1, 2, 3 CHROOT JAIL
Author - Nic Maurel

With the Advent of Redhat finally moving into the 21st century we are finally able to create a chroot sftp server without modified packages. A chroot sftp server is a sftp server that has a locked down home directory to prevent untrusted users from fiddling with the underlying operating system. A typical scenario where one would use this is a secure upload and dropoff for acquiring larger files from untrusted third parties.

To start
I used the free version of Redhat, CentOS release 6.4 (Final), with the stock standard *openssh-server-5.3p1-84.1.el6.i686*. This combination supports what I'm about to show you. Edit your */etc/ssh/sshd_config* ...

```
# vi /etc/ssh/sshd_config

hash out this value
#Subsystem      sftp    /usr/libexec/openssh/sftp-server
Subsystem       sftp    internal-sftp

add these four lines at the bottom of the file
Match group sftpusers
ChrootDirectory /home/%u
X11Forwarding no
ForceCommand internal-sftp

Save and exit the file
```

- *Match group sftpusers* will be the user group that you use for sftp chroot jail. So users that I want jailed must have their primary group configured as *sftpusers*.
- *ChrootDirectory /home/%u* is the directory you would use as home directories for your jailed users where *%u* is the username variable
- *X11Forwarding no* just prevents the display from being forwarded to the shell
- *ForceCommand internal-sftp* tells openssh that the users must be forced into a more restrictive root subsystem

Then we restart sshd with "*/etc/init.d/sshd restart*".

Prepare and create our users
Let's create the group we used above ..

```
# groupadd sftpusers
```

Now we create a user that will use the sftp chroot jail. For your convenience I have created a bash script that does all the creation for you, all you need to do is create type a username and password. (always run as root user)...

```
if [[ -z $1 ]]
        then
                echo ""
                echo "Usage: enter a username after command";
                echo "eg. add-sftpuser bob"
                echo ""
        else
                echo "- creating $1 chroot directory";
                mkdir /home/$1;
                echo "- creating user $1"
                useradd -s /bin/false -g sftpusers $1 > /dev/null 2>$1;
                echo "- modifying home dir for user $1";
                usermod -d / $1;
                echo "- creating home folder for user $1";
                mkdir /home/$1/home;
                echo "- changing home folder ownership for user $1";
                chown -Rf $1 /home/$1/home;
                echo ""
                echo "- Please set password for user $1";
                passwd $1;
fi
```

In a nutshell the script creates a home directory named after the user you type in. For example, we want to create the user *andrew*, we would substitute *$1* (our first output) for *andrew*

```
# mkdir /home/andrew
```

then, creates a user with no shell, primary group sftpusers and discards some unwanted output to */dev/null*

```
# useradd -s /bin/false -g sftpusers andrew > /dev/null 2>$1
```

then, modifies the new user's home dir to /

```
# usermod -d / andrew
```

creates an upload folder and gives the new user ownership

```
# mkdir /home/andrew/home
# chown -Rf andrew /home/andrew/home
```

creates password for the user that you will be required to type in

```
# passwd andrew
```

Here is the script in action, I named my script *add-sftpuser*

```
# /admin/bin/add-sftpuser andrew
- creating andrew chroot directory
- creating user andrew
- modifying home dir for user andrew
- creating home folder for user andrew
- changing home folder ownership for user andrew

- Please set password for user andrew
Changing password for user andrew.
New password:
BAD PASSWORD: it is based on a dictionary word
BAD PASSWORD: is too simple
Retype new password:
passwd: all authentication tokens updated successfully.
```

Ok so I created a weak password, this is just a demonstration hopefully my passwords will be stronger in a live environment :) Lets test our user ..

```
# sftp andrew@192.168.1.7
Connecting to 192.168.1.7...
```

```
andrew@192.168.1.7's password:
sftp> pwd
Remote working directory: /
sftp> cd /
sftp> pwd
Remote working directory: /
sftp> ls
home
sftp> cd home
sftp> ls
sftp> mput 1.png
Uploading 1.png to /home/1.png
1.png 100% 90KB 90.4KB/s 00:00
sftp> ls
1.png
sftp> bye
```

As you can see I am locked down to my home directory and have one directory that I can upload files to. Superb, safe and secure. But...

Automate
What's left? Well what if I didn't just want to use username and password. What if you need you automate. So the third step I will show you how to automate login to the chroot jail. The way I do it differs from the norm but it works for me. Instead of using *authorized_keys* in each users home directory, I will create a directory and put all the public keys there and modify the *sshd_config* to reference the keys. Each public key for each user will be referenced as a username variable- eg *andrew.pub*. So edit your *sshd_config* file ..

```
# vi sshd_config
#AuthorizedKeysFile     .ssh/authorized_keys
AuthorizedKeysFile      /etc/sftpkeys/%u.pub

Save and exit the file. Restart sshd
# /etc/init.d/sshd restart
```

add your public key file to */etc/sftpkeys/andrew.pub* (this file looks exactly the same as any *authorized_keys* file). Now each of my keys is stored in one location for backup purposes. Lets test our user.

```
# sftp andrew@192.168.1.7
Connecting to 192.168.1.7...
sftp>
```

1 2 3 Chroot Jail, no modified packages, stock standard and can be undone easily if need be. This functionality is definately something I will be making use of in future.
A few gotcha's - disable selinux, and if your openssh package is too old it will fail to restart as it does not support the newer configuration commands.
Enjoy!