Contents

7. Other Security resources

1. Disclaimer:
==========

Total security is unattainable in any practical sense. System security is always a fine balance between the users of the system, the admins, and the people who want to break in.

Many of the steps detailed here will be inappropriate for your system. I would suggest a "pick and choose" approach - read and understand the steps here, and then implement what will work for you.

Above all, please check out the Other Security Resources section. Your machines are your own responsibility - follow this HOWTO, but never assume that because you have, your machine is unhackable.

2. Feedback:
==========

Yes please!

I'd particularly like to hear of people's experiences implementing the steps here on different systems, in different roles, with different versions of IRIX.

Also any problems, questions, or things you feel should be updated or changed.

Email **tom@siliconbunny.com**

3. Test kit:
========

Everything in this HOWTO was written and tested on an Origin 200 running IRIX 6.5.19.

It is assumed that you are running *at least* that version of IRIX, and that the machine you are trying to secure is a Internet-connected server.

Many things here might lock down a workstation or multi-user development server to the point of being unusable. Read, look at your system, and modify things to suit your environment.

Please bear in mind that, as with all operating systems, the older the version you use, the more insecure it will be.

Hence following this document to the letter on an IRIX 6.2 system may still leave gaping security holes.

4. Overview:
==========

Many people loudly claim that IRIX is insecure. Naturally SGI will be focussed on making things work out of the box for workstation and HPC users. However, certainly in recent updates, a default IRIX install is pretty good. Certainly a default install of 6.5.19 is more secure in many areas than a similarly installed Solaris 9 system.

I believe some of this attitude is down to the relatively smaller installed base of IRIX systems, leading to a general lack of familiarity with production IRIX systems. As well, unless it's your job to secure systems, many sysadmins and hobbyists are unfamiliar with computing security in general.

It's important to remember that, with default OS installs, it's all swings and roundabouts, though - there are more or less secure aspects to any operating system.

What this HOWTO will present is a range of steps you can take to make your systems more secure. Consider it to be a best practices guide, as opposed to hard and fast rules that will guarantee a secure system. As with most things, security as a constantly changing and evolving area - a fix that today could secure your system, could tomorrow be useless against a new form of attack.

At the very least, reading this should increase your knowledge of IRIX, and make you more aware of the security aspect of administration.

5. Securing IRIX - the steps

==========

5.1 A note on umasks
==========

Let's have a quick run down on file permissions. A normal file will have permissions like this:

-rw-r--r-- 1 root sys 19 12 Sep 2003 /etc/motd

The first field is a special one, and defines the type of file - normal file, directory, block device file, character device file, etc.

We then have 3 groups of 3 fields, which set permissions for the owner, the group, and everyone else.

So, for */etc/motd*, we can see that root can read and write to the file, members of the sys group can read the file, and everyone else can read the file.

We set the permissions using chmod. If we wanted to set the above owner, group, and everyone permissions, we would use:

chmod o+rw,g+r,a+r /etc/motd

This is a bit clumsy, so there's an neater way of handling this. Each of those permissions fields can be expressed as octal number.

r would be *4*
w would be *2*
x would be *1*

Going back to */etc/motd*, we can set the permissions using our numeric values instead:

chmod 644 /etc/motd

You'll notice that the maximum values for each set of permissions is *7* (4 + 2 + 1) - which equates to full permissions of *rwx*.

When you create a file, any UNIX will automatically mask some bits, to giving you a default file creation mask. We can control this with the umask command.

umask works in reverse - the bits you specify are the ones which are masked.

For example:

umask 003 - you can create files mode 774 - *rwxrwxr--*
umask 077 - you create files mode 700 - *rwx------*
umask 022 - you create files mode 755 - *rwxr-xr-x*

From this you can see that umasks are incredibly important - the last thing we want is users create world-writeable files and directories all over our filesystems.

5.2 Setting correct umasks
==========

There are several places we can set umasks - some daemons have their own methods, and we can set them system wide for login sessions.

However, to make doubly sure no-one bypasses this, and that we have sane umasks whatever the run level, you can create startup scripts which will do this for you.

Create *umask.sh* in */etc/init.d*, with the following contents

```
#!/sbin/sh
echo "Setting umask"
umask 022
exit 0
```

Give it a decent permissions and restrict the ownership:

```
chown root:sys /etc/init.d/umask.sh
chmod 744 /etc/init.d/umask.sh
```

Now we've got a script to set the umask, we need to make sure that it is executed before anything else, at each runlevel.

The simplest way to do this is a quick for loop:

```
for DIR in /etc/rc?.d
do
    ln -s /etc/init.d/umask.sh /etc/$DIR/S00umask.sh
done
```

We'll cover startup and shutdown scripts in more detail later on, but for now this will ensure we're setting a decent umask at each run level.

5.3 /etc/default/login

This file describes the default behaviour of the login process. What we want to do here is secure things on a system wide basis for all users who login to the machine.

Edit the file, and make the following changes:

- uncomment the CONSOLE line, to allow root logins only on the system console
- set PASSREQ to YES - we want to force all accounts to have passwords when logging in
- set MANDPASS to YES - we want to force all accounts to have passwords
- change UMASK to 022
- set TIMEOUT to 15 - this sets how long a logged in user is idle before they are disconnected. You can be more or less aggressive here. Just remember that this will apply to *your* login sessions as well!
- uncomment SLEEPTIME and set to 15 seconds - this will slow down dictionary attacks
- uncomment DISABLETIME and set to 60 seconds - this to will slow down dictionary attacks
- uncomment MAXTRYS - it should already be set to 3
- uncomment LOGFAILURES - we want to know when someone is unable to correctly login
- set LOCKOUT to 6 - this is a decent amount of attempts to allow someone to remember their password, but stops repeated dictionary attacks

5.4 /etc/default/su

This file describes the default behaviour of the *su* command.

Edit the file, and make the following changes:

- uncomment the CONSOLE line, so that we get console logging when someone *su*'s to root

5.5 /etc/inetd.conf

inetd.conf controls the behaviour of inetd, the 'super server'. When you connect to a port, inetd will spawn a server daemon for you to service your session. You can best see this behaviour with things like telnet and ftp.

This is a personal preference, and will definitely break workstations, but I would highly recommend disabling *everything* in inetd. None of it is relevant or should be running for an Internet server.

The only exception is a line I add in to support IMAP/S, which is not in there by default. If you have nothing extra that needs to be handled by inetd, comment out everything in *inetd.conf*. To stop *inetd* from starting at all, rename */usr/etc/inetd* to something else - then */etc/init.d/network* won't try and start it.

5.6 chkconfig

chkconfig is an excellent and handy way IRIX uses to control services. You may be familiar with a similar setup on RedHat Linux - I'm sure there are other systems out there that have taken this feature from IRIX.

Typing *chkconfig* on it's own will print out a nice long list of all the services you can turn on or off. Note that not all of these will map to running daemons or startup/shutdown scripts.

Type *chkconfig | grep on* to see a list of enabled services.

I would recommend disabling everything apart from the following:

```
cleanpowerdown
ipaliases
```

mediad
network
nsd
privileges
savecores

If you don't have a CD-ROM drive, or you will not be using any removable media, you can kill off mediad as well.

See later sections for a discussion of Sendmail.

5.7 Disabling user accounts

SGI provide lots of nice accounts for showing off IRIX, workstation capabilities, or allowing an easy initial GUI configuration of the machine.

Using the command *passwd -l <username>* disable the following accounts:

EZsetup
guest
demos
OutOfBox
lp
nuccp

On top of that, look at */etc/passwd*. You will notice that all locked accounts will have either a ***** or ***LK*** in their password fields.

Edit */etc/passwd*, and change the shell on all the locked accounts to be */bin/false*. This provides an extra layer of assurance that no-one will be able to use them.

5.8 Enabling shadow passwords

Normally, a users password is encrypted using *crypt*, and the resulting string written into the password field of */etc/passwd*. This is a bad idea - */etc/passwd* by it's nature has to be world readable, and as such it opens up a whole world of password-guessing pain.

The answer is Shadow Passwords. Shadow passwords place a ***** in the password field of */etc/passwd*. Instead, a new file, */etc/shadow*, which is readable only by root, is used. This still stores the encrypted password, but since only root can read this file, things are a bit more secure.

To enable shadow passwords, run the command */sbin/pwconv*.

5.9 Installing /etc/default/passwd

As with */etc/default/login*, */etc/default/passwd* controls the system wide behaviour of user passwords. This doesn't initially exist - what we need to do is create it, and then put in place some decent password controls.

Create */etc/default/passwd* with the following contents:

```
# Set the maximum time before a password change must be forced to 8 weeks
MAXWEEKS = 8
# Set the minimum time a password can exist before a change can be forced
MINWEEKS = 1
# Set the default password length to the maximum of 8 characters
# No passwords can be shorter than this, and any characters beyond 8 in a password will be ignored
PASSLENGTH = 8
# How long before a forced password change should we remind the user they need to change it?
WARNWEEKS = 1
```

Note that, when a password is expired, the user will be forced to change it on login. Until they change the password, they will not be able to log in to the machine.

```
chown root:sys /etc/default/passwd
chmod 644 /etc/default/passwd
```

5.10 syslog
==========

We want to make sure that all login failures are trapped within syslog. More than that, we want to make sure these failures are saved to a seperate file with strict permissions, so that only the root user can get access to it.

To this end, we need to edit */etc/syslog.conf*, and add the following line:

```
# Trap login failures
auth.notice       /var/adm/login.log
```

Note that *syslog.conf* uses <tabs> between entries, not spaces.

Once we've done this, we need to create the file, set the ownership and permissions, and then restart syslog.

Use the following commands:

```
touch /var/adm/login.log
chown root:sys /var/adm/login.log
chmod 600 /var/adm/login.log
```

5.11 Removing unused startup scripts
==========

You might wonder - if you've turned off something with *chkconfig*, why disable the startup script for it? Remember, startup scripts and *chkconfig* do not necessarily go hand in hand - you can have the machine starting daemons which *chkconfig* doesn't know about, and vice versa.

Obviously, we can disable the startup and shutdown scripts for everything we've disabled with *chkconfig*. We can then prune down things until we only have running what we need.

It's important to remember that, unless you're dealing with a particularly naff software vendor, all the scripts in the */etc/rc?.d* directories should be hard or soft links to scripts in */etc/init,d*. These scripts will all take the same arguments - at the very least, a *start* or *stop* (and what that does should be obvious). Some scripts may also take a *restart* or *reload* argument as well, depending on what the application they control does.

The run levels where the major action takes place are 0, S, 2 and 3. We need to look through the appropriate */etc/rc?.d* directories and disable what we don't need.

When changing run level, SVR4 init will execute scripts in the following order:

- anything starting with a K is executed with an argument of stop, starting at 00 and working up through the numbers until all scripts have been executed
- anything starting with an S is executed with an argument of start, starting at 00 and working up through the numbers until all scripts have been executed

To this end, disabling a script is as simple as renaming it to start with k or s instead of K or S. Alternatively you may want to delete the scripts from the */etc/rc?.d* directories. Although you can always re-link them from the scripts in */etc/init.d*, it can be difficult to remember the exact order they were all executed in.

I play it safe by renaming the files, and then moving them to a directory under */etc/init.d*. This is then set mode 500 and owned by root:sys. By duplicating the directory structure, you can easily see which scripts went where, making it a simple task to re-enable a service.

Obviously if you are doing this on a workstation or server on a private LAN, you may want to keep NFS around - and indeed many other services.

Once again - look at your needs and change things accordingly.

5.12 Securing directory permissions
==========

By default, we have a few open directories floating about, which can be secured. Specifically, because we don't want people arbitrarily looking at people's crontabs, or changing them, we need to tighten up */var/cron*.

Do this with the following command:

```
chmod 700 /var/cron
```

In older versions of IRIX, you should check the following as well:

chmod -fR g-w /etc
chmod g-w /var
chmod g-w /var/spool

5.13 Deny FTP user access
==========

We don't want people using FTP. There are better ways to get files to and from a machine.

To this end, we want to stop people FTPing to the machine. Create the file */etc/ftpusers*, and add all the users from */etc/passwd* to it.

Use the following commands for this:

```
for USER in `cat /etc/passwd`
do
        awk '{print $1}' >> /etc/ftpusers
done
```

We need to make sure the file is secured, so only root can updated it, but that everyone can read it.

Use the following commands for this:

chmod 644 /etc/ftpusers
chown root:sys /etc/ftpusers

If you still feel you need to allow FTP access - stop. Think. If you're sharing out files, why not use rsync over ssh to setup an rsync service? More secure, it can be lighter on the bandwidth, and you'll be helping to make the world a better place.

5.14 Limiting shell access
==========

We also want to limit what shells people can use on the machine. This becomes very much a matter of sysadmin taste - I know what my users commonly use, and so strip out the 'odd' shells, or ones I consider to be hopelessly broken.

The end game here is to stop someone creating their own shell, and using that to login. Note that */etc/shells* places to restriction on what users can do once they're logged in - if someone enjoys the pain so much that they have to use *csh*, then when they've logged in via Korn they can instantly start *csh*.

The aim here is to limit and control how people are connecting to your machine.

Create the file */etc/shells*, and populate it with the full paths to the shells you want your users to use.

An example would be:

/bin/ksh
/sbin/sh
/usr/freeware/bin/bash

We need to make sure the file is secured, so only root can updated it, but that everyone can read it.

Use the following commands for this:

chmod 644 /etc/shells
chown root:sys /etc/shells

5.15 /etc/issue and /etc/motd
==========

When people log in, we want to remind them that the system is secured and monitored. There are varying opinions about the legal benefits of such banners, should a hack attempt come to court, but it's been shown in trials that "Welcome to my machine!" as a login banner is hardly helpful to the prosecution's case.

I would suggest putting the following message in */etc/issue* and */etc/motd*:

Authorised use only. This is a restricted access system. All usage is monitored.

5.16 Setting Sendmail banners

Sendmail can be quite chatty about what version it is when you connect, and this sort of promiscuous behaviour is to be discouraged. To this end, edit /etc/sendmail.cf, and look for the line that starts with *O SmtpGreetingMessage*

Edit this and change it to:

O SmtpGreetingMessage = Mail Server Ready\g

5.17 Better yet, ditch Sendmail

MTAs are a very personal thing. However, even the most die hard Sendmail fans admit that it's a bit big, and a bit complex. The complexity of the code is amply demonstrated by the frequency of security updates to Sendmail.

Personally, I'm not a big fan - the configuration is complex and arcane, and there are faster, more secure MTAs out there.

I would highly recommend disposing of Sendmail completely, and replacing it with a newer MTA. My own favourite is Exim, which is a lot faster, and has some nifty features like TLS encryption of SMTP message bodies.

If you find an MTA's configuration files are easy to understand and modify, then it's a lot less likely that you, as a sysadmin, are going to make a mistake in the config, and open your machine up to relaying or worse.

5.18 Setting ftpd options

ftpd will use */etc/issue* to display our nice restrictive banner on login. On top of that, we want to enable some extra features that will make it slightly more secure and manageable.

Note that, if you've been following previous steps, *ftpd* will be completely disabled on your machine. This is a worthy goal, but a lot of people either must use it, or insist on still using it.

So, edit */etc/inetd.conf*, and look for the f*tpd* line. It will look like this:

ftp stream tcp nowait root /usr/etc/ftpd ftpd -l

We want to add some extra options to ftpd itself, so we would make the following changes:

ftp stream tcp nowait root /usr/etc/ftpd ftpd -l -u 077 -S -p

These options give us:

-ll - this enables logging of succesful and unsuccessful logins, as well as all get and put commands, and the number of bytes transferred
-u 077 - this sets a nicely restrictive umask for all ftp sessions
-S - this enables extra security settings - like making soft links appear as ordinary files, so as not to show directory names which would not normally be visible to anonymous users
-p - this compares the IP addresses of the control and data connections. Data transfer will only be allowed if they match - this prevents some spoofing and relaying attacks, but will also break any proxy FTP connections

Send inetd a *kill -HUP* and new ftp sessions will start to use these settings.

5.19 A note about cron and scheduled jobs

Only a select few users should be running cronjobs, or scheduling job execution. Quite apart from the problem of your machine being thrashed at odd hours because someone thought their badly written script should execute at 3am, there's the problem of an unfriendly user attempting bad things on the machine when you're not around to check things. (But that should never be a concern, because you're routinely checking your system logs, aren't you?)

Looking more closely at the cron jobs on a freshly installed IRIX system, we can see that the only user who needs to actually run a cron job is root.

Now, you probably will have some sort of backup account, maybe some web server jobs which execute overnight - lots of customisations. If you blindly follow the instructions below, you will break all of that - so be aware of what crontabs you have on the system, and make modifications accordingly.

5.20 Setup cron.allow and at.allow
==========

I'm just going to allow root to schedule jobs or run a crontab. To this end, we need to create the file */etc/cron.d/cron.allow* with the following contents:

root

Obviously, if you want other people to run crontabs, add them in.

Then secure the file:

chmod 644 /etc/cron.d/cron.allow
chown root:sys /etc/cron.d/cron.allow

Copy *cron.allow* to *at.allow*, to apply the same restrictions to scheduling jobs with the at command.

5.21 Setup cron.deny and at.deny
==========

As *cron.allow* and *at.allow* let people run crontabs or schedule jobs, so *cron.deny* and *at.deny* define who can't do that. Basically you want to create these files, and populate them with all the other users from */etc/passwd* who aren't in the allow files.

Once you've done this, set the ownership and permissions:

chmod 644 /etc/cron.d/cron.deny /etc/cron,d/at,deny
chown root:sys /etc/cron.d/cron.deny /etc/cdron.d/at.deny

5.22 Find rhosts and hosts.equiv files (and how to keep on checking)
==========

rhosts files are evil. *hosts.equiv* files are evil. The whole raft of r commands (*rsh*, *rlogin* etc.) are incredibly insecure and nasty - they must be purged with prejudice.

No matter how hard you try, though, there's always going to be someone who want to cling to the 60s security model of "Sharing is Fun!".

To get by this, not only do you want to do an initial sweep (and deletion) of *rhosts* and *hosts.equiv*, but you also want to script this into root's crontab, and run it regularly - at least once a day.

To find these nuggets of evil, we can use the following commands:

find / ! -local -prune -o \(-name .rhosts -o -name hosts.equiv \) -print

This will go away and print out a nice list of the offending files. Before you delete them, make a note of who created them, and then give your users a rap on the knuckles.

5.23 Finding potentially Bad files and directories (and how to keep on checking)
==========

The final thing we want to do is look for Bad files and directories. By this I mean files which have elevated permissions, insecure permissions, or are owned by nobody. Sometimes there is a good reason for this - very often it's a mistake, or laziness on the part of developers.

Again, you should script this up and run it regularly from root's crontab - at least once a day. Once you've got to know your IRIX system, and are familiar with which files are Bad but can be ignored, you can remove these from your output.

The find commands we'll use are:

Check for SGID files:
find / ! -local -prune -o \(! type l -perm -2000 \) -print

Check for SUID files:
find / ! -local -prune -o \(! type l -perm -4000 \) -print

Check for unowned files:

find / ! -local -prune -o \(-nouser -o -nogroup \) -print

Check for world-writeable files:
find / ! -local -prune -o \(\(-type f -o -type d \) -a -perm -0002 \) -print

Examine the output from all of these, and check to see whether the files and directories it traps are OK, or if they need to be secured. Once you've run them a few times you should have an idea of which files and directories are alright. Stick those in a seperate file, and then use that to filter the output from the find commands.

5.24 Tools you should install and use

SGI's Freeware site has tardist packages of common free software packages. They are usually older versions, but some security is better than none - if you can't compile or build your own software, consider grabbing it from here.

Nekochan.net has more up to date versions of packages to download, and if you want the latest patched versions, get your tardists from here.

SGI's Freeware packages: *http://freeware.sgi.com*

Nekochan: *http://www.nekochan.net*

SSH is a secure drop in replacement for telnet, ftp, and the hated r commands. There is a commercial version, with support, from SSH Communications, and an Open Source version from the OpenBSD team, called OpenSSH.

Any Internet connected machine should have SSH in place as a matter of course, and you should try and implement it everywhere just to get your users comfortable with the idea.

OpenSSH: *http:.//www.openssh.com*

SSH Communications: *http://www.ssh.com*

tcpwrappers provides another layer of security by restricting which hosts can access a particular service. If you're going to be keeping inetd, and running services from it, you should put tcpwrappers in place to protect them.

The latest version of tcpwrappers can be downloaded from *ftp://ftp.porcupine.org/pub/security/tcp_wrappers_7.6.tar.gz*

Above and beyond this, you should make sure that any application software you have installed - Apache, BIND, Postgres, whatever - is the latest version. Keep up to date with the supplier's security updates, and implement updates as they are released.

Remember - you're taking all this effort because you want a secure system. If a security patch for an application comes out, you're running an insecure system. So why bother going to the effort to secure your machine in the first place?

I'd like to re-iterate the point I made earlier - security is a moving target, and you need to keep up with it. It's constant work to beat the bad guys.

5.25 Configuring tcpwrappers

Grab and install the latest version. Once it's in place, we need to setup a few things to take advantage of it.

When using tcpwrappers, it's a good idea to log all denied requests. We know who we allowed to access something - who are these guys trying to get to stuff we explicitly disallowed? Maybe it's a confused user. Maybe it's a zombie PC on ADSL, begging for a nastygram to the user's ISP. Whatever the cause, we need to log, then investigate.

Create the log file and set the permissions with the following commands:

touch /var/adm/tcpd.log
chown root;sys /var/adm/tcpd.log
chmod 600 /var/adm/tcpd.log

Now we need to create the two files that allow and deny access. The format of these is similar to the *cron.allow* and *cron.deny* files - only this time we're specifying hosts, not users.

Create */etc/hosts.allow* with the following content:

 # Allow access to all services from your own machines
ALL : localhost
ALL : .<yourdomain.com>

 # Allow access from a trusted user machine
ALL : <truster_user_machine>

Obviously, your_domain is your local domain, and truster_user_machine will be the hostname of a user's machine.

Now create */etc/hosts.deny* with the following content:

 # Explicitly deny everyone who is not in hosts.allow
ALL : ALL : spawn echo "Access attempt from host %h address %a to %d at `date`" >> /var/adm/tcpd.log

If you wanted, you could change the above so that the echo piped out to tee, and redirect to both the log file and mailx. If you get a lot of denied connection requests, you may not want an email each time one gets blocked.

Once you've created those two files, use the following commands to set permissons:

chown root:sys /etc/hosts.allow /etc/hosts.deny
chmod 440 /etc/hosts.allow /etc/hosts.deny

Now that we've got tcpwrappers in place, we need to start using it. Using ftpd as an example, here's how we would implement tcpwrappers in */etc/inetd.conf*

Before:

ftp stream tcp nowait root /usr/etc/ftpd ftpd -l -u 077 -S -p

After:

ftp stream tcp nowait root <path_to_tcpd> ftpd -l -u 077 -S -p

After this, send *inetd* a *kill -HUP* to make the changes take effect.

5.26 About chroot jails
==========

chroot is a command (and system call) to change the root for a command. By using *chroot*, you can create a virtual space for a process. This stops an attacker from utilising a flaw in an application to get elevated privileges - if they did exploit the process, they would be stuck in the *chroot* directory.

There are ways to break out of *chroot* jails, but combined with the other security measures, using *chroot* jails will prevent all but the most determined attackers.

5.27 Implementing chroot jails for FTP users
==========

If you absolutely must have to provide FTP services, you can provide an extra level of protection by implementing chroot jails for your ftp users.

At a minimum, you should do the following to secure FTP:
- create seperate accounts for shell users and FTP users
- edit */etc/passwd* and make the shell for all FTP users */bin/false*
- make sure */bin/false* is added to the end of */etc/shells*

Once you've taken these steps, you're ready to implement chroot jails for FTP.

The instructions below are for anonymous FTP (user account in */etc/passwd* is *ftp*). If setting up a chroot jail for a named FTP-only account, replace *~ftp* below with the user's home directory.

~ftp
Make the home directory owned by ``bin'' and unwritable by anyone (mode 555):

chown bin ~ftp
chmod 555 ~ftp

~ftp/bin
Make this directory owned by root and unwritable by anyone (mode 555). *ls* needs to be copied here, and should be made execute only (mode 111).

mkdir ~ftp/bin
chown root ~ftp/bin
chmod 555 ~ftp/bin
cp /bin/ls ~ftp/bin
chmod 111 ~ftp/bin/ls

~ftp/etc

Make this directory owned by root and unwritable by anyone (mode 555). The files *passwd* and *group* must be present for the ls command to be able to produce owner names rather than numbers. This should not be a copy of the real file in */etc*, and in particular, it should contain no encrypted passwords from the real */etc/passwd* or */etc/group*. You should limit the accounts listed to the bare minimum. These files should be mode 444.

mkdir ~ftp/etc
chmod 444 ~ftp/etc/passwd ~ftp/etc/group

~ftp/lib32

Make this directory own by root and unwritable by anyone (mode 555). In order for *ls* to run, the files */lib32/rld* and */lib32/libc.so.1* must be copied into *lib32* Both *rld* and *libc.so.1*should be readable and executable by everyone, (mode 555).

mkdir ~ftp/lib32
cp /lib32/rld ~ftp/lib32
cp /lib32/libc.so.1 ~ftp/lib32
*chown root ~ftp/lib32/**
*chmod 555 ~ftp/lib32/**

~ftp/dev

Make this directory owned by root and unwritable by anyone (mode 555). *rld* uses */dev/zero*, so use *mknod* to make a copy of */dev/zero* in *~ftp/dev* with the same major and minor device numbers. Make */dev/zero* read-only (mode 444).

mkdir ~ftp/dev
mknod ~ftp/dev/zero c 37 0
chmod 444 ~ftp/dev/zero

~ftp/pub

Make this directory owned by the user - eg. ftp. If local users and remote anonymous users are to be allowed to write in this directory, change the directory's mode to 777.
If write accesses are to be denied, change the directory's mode to 555.

mkdir ~ftp/pub
chown ftp ~ftp/pub
chmod 777 ~ftp/pub

Now that you've got all the directory structure setup, the last thing to do is edit */etc/passwd* and change the user's shell, by adding * to the front of it.

For example, for the FTP user, we would have:

ftp::5000:5000:Anonymous FTP user:/var/ftp:/bin/false*

and this would then be changed to:

ftp::5000:5000:Anonymous FTP user:/var/ftp:*/bin/false*

5.28 Using chroot jails for applications

The joy of chroot can be spread to other areas, notably the applications you run. I'm going to detail setting up a chroot jail for BIND, but you can also apply this to SSH, Apache, whatever. The usual tradeoff applies - how much time and effort are you willing to expend to secure your system, versus how likely is it that a given service will be attacked?

Personally, I would consider setting up chroot jails for FTP and BIND at least.

In this example we'll be using the lastest version of BIND - 9.2.3.

Let's assume you have downloaded the BIND 9.2.3 tar, and uncompressed it in */usr/local/src/bind-9.2.3*.

First we need to create a group and user:

Edit */etc/group* and add the following line:
named::53:*

Edit */etc/passwd* and add the following line:
named::53:53:named daemon account:/dev/null:/bin/false*

Then configure and build BIND:

```
cd /usr/local/src/bind-9.2.3
./configure --prefix=/var/named
make
make install
```

Then finish setting up the directories and support files for the chroot jail:

```
cd /var/named
mkdir -p dev var etc var/named/var/run var/named/var/log var/named/etc var/dns lib32
cp /etc/syslog.conf /etc/netconfig /etc/nsswitch.conf /etc/resolv.conf /etc/TIMEZONE etc
chown named var/named/var/run
```

We need to find out what libraries the BIND executables need, and then copy them into place:

```
cd /var/named
for LIB in `ldd bin/* sbin/* | awk '{print $3}'`
do
        cp $LIB /var/named/lib32
done

cp /lib32/rld /var/named/lib32
cp /lib32/libc.so.1 /var/named/lib32
```

Obviously BIND will required some device files, so we need to make sure those exist as well:

```
cd /var/named/dev
mknod tcp c 10 18
mknod udp c 10 19
mknod log c 10 80
mknod null c 1 2
mknod zero c 37 0
mknod conslog c 10 80
mknod syscon c 58 0
chmod 666 *
chmod 644 tcp
chmod 622 syscon
```

Now the final stage is to copy your *named.conf* to */var/named/var/named/etc/named.conf*, and touch any required logfiles under */var/named/var/log.*

Note that what we have recreated is a subset on a normal filesystem under */var/named* - only implementing the parts that named will need. This is the fundamental idea behind chroot jails.

Finally we can start BIND.

```
/var/named/sbin/named -u named -t /var/named
```

Make sure to modify your BIND startup/shutdown script to execute BIND as a non-privileged user within the chroot jail.

6. What SGI provides for you
==========

SGI will provide security patches for free. Their online support offering is called Supportfolio Online.

You'll need a login to get access, but you don't need a support contract - SGI will let you create a free Supportfolio account that will let you have access to most things - most notably patches.

To get an account, go to *http://www,sgi.com/support/online/* and click on the "Sign Up" link.

IRIX security patches can be directly downloaded from http://support.sgi.com/colls/patches/tools/patchset/index.html

SGI will make maintenance releases of IRIX available for download for free. These are 3 release behind the current release that people will support contracts get. ie. If the latest is 6.5.23, you can download 6.5.20 with a free Supportfolio account.

There is also a way to get access to the latest maintenance releases. If you sign up for SGI's Developer program, and use **exactly** the same details as you used with your Supportfolio account, you should get access to the latest maintenance releases. If for some reason this doesn't work, email the Developer program support people and (nicely) ask for them to fix this for you.

7. Other Security Resources
==========

It is highly recommended you browse through these sites. Between them they contain a wealth of information about securing systems, how to detect hack attempts, and mailing lists with vendor patches and security alert announcements.

CERT	-	http://www.cert.org
SANS	-	http://www.sans.org
SecurityFocus	-	http://www.securityfocus.com