Chrooting SSHd on Linux

Date: July 13th 2007

Author: Paul Sebastian Ziegler

E-Mail: psz _at_ observed _dot_ de

Website: https://observed.de

Content

0x01 Chrooting the daemon itself

There are two different approaches when it comes to *chroot* and *sshd*. The first one will run the SSH daemon in a normal environment and then chroot selected users as they connect to the system. This approach is well understood and supported by patches. It can be a great help if you want to separate users and keep privileges separate. However sometimes it may become necessary to jail the sshd itself. Of course all users who connect to the chrooted sshd will end up in the same jail as well. This paper discusses the later approach with all it's advantages and drawbacks.

0x02 Who needs this?

There are several reasons why you might want to jail sshd. The most important of them is to delay or prevent all attacks based on weaknesses in the sshd itself. This might not be necessary for systems under normal circumstances. However sometimes a high-security box is administrated locally and the sshd is only used to allow shell-access for limited users. Also this technique might delay an attacker if the box worked with will be unmaintained for some time for some reason.

Please be aware that you should only use this approach if you do not plan to create individual chroots for each user and have a good hang for paranoia and an excellent understanding of what you are doing.

0x03 Why was this written?

For several reasons I decided to jail an sshd some days ago. Unfortunately the materials that can currently be found online are either out of date, completely useless (some even suggested to put a complete Linux system into the chroot – including suid-files and devices..) or explaining how to jail the users only. Therefore I decided to try for myself. After having spent days attaching strace to forking daemons and greping through the output to determine all the needed extras I eventually succeeded. Since I figured that some may want to chroot sshd as well I decided to write this paper and share the information I won. I can and will not guarantee that this paper's approach is secure or complete. However it has been severely tested and should at least save you a couple 1000 lines of strace output.

0x04 Assumptions

All the file's paths are the ones used on Gentoo-Linux[1]. However the approach should work with all Linux distributions. Some parts of it might also apply to BSD, MacOS X or Solaris, however it has not been tested on any of these.

The SSH daemon used is OpenSSH-4.5p1[2] compiled with support for pam and tcpd.

I assume that you know how to administrate a Linux system and how to solve individual problems as they arise. Also I assume that you know what you are doing. I can not stress this enough: A bad chroot will make your system's security *worse* instead of hardening it.

Sometimes libraries and executables need to be copied into the jail including all their runtime dependencies. I will not go into detail regarding these dependencies. You can either manually resolve and copy all of them by repeatedly using *ldd* on them or simply use a tool to automatically resolve and copy all dependencies. One of those tools is jailkit[3] which is released as open source and comes with many features that make the creation of secure jails much easier.

It is assumed that the jail's path is */ssh* you will have to change all the instructions in this paper relative to your specific path.

0x05 Building the jail – Files

The first thing to do is to build a minimal jail that will allow you to actually chroot into it, get a shell and deal with some users. There are several approaches to doing this. You could either install a very minimal Linux system (not recommended), manually tune your own basic jail (good but hard to realize) or use jailkit's *jk_init* to have the process automatized.

Jk_init will allow you to create minimal environments for several occasions. The ones I used for this examples were *extendedshell*, *logbasics*, *netbasics* and *uidbasics*. So if you are of the lazy kind the easiest way to get started is to simply issue the following command:

 # jk_init -f -v /ssh extendedshell logbasics netbasics uidbasics

You should end up with a working jail you can chroot into and perform easy commands like *ls* and *cat*.

Now the time has come to copy all the required files into the jail. The following is a list of all files and directories that will need to be copied. Some of them may already have been pulled in as a dependency to other files.

/etc/gai.conf

/ets/hosts.conf

/etc/hosts

/etc/nsswitch.conf

*/etc/pam.d/**

/etc/profile

/etc/resolv.conf

*/etc/ssh/**

*/lib/libpam**

*/lib/security/**

Once more you can either manually copy those files using the *cp* command for copying and *ldd* for tracking dependencies or rely on jailkit's *jk_cp* to automate the job for you.

Other libraries will be required as well. However they will be pulled in as direct dependencies of sshd.

The next step is to create some additional folders that the sshd will require. Don't worry, there will only be two of them:

/var/empty

/var/run

0x06 *Jailing the users*

Of course you will need a couple of users in the chroot that are actually able to connect to the daemon. I assume that the users to jail already exist on your system. So all you have to do is to create a home directory for each user and grep their information out of the various files:

*mkdir /ssh/home/***username**

grep **username** */etc/passwd >> /ssh/etc/passwd*

grep **username** /etc/shadow >> /ssh/etc/shadow

grep **username** /etc/group >> /ssh/etc/group

0x07 Building the jail – Devices

The sshd will need a couple of devices to function properly. Those devices can either be manually created using the appropriate *mknod* parameters or automatically created using *jk_cp*. Anyhow, you will need the following:

null

ptmx

pts

urandom

Furthermore you will need one pty and one tty for each user you want to allow to log in. The quickest way to create them is to issue the following commands:

*jk_cp -f -v /ssh /dev/tty**

jk_cp -f -v /ssh /dev/tty

*jk_cp -f -v /ssh /dev/pty**

0x08 Building the jail – /proc

The SSH daemon will need to access the /proc filesystem in order to map pty-devices to tty-devices. If you do not mount proc into the chroot, the daemon itself will run just fine. However each and every user who tries to connect into the system will have his/her session closed immediately after

authentication took place since the shell could not be bound to the socket due to the limitation explained above.

There currently seems to be no other way then to mount proc within the jail:

 # mkdir /ssh/proc

 # mount -t proc none /ssh/proc

0x09 Jailing the daemon

The time to jail the daemon has finally come. All the preparations have been completed, so feel free to move */usr/sbin/sshd* into the jail using any approach you want. I recommend using *jk_cp* here as well since it will be quite some work to resolve all the dependencies for this executable.

 # jk_cp -v -f /ssh /usr/sbin/sshd

Now that the daemon is jailed you can start it and try to login as one of the jailed users.

 # chroot /ssh /usr/sbin/sshd

If you can't connect for some reason there are three different approaches to solve your problem:

1) If the client receives any kind of error – just fix it
2) Go over this paper again and double check all the steps you took
3) Use strace to look for any kind of error or irregularity while running the daemon

   # strace -f -F chroot /ssh /usr/sbin/sshd | less

0x0A Logging

The finishing touch. You will probably want to be able to log your newly chrooted sshd's messages. Be it for error-analysis or attack detection through some sort of IDS/IPS. This is actually very easy. The following example will show you how to configure syslog-ng[4] to log all messages from the jail.

Syslog-ng is configured in */etc/syslog-ng/syslog-ng.conf*. Near the top you will find one or more lines like the following:

source src { unix-stream("/dev/log"); internal(); };

Now guess what you will have to do to make syslog-ng work within the jail as well... Exactly. Create another line exactly under the one(s) similar to what you can see above. However this time let it point to */ssh/dev/log*:

source src { unix-stream("/ssh/dev/log"); internal(); };

Restart syslog-ng and you are all set.

0x0B Final thoughts

I hope this paper has been helpful for you. I would have wanted something like this when I myself was searching for various solutions. Therefore I decided to write a paper myself once I was finished. Any kind of feedback, new/better ideas and corrections is always welcome.

Since you took the trouble to jail the sshd itself I can assume that security is an important factor for you. Considering this you should probably take a look at the vast possibilities of hardening your jail and making a breakout even harder. The easiest ways to do this probably are the GRSecurity[5] patch for the Linux kernel and a careful layout of partitions mounted with *nodev*, *nosuid*, *noexec*, or *ro* – depending on the specific partition of course.

0x0C License

0x0D References

[1] http://gentoo.org

[2] http://www.openssh.com/portable.html

[3] http://olivier.sessink.nl/jailkit/

[4] http://www.balabit.com/network-security/syslog-ng/opensource-logging-system/

[5] http://www.grsecurity.net/

[6] http://www.gnu.org/licenses/lgpl.html