -F -a accept -S 192.168.40.0/24 -D 192.168.0.0/16
# /sbin/ipfwadm -F -a accept -b -S 192.168.10.0/24 -D 192.168.0.0/16
# /sbin/ipfwadm -F -a accept -b -S 192.168.11.0/24 -D 192.168.0.0/16
```

To set the rules with `ipchains`, run it like so:

```
# /sbin/ipchains -F forward
# /sbin/ipchains -P forward DENY
# /sbin/ipchains -A forward -j ACCEPT -s 192.168.40.0/24 -d 192.168.0.0/16
# /sbin/ipchains -A forward -j ACCEPT -b -s 192.168.10.0/24 -d 192.168.0.0/16
# /sbin/ipchains -A forward -j ACCEPT -b -s 192.168.11.0/24 -d 192.168.0.0/16
```

This tells the kernel to deny all traffic except for the traffic that is coming from the 192.168.40.0/24 network and destined for the 192.168.0.0/16 network. It also tells the kernel that traffic going between the 192.168.10.0/24 and 192.168.0.0/16 nets is allowed, and the same for the 192.168.11.0 net. These last two are bidirectional rules, this is important for getting the routing to work going both ways.

Routing

For the home users, everything will work just fine to here. However, for the remote offices, we need to do some routing. First of all, we need to tell the main router, or Cisco, that the remote offices are behind the VPN server. So specify routes on the Cisco that tell it to send traffic destined for the remote offices to the VPN server. Now that that is taken care of, we must tell the VPN server what to do with the traffic destined for the remote offices. To do this, we run the `route` command on the server. The only problem is that in order for the `route` command to work, the link must be up, and if it goes down, the route will be lost. The solution is to add the routes when the clients connects, or more simply, to run the route command frequently as it's not a problem to run it more than is necessary. So, create a script and add it to your crontab to be run every few minutes, in it, put the following:

```
/sbin/route add -net 192.168.11.0 gw 192.168.10.253 netmask 255.255.255.0
/sbin/route add -net 192.168.10.0 gw 192.168.11.253 netmask 255.255.255.0
```

5.5 Server: Configure `pppd`

Now we will configure pppd on the server to handle VPN connections. If you are already using this server to handle dialup users or even dialing out yourself, then you should note that these changes may affect those services. I go over how to avoid conflicts at the end of this section.

`/etc/ppp/`

This directory may contain a number of files. You probably already have a file called `options`. This file holds all of the global options for `pppd`. These options cannot be overridden by `pppd` on the command line.

`/etc/ppp/options`

Your `options` file should contain at least the following:

```
ipcp-accept-local
ipcp-accept-remote
proxyarp
noauth
```

The first two lines tell `pppd` to accept what the other end specifies for IP addresses. This is necessary when hooking up remote offices, but can be disabled if you are only connecting home users. It's ok to leave it on, as it does not prevent the server from assigning addresses, it only tells it that it's ok to accept what the client asks for.

The third line is very important. From the `pppd` man page:

```
proxyarp
       Add an entry to this system's ARP [Address  Resolu-
       tion  Protocol]  table  with  the IP address of the
       peer and the Ethernet address of this system.  This
       will  have  the effect of making the peer appear to
       other systems to be on the local ethernet.
```

This is important because if it is not done, local traffic will not be able to get back through the tunnel.

The last line is just as important. This tells `pppd` to allow connections without username and password. This is safe since authentication is already handled by `sshd`.

Avoiding conflicts

If you are handling other services with `pppd`, you should consider that the configurations for these other services may not be the same as what the VPN system needs. `pppd` is designed such that the options in the main options file `/etc/ppp/options` cannot be overridden by options specified at runtime. This is done for security reasons. In order to avoid conflict, determine which options cause the conflict, and move them from the main file into a separate options file that is loaded when the appropriate application of `pppd` is run.

5.6 Server: Configure `sshd`

The following is what my `/etc/sshd_config` file looks like. Yours should look the same or similar:

```
# This is the ssh server system wide configuration file.

Port 22
ListenAddress 0.0.0.0
HostKey /etc/ssh_host_key
RandomSeed /etc/ssh_random_seed
ServerKeyBits 768
LoginGraceTime 600
KeyRegenerationInterval 3600
PermitRootLogin yes
IgnoreRhosts yes
StrictModes yes
QuietMode no
FascistLogging yes
CheckMail no
IdleTimeout 3d
X11Forwarding no
PrintMotd no
KeepAlive yes
SyslogFacility DAEMON
RhostsAuthentication no
RhostsRSAAuthentication no
RSAAuthentication yes
PasswordAuthentication no
PermitEmptyPasswords no
UseLogin no
```

The important points to note are that password authentication is disabled as are all of the "r" services. I have also turned off mail checking and the message of the day as they can confuse `pppd` on the client side. I still allow root login, but as this can only be done with a key, it is adequately safe.

5.7 Server: Set up user accounts

Now we'll set up the user accounts.

Add `vpn−users` group

just run:

```
# /usr/sbin/groupadd vpn−users
```

Now cat the `/etc/group` file and look at the last line. It should be the entry for the vpn−users group. Note the third field. This is the group ID (GID). Write it down, as we'll need it in a minute. For this example, the GID is 101.

create the `vpn−users` home directory

We're going to use a single home directory for all of the users. So just run:

```
# mkdir /home/vpn−users
```

The `.ssh` directory

Now create the `.ssh` directory in the `vpn−users` home directory.

```
# mkdir /home/vpn−users/.ssh
```

Adding users

Now comes the fun part. We're going to edit the `/etc/passwd` file by hand. :) Normally you let the system handle this file, but for a wierd setup like this, it is easier to do it yourself. To start, let's open the `/etc/passwd` file and see what's in there. Here's an example of what you might find:

```
...
nobody:x:65534:100:nobody:/dev/null:
mwilson:x:1000:100:Matthew Wilson,,,:/home/mwilson:/bin/bash
joe:*:1020:101:Joe Mode (home),,,:/home/vpn−users:/usr/sbin/pppd
bill:*:1020:101:Bill Smith (home),,,:/home/vpn−users:/usr/sbin/pppd
frank:*:1020:101:Frank Jones (home),,,:/home/vpn−users:/usr/sbin/pppd
...
```

You'll find the first user on most any system. The second one is me. :) After that are a few made up vpn−users. The first field is the username, and the second is the password field. The third is user ID (UID) and the fourth is the group ID (GID). After that comes some info on who the people are in the fifth field. The sixth field is the user's home directory, and the last is their shell. As you can see, each field is separated by a colon. Look at the last three lines. The only difference between them is the username in the first field, and the user info in the fifth field. What we want to do is create lines like this for each user. Don't just use one user for all of the connections, you'll never be able to tell them apart if you do. So, copy the last line of this file and edit it so that it looks something like the above. Make sure that the second field has an asterisk (*). The second field should be unique to all the other IDs in the file. I used 1020. You should use a number above 1000, since those below are typically reserved for system use. The fourth field should be the group ID for vpn−users. I told you to write it down, now is the time that you need it. So put the group ID in there. Lastly, change the home directory to `/home/vpn−users`, and the shell to `/usr/sbin/pppd`. That's it. Now copy that line to make more users. Just edit the first the fifth fields and you're set.

5.8 Server: Administration

One of the advantages to using this system for user accounts is that you can take advantage of the UNIX user administration commands. Since each client is logged in as a user, you can use standard methods to get user statistics. The following are a few commands that I like to use to see what all is going on.

who

Prints the users currently logged in, as well as when they logged in, from where (name or IP), and on which port.

w

This command prints a more extensive listing of who is currently logged in. It also tells you uptime and load averages for the system. It also lists the user's current process (which should be −pppd for VPN clients) as well as idle time, and current CPU usage for all processes as well as the current process. Read the `w` man page for more info.

last [username]

This lists the login history for the specified user, or for all users if a username is not provided. It's most useful for finding out how well the tunnels are running as it prints the length of time that the user was logged in, or states that the user is still logged in. I should warn you that on a system that has been up a long time, this list can grow extremely long. Pipe is through `grep` or `head` to find out exactly what you want to know.

You can also control which users are allowed to connect by modifying the `/home/vpn−users/.ssh/authorized_keys` file. If you remove the user's public key line from this file, they won't be able to log in.

5.9 Client: Build the kernel

Now we move onto the client. First we must rebuild the kernel so that it can support all of the functions that we need. The minimum requirement is to have ppp in the kernel. After that, you will need forwarding, firewalling, and gatewaying only if you are going to allow other machines access to the tunnel. For this example, I will setup one of the remote office machines in my example layout. Add the following options to your kernel. Again, if you've never built a kernel before, read the Kernel HOWTO.

For 2.0 kernels:

- CONFIG_PPP
- CONFIG_FIREWALL
- CONFIG_IP_FORWARD
- CONFIG_IP_FIREWALL
- CONFIG_IP_ROUTER
- CONFIG_IP_MASQUERADE
- CONFIG_IP_MASQUERADE_ICMP

For 2.2 kernels:

- CONFIG_PPP
- CONFIG_FIREWALL
- CONFIG_IP_ADVANCED_ROUTER
- CONFIG_IP_FIREWALL
- CONFIG_IP_ROUTER
- CONFIG_IP_MASQUERADE
- CONFIG_IP_MASQUERADE_ICMP

5.10 Client: Configure Networking

Now we should setup the networking on our client box. Let's assume that we've configured the external network and that it works. Now we will configure the internal interface of the client to service our intranet.

Interface

We need to first bring up the internal network interface. To do this, add the following to your `/etc/rc.d/rc.inet1` (or equivalent) file:

For 2.0 Kernels:

```
/sbin/ifconfig eth1 192.168.10.253 broadcast 192.168.10.255 netmask 255.255.255.0
/sbin/route add -net 192.168.10.0 netmask 255.255.255.0 dev eth1
```

For 2.2 Kernels:

```
/sbin/ifconfig eth1 192.168.10.253 broadcast 192.168.10.255 netmask 255.255.255.0
```

Filter rules

For setting up the remote office, we will want to set up our filter rules that allow traffic to go both directions through the tunnel. Add the following lines to your `/etc/rc.d/rc.inet1` (or equivalent) file:

For 2.0 kernels:

```
/sbin/ipfwadm -F -f
```

```
/sbin/ipfwadm -F -p deny
/sbin/ipfwadm -F -a accept -b -S 192.168.10.0/24 -D 192.168.0.0/16
```

For 2.2 kernels:

```
/sbin/ipchains -F forward
/sbin/ipchains -P forward DENY
/sbin/ipchains -A forward -j ACCEPT -b -s 192.168.10.0/24 -d 192.168.0.0/16
```

You may have noticed that these lines look like what we have on the server. That's because they are the same. These rules just say where traffic is allowed to go, that is, between these two networks.

Routing

The only extra routes that are needed are created by the script that bring the tunnel up.

5.11 Client: Configure `pppd`

You may not need to edit the client's `/etc/ppp/options` file at all. You will if the "auth" option is present, or some of the other priveledged options. Try it, and if it fails, a black `/etc/ppp/options` will work. just keep adding the options from the old file to figure out which one broke it (if it's not obvious) and see if you can get around that. Maybe you don't need them at all. You probably don't if you don't use `pppd` for anything else.

5.12 Client: Configure `ssh`

As root on the client, run the following lines:

```
# mkdir /root/.ssh
# ssh-keygen -f /root/.ssh/identity.vpn -P ""
```

This will create two files, `identity.vpn` and `identity.vpn.pub` in the `.ssh` directory. The first is your private key, and should be kept such. *NEVER SEND THIS OVER THE NET* unless it is via an encrypted session. The second file is your public key, and you can send this anywhere you want, it only serves to allow you access to other systems, and cannot be used to get into your own. It is a text file with one line in it that is your actual key. At the end of the line is the comment field which you may change without fear of breaking the key. an example key looks something like this:

```
1024 35 1430723736674162619588314275167.......250872101150654839 root@vpn-client.mycompany.com
```

It's actually a lot longer than that, but it wouldn't fit on the page if I showed the whole thing. Copy your key into the `/home/vpn-users/.ssh/authorized_keys` file on the server. Make sure that there is only one key per line, and that each key is not broken onto multiple lines. You may alter the comment field all that you like in order to help you remember which line goes with which user. I highly recommend doing so.

5.13 Client: Bring up the connection

Now we'll try to actually make the connection to the VPN server. First we'll need to make a single connection the set up `ssh`'s known_hosts file. Run this:

```
# ssh vpn.mycompany.com
```

Answer "yes" when it asks you if you want to continue connecting. The server will tell you "permission denied", but that"s ok. It's important that you use the same name for the server that you are using in your connection scripts. Now run the following lines. You will obviously need to change most of the options to fit your setup.

```
# /usr/sbin/pty-redir /usr/bin/ssh -t -e none -o 'Batchmode yes' -c blowfish -i /root/.ssh/identity.vpn -l vpn-user vpn.mycompany.com > /tmp/vpn-device

        (now wait about 10 seconds)

# /usr/sbin/pppd `cat /tmp/vpn-device` 192.168.10.254:192.168.40.254
```

Note the IP addresses specified on the pppd line. The first is the address of the client end of the tunnel. The second is the address of the server end of the tunnel, which is set to the server's internal address. If all of that seemed to work, move on. If not, check that you have all of the options, and that they are spelled right. If something is still going wrong, check the Pitfalls section.

5.14 Client: Set the routes

Now set the route to send traffic through the tunnel. Just run this:

```
# /sbin/route add -net 192.168.0.0 gw vpn-internal.mycompany.com netmask 255.255.0.0
```

You should now be able to communicate with machines on the other side of the tunnel. Give it a try. Neat huh? If it doesn't work, try using `ping` and `traceroute` to figure out where your problem might be. If in fact it does work, move on to setting up scripts to do the work for you.

5.15 Client: Scripting

Use the vpnd script that I show vpn−scripthere. Only, you need to modify it a little. Make the following changes:

- Change the variables at the top to match your setup. Most should be just fine as they are, but you can change them should you need to.
- Line 27: add the local and remote IP addresses before $PPP_OPTIONS
- Line 31: Change this line, and the two after it to set routes for your internal nets.

Keeping it running

While bash scripts are generally stable, they have been known to fail. In order to make sure that the `vpnd` script keeps running, add an entry to the client's crontab that runs the `check−vpnd` script. I run mine every 5 minutes or so. If `vpnd` is indeed running, `check−vpnd` doesn't use much CPU.

6.Addenda

6.1 Pitfalls

Here are just a few of the snags that I've run into while using this system. I put them here so that you can hopefully avoid them. If you run into any new ones, please email them to me so that I can keep track, and help others avoid them.

read: I/O error

This error apparently comes from pppd. It's associated with mis−matched versions off pppd. If you get it, try upgrading both ends of the connection to the latest version of pppd. I've found that pppd version 2.2 has this problem, and use version 2.3.7 or 2.3.8 instead.

SIOCADDRT: Network is unreachable

This error is generated by `route`. I've seen it happen when the sleep time between `ssh` and `pppd` is not long enough. If you get this error, run `ifconfig`, you may see that there is no pppX interface. This means that `ssh` was not done authenticating before `pppd` was launched, and therefore `pppd` did not make the connection. just increase the delay, and your problems will be solved.

I wonder however if there might be some pppd option that will fix this problem.

IPv4 Forwarding and 2.2 kernels

In the new 2.2 kernel, you must specifically enable IP forwarding in the kernel at boot up. This with the following command:

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

Without this, the kernel will not forward any packets, and hence the server will not work, nor will any of the gatewaying clients.

Routing

it should go without saying, but be careful when you are routing real numbers that you don't route traffic destined for the VPN server's external address through the tunnel. It won't make it. (yes, this *is* from personal experience.)

6.2 Hardware and Software Requirements

Minimum Hardware Requirements

Believe it or not, this system has been run on a 486SX33 with 8 megabytes of RAM. It didn't run very well though, it had trouble handling heavy traffic.

It doesn't take much more to make it work though. This system does work very well on a Pentium 75 with 16 megs of RAM, using an LRP distribution running off of a floppy, with a 6 meg ramdisk, and 10 megs of main memory. I've tested this setup by running a 700kbit RealVideo stream through it for over an hour.

I now typically run it on Pentium 90's, as their PCI clocking plays nicer with cheap 100Mbit Ethernet cards.

Software Requirements

This system works with both the 2.0 and 2.2 kernels. The script to keep the tunnel up requires a reasonably modern bash. I have however noticed that certain distribution's versions of bash don't play too well with the script.

Also, if someone could help me refine my scripts (or even write an executable?) that would helps things a lot. I'm not sure why, but even my own bash doesn't follow the rules and doesn't seem to interpret signals correctly. If you do make any improvements, please email me at