LinuxFocus article number 225
http://linuxfocus.org

Chrooting All Services in Linux

by Mark Nielsen (homepage)

About the author:
Mark works as an independent consultant donating time to causes like GNUJobs.com, writing articles, writing free software, and working as a volunteer at eastmont.net.

Abstract:

Chrooted system services improve security by limiting damage that someone who broke into the system can possibly do.

Introduction

What is chroot? Chroot basically redefines the universe for a program. More accurately, it redefines the "ROOT" directory or "/" for a program or login session. Basically, everything outside of the directory you use chroot on doesn't exist as far a program or shell is concerned.

Why is this useful? If someone breaks into your computer, they won't be able to see all the files on your system. Not being able to see your files limits the commands they can do and also doesn't give them the ability to exploit other files that are insecure. The only drawback is, I believe it doesn't stop them from looking at network connections and other stuff. Thus, you want to do a few more things which we won't get into in this article too much:

- Secure your networking ports.
- Have all services run as a service under a non-root account. In addition, have all services chrooted.
- Forward syslogs to another computer.
- Analyze logs files
- Analyze people trying to detect random ports on your computer
- Limit cpu and memory resources for a service.
- Activate account quotas.

The reason why I consider chroot (with a non-root service) to be a line of defense is, if someone breaks in under a non-root account, and there are no files which they can use to break into root, then they can only limit damage to the area they break in. Also, if the area they break into is owned mostly by the root

account, then they have less options for attack. Obviously, there is something wrong if someone actually does break into your account, but it is nice to be able to limit the damage they can do.

PLEASE REMEMBER that my way of doing this is probably not 100% accurate. This is my first attempt at doing this, and if it just partially works well, it should be easy to finish out the rough edges. This is just a roadmap for a HOWTO I want to create on chroot.

How are we going to chroot everything?

Well, We create a directory, "/chroot" and we put all of our services under there in the following format:

- Syslogd will be at chrooted with each service.
- Apache will be at /chroot/httpd.
- Ssh will be at /chroot/sshd.
- PostgreSQL will be at /chroot/postmaster.
- Sendmail will be chrooted, but it won't be running under a non-root account, unfortuantely.
- ntpd will be chrooted to /chroot/ntpd
- named will be chrooted to /chroot/named

Each service should be completely isolated.

My Perl script to create chrooted environments.

Config_Chroot.pl.txt should be renamed Config_Chroot.pl after you download it. This perl script lets you list the services being installed, view the config files, configure a service, and start and stop the services. In general, this is what you should do.

1. Create the chroot directory.
   mkdir -p /chroot/Config/Backup
2. Download Config_Chroot.pl.txt to /chroot/Config_Chroot.pl
3. Change the $Home variable in the perl script if you are not using /chroot as the home directory.
4. Download my config files.

Now, the important thing here is: **I have only tested in on RedHat 7.2 and RedHat 6.2**.

Modify the perl script for your distribution.

I ended up making a huge gigantic article on Chroot, but with my Perl script, it became much smaller. Basically, I noticed after chrooting many services, they all have very similar files and configurations that needed chrooted. The easiest way to figure out which files need copying for a particular service is to look at the manpage and also type "ldd /usr/bin/file" for programs that use library files. Also, you can chroot the the service you are installing and manually start it to see what errors you get or look at its log files.

In general, to install a service do this:

```
cd /chroot
./Config_Chroot.pl config  SERVICE
```

```
./Config_Chroot.pl install SERVICE
./Config_Chroot.pl start   SERVICE
```

Chrooting Ntpd

Ntpd is just a time service that lets you keep your computer and other computers in sync with the real time. It was a simple thing to chroot.

```
cd /chroot
 # Uncomment the next line if you don't use my config file.
#./Config_Chroot.pl config  ntpd
./Config_Chroot.pl install ntpd
./Config_Chroot.pl start   ntpd
```

Chrooting DNS or named

Already done, check out
http://www.linuxdoc.org/HOWTO/Chroot-BIND8-HOWTO.html
or
http://www.linuxdoc.org/HOWTO/Chroot-BIND-HOWTO.html

Or, if you want to use my script,

```
cd /chroot
 # Uncomment the next line if you don't use my config file.
#./Config_Chroot.pl config  named
./Config_Chroot.pl install named
./Config_Chroot.pl start   named
```

Chrooting Syslog with services and my complaints.

I want to chroot syslogd. My problem is, syslogd uses /dev/log by default, which can't be seen by chrooted services. Thus, I can't syslogd easily. Here are the possible solutions:

- Chroot syslogd with every service. I actually tested this, and yes, I was able to log stuff. I don't like this since I have a root running service.
- See if we can connect to an offsite logging facility.
- Just log files to a file and not through syslogd. This is probably the best security option, although if someone breaks, they could play around with the logs.
- Configure the main syslogd to look at several locations to get all the services. You use the -a option with syslogd to do this.

My only solution was to make sure syslogd is chrooted with every service. I would like some sort of solution which would log stuff in a non-root account using its own chrooted environment, like maybe a network port. It can probably be done, but I am going to stop where I am at and figure out a better

solution later.

If you do not want to make a separate syslogd for each service, then with the main syslogd that you are running on your system, add the following command when syslogd starts:

```
syslogd -a /chroot/SERVICE/dev/log
```

If I had ssh and dns running, it might look like,

```
syslogd -a /chroot/ssh/dev/log -a /chroot/named/dev/log -a /dev/log
```

Last note on Syslogd, I wish I could make it run under a non-root account. I tried a couple of simple things, but it didn't work and I gave up. If I could run syslogd under a non-root account with each service, that would satisfy my security issues. Possibly, even have it log offsite.

Chrooting Apache

This was extremely easy to do. Once I got it setup, I was able to execute Perl scripts. Now, my config file is rather long because I had to include Perl and the PostgreSQL libraries into the chrooted area. One thing to note, if you are connecting to a database, make sure your database service is running on the 127.0.0.1 loopback device and you specify the host to be 127.0.0.1 in your Perl scripts for the DBI module. Here is an example of how I connect to a database using persistent connections in apache:

```
$dbh ||= DBI->connect('dbi:Pg:dbname=DATABASE',"","", {PrintError=>0});

if ($dbh ) {$dbh->{PrintError} = 1;}
else
  {$dbh ||= DBI->connect('dbi:Pg:dbname=DATABASE;host=127.0.0.1',"","",
      {PrintError=>1});}
```

Source: http://httpd.apache.org/dist/httpd/

Compile and install apache on your main system at /usr/local/apache. Then use the perl script.

```
cd /chroot
 # Uncomment the next line if you don't use my config file.
 # ./Config_Chroot.pl config  httpd
./Config_Chroot.pl install httpd
./Config_Chroot.pl start   httpd
```

I changed my httpd.conf file to have this stuff:

```
ExtendedStatus On

<Location /server-status>
    SetHandler server-status
    Order deny,allow
    Deny from all
    Allow from 127.0.0.1
</Location>

<Location /server-info>
    SetHandler server-info
    Order deny,allow
```

```
    Deny from all
    Allow from 127.0.0.1
</Location>
```

Then, just point your browser at http://127.0.0.1/server-status or http://127.0.0.1/server-info and check it out!

Chrooting Ssh

First off, ideally, you should port forward ssh on port 22 to port 2222. Then, when you start ssh, have it listen to port 2222 under a non-root account. For the initial ssh connection, we want to have secure accounts with passwords just to let the people in, but not do anything else. After they log in, then have a second ssh program running on port 127.0.0.1:2322 which will let them connect to the real system -- the second ssh program should ONLY listen on the loopback device. Now this is what you should do. We aren't going to do it. The only thing we are going to do is chroot ssh for this example. Exercises which are left up to the reader include putting sshd under a non-root account and to install a second sshd which listens on the loopback device to let people into the real system.

Again, we are going to just chroot ssh and let you worry about the consequences of doing that (you won't be able to see your entire system if you just do this). Also, ideally, it would be nice to set this up to record logs offsite. Also, we should use OpenSSH, but I am using the commercial SSH for simplicity (which is not a good excuse).

Source: http://www.ssh.com/products/ssh/download.cfm

Install ssh at /usr/local/ssh_chroot. Then use the Perl script.

```
cd /chroot
 # Uncomment the next line if you don't use my config file.
 # ./Config_Chroot.pl config  sshd
./Config_Chroot.pl install sshd
./Config_Chroot.pl start   sshd
```

I suppose one really good thing with putting ssh under a chrooted environment is that if you want to use it to replace an ftp server, people will have limited access to your area. Rsync and SCP go very well together for letting people upload files. I don't really like to put an ftp server up for people to log into. A lot of ftp servers are also chrooted, but they still transmit passwords in the clear, which I don't like.

Chrooting PostgreSQL

This was almost as simple as perl, except it required a few more libraries. Overall, it wasn't that hard to do. One thing I had to do was put PostgreSQL open to the network, but only on the loopback device. Since it was chrooted, other chrooted services couldn't get to it, like the apache web server. I did compile Perl into PostgreSQL, so I had to add a lot of Perl stuff to my config file.

Source: ftp://ftp.us.postgresql.org/source/v7.1.3/postgresql-7.1.3.tar.gz

Compile and install apache on your main system at /usr/local/postgres. Then use the Perl script.

```
cd /chroot
 # Uncomment the next line if you don't use my config file.
 # ./Config_Chroot.pl config  postgres
./Config_Chroot.pl install postgres
./Config_Chroot.pl start   postgres
```

Chrooting Sendmail

Go ahead and execute my script.

```
cd /chroot
 # Uncomment the next line if you don't use my config file.
 # ./Config_Chroot.pl config  sendmail
./Config_Chroot.pl install sendmail
./Config_Chroot.pl start   sendmail
```

Now are there catches? Yes. It is still running as root. Darn. Also, certain files are recreated by the /etc/rc.d/init.d/sendmail file when it is started. Mine script doesn't handle that. Anytime you make changes to sendmail under /etc/mail, please copy the changes to /chroot/sendmail/etc also. Also, you will have to point /var/spool/mail to /chroot/sendmail/var/spool/mail so that the sendmail program and the users (when they log in) can see the same files.

The good thing is, you can always send mail out, it is just receiving it that is the problem. Thus, I was able to install sendmail with apache without any problems. Some of my perl scripts send mail out, and so, I needed the sendmail files copied into the chroot area for apache.

Other things to chroot.

Here is my philosophy:

1. Everything should be chrooted, including sendmail, ssh, apache, postgresql, syslog, and any service running on the computer.
2. Everything should be put under a non-root account (you might need to port forward protected ports to a non-protected port). This includes sendmail and syslog by the way.
3. Logs should be sent offsite.
4. A partition should be setup for each service to limit the amount of diskspace a hacker can use up if they decide to write files. You could use a loopback device to mount files as filesystems for some of these services if you run out of partitions.
5. Root should own all files that do not change.

Now, when it comes to sendmail and syslogd, I still think they should be run under a non-root account. For sendmail, this should be possible, but I found it extremely difficult to run as a non-root account. I haven't been successful getting sendmail to run as a non-root account, and I think it is a serious mistake for it not to be. I know there are problems doing that, but I think they can ALL be taken care of. As long as file permissions are taken care of, I don't see why sendmail needs to be run as root. There might be some reason I am overlooking, but I doubt any of the obstacles can't be overcome.

For syslog, I haven't even tried, but I would say logs should should be logged under a non-root account

and I don’t see why that shouldn’t be possible. At least I was able to get syslog to be chrooted for each service.

All services should be setup as non-root accounts. Even NFS. Everything.

Suggestions

- Use two logins for ssh and have two running sshd daemons.
- Figure out how to get sendmail or some other mail program running as non-root.
- Strip out the unnecessary libraries under /lib. I just copied everything to make it easy on myself. Most of it you don’t need.
- Do remote logging of syslogd and find out if we can attach syslogd to a network port and get all the services to connect to that network port on the loopback device. See if we can get syslogd to run as a non-root account.

Conclusion

I think chroot is cool for all services. I believe it is a big mistake not to chroot all services under non-root accounts. I wish a major distributions would do that, or a smaller distribution: ANY distribution. Mandrake started off by taking stuff from RedHat and expanding off of it, so perhaps, someone should take Mandrake and expand chroot off of them. Nothing prevents people from redoing other people’s work in GNU/Linux, so I think it is possible. If some company wanted to chroot everything and create a systematic easy environment for people to manage their chrooted services, they would have a fantastic distribution! Remember, now that Linux is going mainstream, people don’t want to see the command line, so if everything is done at a gui level, they don’t need to see the guts and they really don’t need to know what is going on -- they just need to be able to configure it and know that it just works!

I am in 100% complete support of the idea that all services should be chrooted with non-root accounts and that any distribution that doesn’t do this is less than proper for me to use in a production environment. I am going to chroot everything, as much as possible -- eventually I will get there.

I plan on creating s HOWTO about chrooting. I am submitting a request to have someone help me convert this article into LyX format so that it can be put in the HOWTOs for Linux.

References

1. If this article changes, it will be available here http://www.gnujobs.com/Articles/23/chroot.html

Translation information:
en --> -- : Mark Nielsen (homepage)

2005-01-14, generated by lfparser_pdf version 2.51