

## Building a Secure User Environment with SSH ChRootGroups

Blake R. Swopes 2001-07-23

### Building a Secure User Environment with SSH ChRootGroups

by Blake R. Swopes

last updated July 23, 2001

---

#### What It Is

Chroot, as you may know, alters the effective root directory of a user or process to one specified by the root user. Any resources outside of the chrooted environment ("jail" or "cage") would be inaccessible to that user. Obviously, chroot can be broken if the user gains root access, so this is not entirely fool-proof. However, chroot can make things very difficult for an attacker; and that's what we like.

Some popular uses for chroot involve setting daemons to run inside a chrooted cage. Should the daemon be compromised, a properly chrooted environment will minimize the damage an attacker can do, or at least serve as another obstacle in their path. Chroot is also a popular feature in many ftp daemons; granting users access to only a limited portion of the filesystem.

With SSH version 2.1.0, [SSH Communications Security](#) introduced the ChRootGroups feature; which provides a quick and easy way for administrators to lock users inside a chrooted cage.

Thus far, chroot has not been widely used for creating secure user environments; the difficulties involved with creating a functional cage are an obstacle that still needs to be overcome. Lack of support and the sparsity of chroot products makes the administrator's job even more difficult. These are some of the reasons that SSH ChRootGroups is such an interesting feature.

SSH 2.1.0 (and above) can be set to automatically chroot specified users and groups within their home directory. This gives administrators an excellent start toward providing a secure environment for their users. It's built into a daemon that you're probably already running, so there's no extra overhead. And since it's SSH, you also have the added benefit of encrypted sessions with no extra effort.

Many chroot solutions require that the login shell be setuid root in order to execute chroot(). Since sshd runs as root and handles the chroot() call, there is no need for suid root files within the jail. This greatly reduces the risk of a chrooted user obtaining root privileges and breaking out of the chrooted environment.

Of course, there are limitations to ChRootGroups. For example, ChRootGroups doesn't work with vanilla FTP. If sftp isn't workable for you (and actually, there are reasons not to use sftp with ChRootGroups, which I'll address later), you'll need to use an FTP daemon that includes its own Chroot feature. Luckily, as I have already mentioned, these are very popular (e.g., wu-ftpd, proftpd).

Since it is a feature of SSH, ChRootGroups doesn't work with Telnet. If you can't require your users to stick to ssh, ChRootGroups may not be the solution for you.

Finally, if disk space is an issue, then the additional binaries needed for use within the cage might cause problems. You may want to give some thought to putting users inside the same cage, or using multiple hard links to the binaries, which you can store on the same partition as the chrooted user's home directories.

## Making It Work

The first step in setting up ChRootGroups would obviously be to create your Chroot group account. For the purposes of this document, we'll use the group name 'chrooted'. I recommend putting users into their own directory tree, such as [/usr]/home/chrooted/\* for ease of management. This will separate the chrooted users from normal user accounts, while still keeping everything in a central location.

SSH provides a program (ssh-chrootmgr) for installing a dummy shell in user directories if all you want is a chrooted account for use with sftp. Here's what a user would see if they tried to log into an account with the ssh dummy shell:

```
[bswopes@shiva bin]$ ls
sftp-server  sftp-server2  ssh-dummy-shell
[bswopes@shiva bin]$ ./ssh-dummy-shell
This is ssh-dummy-shell.  Edit $(ETCDIR)/ssh2/ssh_dummy_shell.out
in order to alter this message.
```

Press any key to exit.

```
[bswopes@shiva bin]$
```

As you can see, this dummy shell won't be of much use if you want users to be able to log in; we'll need to build our own jail.

Probably the easiest way to start building the jail is first to create a separate skeleton directory for use with chrooted users. I use /etc/chrootskel/, but if you have / on a separate partition without a lot of extra room, you may find that you need to place it elsewhere, such as /home/chrooted/skel/. Also, you may want to write an "addchrootuser" script for ease of administration; this will be especially helpful when testing the cage. Here's a simple version:

```
#!/bin/sh
#
# Add Chroot User - Blake R. Swopes - 07/01/01
#
# Creates a user account for use with SSH ChRootGroups.
#
#
#          usage: addchrootuser
#
USER=$1
GROUP="chrooted" # Chroot group

SKEL="/etc/chrootskel/" # Location of chroot skeleton directory
HOME="/home/chrooted/$USER/" # Home directory for chrooted users
SHELL="/bin/sh" # Shell chroot users will use

ADDUSER="/usr/sbin/useradd" # Path to adduser/useradd
COMMENT="$USER - ChRooted User" # Comment for user's geckos field

echo -e "\nCreating user: $USER\n"

echo "User's Home Directory will be: $HOME"
echo "Copying base directory structure from: $SKEL"

#echo -e "Running: $ADDUSER -c \"$COMMENT\" -d $HOME -m -k $SKEL -g $GROUP
-s $$
HELL $USER\n\n"

# Create the user
$ADDUSER -c "$COMMENT" -d $HOME -m -k $SKEL -g $GROUP -s $SHELL $USER

# Set the password
exec passwd $USER
```

Building the jail is the most difficult part of creating a secure chrooted user environment. Simply copying files over won't do. Ordinarily, most programs are built using shared libraries (dynamically linked); this allows administrators to update the libraries without recompiling all the programs linked

to them. Unless you also place these libraries within the cage and rebuild the directory of library files (using `ldconfig`) you'll be in for a lot of errors (This is the method described in the [Chroot BIND-HOWTO](#).)

Most binaries within the cage can be linked statically to avoid problems accessing these libraries. Often this can be done with a switch for `configure` (`--enable-static-link`), `make` (`LDFLAGS=-static`) or by editing the `Makefile`. You may find that this leads to extra work down the line as you update the contents of your cage, though, so the choice is up to you.

The first bit of research toward building a cage has already been done for you by the folks over at [Linux From Scratch](#). They have information about building `bash` and other utilities for use in a `chrooted` environment (which is part of the process of building your own linux distribution) in Chapter 5 of their book. Utilities like `ls`, `cp` and `cat` will be simple enough. The trouble comes when you want to start adding text editors and the like; anything that requires special formatting, like a full screen visual interface, will be tricky.

At a minimum, you probably will want to provide your users with a statically linked shell, as well as portions of the `fileutils` and `textutils` packages (`ls`, `cp`, `cat`, etc.).

Dynamically Linked binaries can cause problems if the necessary libraries are missing or if there is a problem with the directory of library files inside the cage. In the short term, you may find it easier to work with statically linked binaries.

```
[root@shiva bin]# /usr/sbin/chroot /etc/chrootskel/ /bin/sh
sh-2.05# vim
sh: /bin/vim: No such file or directory
sh-2.05# exit
[root@shiva bin]# file vim
vim: ELF 32-bit LSB executable, Intel 80386, version 1, dynamically linked
(uses shared libs), stripped
[root@shiva bin]#
```

Statically linked binaries work just fine, but can lead to extra work down the line; if a library is updated, the admin will need to recompile all binaries that make use of that library, not just the library itself. This is the main drawback to not using dynamically linked binaries.

```
[root@shiva bin]# file ls
ls: ELF 32-bit LSB executable, Intel 80386, version 1, statically linked,
not stripped
[root@shiva bin]# /usr/sbin/chroot /etc/chrootskel/ /bin/sh
sh-2.05# ls
```

```
bin etc lib public_html
sh-2.05# exit
[root@shiva bin]#
```

Something important to consider while building your jail is that there are a lot of programs you probably shouldn't include. Do you want users to be able to create executable shell scripts? If not, you probably shouldn't include chmod. Do you want them to be able to create device files, symbolic or hard links? No? Then don't include mknod or ln. You should carefully consider every file that you are going to include in the cage.

Remember, you are creating a restricted environment. Usability is important, but in this case, security comes first.

## Building A Simple Cage

Building a simple cage isn't very difficult; it's when you begin adding more functions for the users that you start to run into problems. For those of you who are interested, here is a walkthrough for building a basic jail.

The purpose of this jail will be to allow users some basic shell access to control their personal web pages. To begin, I create the /etc/chrootskel/ directory, along with a public\_html/ and etc/ directory. In the web directory, I place a simple "coming soon" index.html file for testing purposes. If you don't have a lot of free space to work with on your / partition, you may want to use another directory as your skeleton.

I also copy /etc/profile into the etc/ directory, as well as .bash\_profile and .bashrc into the /etc/chrootskel/ directory (careful to examine their contents) for use with bash login.

The next step will be to build bash for the jail. This will create a statically linked copy in /etc/chrootskel/bin/.

I've removed the output from many of these processes for readability.

```
[root@shiva archive]# tar -zxf bash-2.05.tar.gz
[root@shiva archive]# cd bash-2.05
[root@shiva bash-2.05]# ./configure --enable-static-link
--prefix=/etc/chrootskel/
[root@shiva bash-2.05]# make && make install
[root@shiva bash-2.05]# cd /etc/chrootskel/
[root@shiva chrootskel]# ls
```

```
bin etc info man public_html
[root@shiva chrootskel]#
```

We don't need those info or man pages, so lets get rid of them...

```
[root@shiva chrootskel]# rm -rf info/ man/
[root@shiva chrootskel]# ls
bin etc public_html
[root@shiva chrootskel]#
```

Now we make sure we only have what we need in bin/ and create a symbolic link to bash for sh.

```
[root@shiva chrootskel]# cd bin/
[root@shiva bin]# ls
bash bashbug
[root@shiva bin]# rm -f bashbug
[root@shiva bin]# ln -s bash sh
[root@shiva bin]# ls
bash sh
[root@shiva bin]#
```

At this point, you would be able to create a chrooted account and log in. However, you would be extremely limited in what you would be able to accomplish.

Next we'll build static fileutils and textutils for the cage.

## FileUtils

```
[root@shiva archive]# tar -zxf fileutils-4.1.tar.gz
[root@shiva archive]# cd fileutils-4.1
[root@shiva fileutils-4.1]# ./configure --disable-nls
--prefix=/etc/chrootskel/ --libexecdir=/etc/chrootskel/bin/
--bindir=/etc/chrootskel/bin/
[root@shiva fileutils-4.1]# make LDFLAGS=-static && make install
[root@shiva fileutils-4.1]# cd /etc/chrootskel/
[root@shiva chrootskel]# ls
bin etc info lib man public_html
[root@shiva chrootskel]# ls lib/
```

lib/ is empty so lets kill it, along with info and man.

```
[root@shiva chrootskel]# rm -rf info/ man/ lib/
[root@shiva chrootskel]# cd bin/
[root@shiva bin]# ls
```

```

bash    chown  df          du          ls          mknod  rmdir    sync
chgrp  cp      dir        install    mkdir      mv      sh       touch
chmod  dd      dircolors  ln          mkfifo    rm      shred   vdir

```

Consider whether you want each program available to your chrooted users. Get rid of those you don't.

```

[root@shiva bin]# rm -f chown df du mknod sync chgrp install chmod dd
dircolors ln mkfifo shred
[root@shiva bin]# ls
bash cp dir ls mkdir mv rm rmdir sh touch vdir
[root@shiva bin]#

```

## TextUtils

```

[root@shiva archive]# tar -zxf textutils-2.0.tar.gz
[root@shiva archive]# cd textutils-2.0
[root@shiva textutils-2.0]# ./configure --disable-nls
--prefix=/etc/chrootskel/ --libexecdir=/etc/chrootskel/bin/
--bindir=/etc/chrootskel/bin/
[root@shiva textutils-2.0]# make LDFLAGS=-static && make install

[root@shiva textutils-2.0]# cd /etc/chrootskel/
[root@shiva chrootskel]# rm -rf info/ man/
[root@shiva chrootskel]# cd bin/
[root@shiva bin]# rm -f comm paste sum tr expand join pr tac tsort csplit
fmt nl ptx unexpand cksum cut fold md5sum od split uniq
[root@shiva bin]# ls
bash cp head mkdir rm sh tail vdir
cat dir ls mv rmdir sort touch wc
[root@shiva bin]#

```

As you can see, the cage now contains some basic commands. Enough to move around and do some simple file manipulation. There are no suid files, no compilers, and the user will not be able to chmod files +x. Lets take a look at an example of why chmod could be an issue...

## Threats to ChRootGroups Security

### chmod

Our sample cage was designed to provide users the ability to manipulate web sites. What if there is a mistake in the security settings of the web server, and users with these chrooted accounts can execute CGI's. If they can't chmod +x, then this wont do any harm, but if they can...

```
sh-2.04$ cat >exploit.cgi
#!/bin/sh
cp /bin/sh /home/chrooted/test/
chmod a+s sh
sh-2.04$ chmod a+x exploit.cgi
```

Now, the attacker simply requests the cgi, and ...

```
sh-2.04$ ls -l
-rwsr-sr-x    1 nobody    nobody          512540 Jul  8 21:06 sh
```

## SSH 1

Ironically, some serious threats for SSH ChRootGroups come from ssh itself. Many administrators install ssh 2 on top of ssh 1 for backward compatibility, however ssh 1 did not offer chrootgroups support. An attacker can simply sign in using an ssh 1 client, and slip right past your cage. Since scp is provided by ssh 1, it is dangerous as well.

SSH 1 login as chrooted user:

```
sh-2.04$ pwd
/home/chrooted/test
sh-2.04$
```

## SCP

While ssh 2 does provide an scp client, the scp server is a part of ssh 1 (implemented in ssh 2 via ssh 1 compatibility). Because of this, scp also is not designed to work with chrootgroups; it will allow an attacker to place files on the system outside of the cage. When combined with ssh 1 shell access to the system, scp could place you at severe risk of compromise.

The simple solution to both these problems is to install only ssh 2.

SCP login to a pure SSH 2 server:

```
[root@shiva /root]# /usr/local/bin/scp index.html
bswopes@localhost:index.html
bswopes@localhost's password:
scp: warning: child process (/usr/local/bin/ssh2) exited with code 0.
```

```
[root@shiva /root]#
```

## SFTP

SFTP can be made to work with ChRootGroups, but it has its own risks. There is no way to set umask for SFTP; an attacker could easily upload executable exploits to their home directory, which would retain their permissions after the transfer (but not their ownership, they will be owned by the chroot users.) With this issue, not including chmod in the skeleton directory becomes moot.

```
[bswopes@shiva bswopes]$ ls -l exploit
-rwxrwxr-x    1 bswopes  bswopes          0 Jul  3 12:52 exploit
[bswopes@shiva bswopes]$ sftp test@localhost
test@localhost's password:
sftp> cd public_html
/public_html
sftp> put exploit
exploit      |                0 kB |    0.0 kB/s | TOC: 00:00:01 |
100%
sftp> ls -l
-rw-rw-r--    1 501      501          112 Jul  3  8:34 index.html
-rwxrwxrwx    1 501      501           0 Jul  3 19:53 exploit*
sftp> quit
```

For users within the cage, your biggest concern is their ability to become root. Even within a chrooted environment, if an attacker can become root, they can take control your system. However, you should be able to design a jail that won't provide attackers with the means to become root. For example, there will usually be no call for suid programs within the jail. Consider setting the /home partition nosuid in /etc/fstab.

Setting a partition nosuid prevents the suid bit from taking effect. Suid programs on a nosuid partition will run as the user executing them, or not at all. With nosuid set for the partition, you could practically leave a suid root shell in the cage (though I wouldn't recommend it).

Example from /etc/fstab:

```
/dev/hda5          /home              ext2      nosuid      1
2
```

Executing a suid root shell on a nosuid partition can be done as follows:

```
[bswopes@shiva bswopes]$ pwd
```

```

/home/bswopes
[bswopes@shiva bswopes]$ ls -l sh
-rwsr-sr-x    1 root    root          316848 Jul 17 19:54 sh
[bswopes@shiva bswopes]$ ./sh
bash: ./sh: Operation not permitted
[bswopes@shiva bswopes]$

```

In most cases, chrooted users should not be able to make outbound connections; there's no need for telnet, ping or traceroute (remember last year's traceroute -g exploit?). They won't be running daemons, they probably won't be printing documents, and no matter when that assignment for their fortran class is due, they do not need access to a compiler; not on your system anyway...

Give some thought to permissions and ownership. Do you want them being the owner of every file within their home directory? Even those executable ones? Your first reaction may be to let them destroy their user environment if they're foolish enough to do so. But if you don't want your users writing exploits on top of one of the lesser used programs (retaining their executable status), then you may want to consider having those programs (as well as the directories) owned by root or another user, like nobody.

```

sh-2.05$ cd bin/
sh-2.05$ ls -l ls
-rwxr-xr-x    2 501      501      1963528 Jul  2 08:31 ls
sh-2.05$ cat >ls
#!/bin/sh
echo "I am a root exploit!"
sh-2.05$ ls
I am a root exploit!
sh-2.05$

```

## Recap

To summarize, ssh chrootgroups can provide you with a good start toward creating a secure user environment, without some of the risks involved with other chroot solutions.

Don't forget to investigate ways of bypassing the cage; for example, if you are running ssh2 with ssh1 compatibility, you may as well not be running chroot at all. Look at the file transfer methods available to your chrooted users, are they chrooted through ftp as well? Do the files they transfer retain their original permissions?

Building the jail can be tough, but there are a few tips to remember that will keep you on the right path. Most of the binaries you want in your cage should be statically linked, unless you want to work

with libraries and ldconfig. Don't include any suid files or compilers. Give careful consideration to file permissions, and whether files belong in the cage at all. Finally, if your cage is growing so large you find that it contains files you don't recognize, then you are probably doing something wrong.

When properly configured, this chroot solution can provide your users with all the functionality they need without endangering the security of your system or the privacy of other users. Encrypted communication, and limited system access are a solid combination that can be hard to crack. If you have any feedback, or would like to share your success stories, feel free to contact me.

[Privacy Statement](#)

Copyright 2006, SecurityFocus