

OpenBSD Guide

Taylor Venable

October 22, 2009

Abstract

How to do various useful and important things on an OpenBSD server.



Contents

1	Encrypted Partition	1
2	Anonymous CVS Access over SSH	2

1 Encrypted Partition

Like many things in OpenBSD, setting up an encrypted partition is simple and straightforward. The modern way of doing this is to use the `softraid(4)` framework, which supports `RAID0`, `RAID1`, and encrypted partitions as drivers. Say I have a brand new disk that I want to use for an encrypted backup. Starting from the first time I plug it in, let's call it `sd0`. First I must set up an OpenBSD slice, and create a partition that will contain the encrypted virtual disk.

```
# fdisk -iy sd0
# disklabel -E sd0
```

Inside `disklabel`, create an entry with type `RAID` (for comparison, the default type is `4.2BSD`). I now have a partition `sd0a` that I want to encrypt. First we need to configure the `softraid` framework to create a virtual disk of the partition and encrypt it.

```
# bioctl -c C -l /dev/sd0a softraid0
```

You will be prompted for a password which you must then confirm. You'll then get kernel messages about a new disk being attached, in my case this was `sd1`. Now the recommendation is to zero out the first megabyte of the disk and put a slice on it.

```
# dd if=/dev/zero of=/dev/rsd1c bs=1m count=1
# fdisk -iy sd1
```

Finally you have a disk that you can create partitions and filesystems in.

```
# disklabel -E sd1
# newfs /dev/rsd1a
```

In order to mount this partition again after a reboot, simply issue the same `bioctl` command and enter the same password you used when you set up the disk. It was recommended to me to put these commands into `rc.local`, and since this happens after `fstab` is processed, I had to mount it myself. Here's my `fstab` entry:

```
/dev/sd1a /home ffs rw,noauto,nodev,nosuid,noatime,softdep 0 0
```

And then I added this to `rc.local` to mount it on boot.

```
echo 'adding encrypted partition'
bioctl -c C -l /dev/sd0a softraid0 && \
  fsck /dev/sd1a && \
  mount /home
```

2 Anonymous CVS Access over SSH

I like to use CVS because it's simple, centralized, and gives me access to the raw changesets in a nice plain-text format. If I want to go in and edit a commit message, or move some stuff around, I just do it. It's great for a single person or small disciplined group. But one problem with CVS is anonymous access: although mechanisms such as `pserver` exist, they have been plagued with bugs historically. The most common way to access CVS is over SSH, so is it possible to provide some kind of anonymous CVS access through SSH? Of course it is, and OpenBSD makes it not only fairly simple to set up, but also provides the kinds of security assurances we need when doing something which is so fraught with peril.

First we need to create our anonymous user; typically the name "anon-cvs" is used. We need to give them a legitimate shell and home directory because even though they will only be able to execute `cvs server` the SSH process gives them a shell first and plunks them down in their home directory. Note also that I put them in the `cvs` group, which I've reserved for anonymous CVS access.

```
# useradd -d /cvs -s /bin/sh -g cvs anoncvs
```

Now that we have a user, we need to configure their SSH access using some special tunings which vary from normal SSH user access. In my case, I need to enable password authentication (which is normally disabled). Of

course we want to inhibit them from using any command other than `cvs` server so we adjust that as well. We also want to chroot them to a particular location when they log in, to mitigate any security problems that may occur with the `cvs` server command. Add this block to the bottom of your `sshd_config` file.

```
Match User anoncvs
    ChrootDirectory /cvs
    MaxAuthTries 2
    X11Forwarding no
    AllowTcpForwarding no
    ForceCommand cvs server
    PasswordAuthentication yes
    PermitEmptyPasswords yes
```

The reason we have to set `MaxAuthTries` to 2 (I have my default set to 1) is that some clients seem to always try the “none” method before “password,” which will cause the SSH authentication to fail immediately (one of one allowed attempts failed). Now we’ve also set the chroot directory to be `/cvs` which is what we’ve given the home directory as, but the way SSH chroot works is that the user is logged in, the chroot is performed, and then the user’s shell is started in their home directory within the chroot. So effectively the user’s home directory will be `/cvs/cvs` which is where we will put the CVS repository.

Because I’m extra paranoid I’m not going to allow the anonymous user to even access the repository directly though. Instead, I’m going to reuse the same trick I do to allow CVS access via `cvsweb` in the chrooted web server: with a local read-only NFS mount. In your `/etc/exports` file place this:

```
/var/cvs/public -ro
```

using the location of your public CVS repository. Then in `/etc/fstab` we will use this:

```
localhost:/var/cvs/public /cvs/cvs nfs ro 1 0
```

Now we’re ready to configure the chroot environment, by adding everything that we need to run in it. Obviously we’ll need the `cvs` command, so run `ldd` on that to figure out the libraries it needs:

```
# ldd /usr/bin/cvs
/usr/bin/cvs:
    Start      End          Type Open Ref GrpRef Name
    1c000000   3c01f000   exe  1    0    0    /usr/bin/cvs
    0b1b8000   2b1c0000   rlib 0    1    0    /usr/lib/libz.so.4.1
    0aeb8000   2aebd000   rlib 0    1    0    /usr/lib/libgssapi.so.5.0
    0407d000   2408d000   rlib 0    1    0    /usr/lib/libkrb5.so.17.0
    00f8e000   20fce000   rlib 0    1    0    /usr/lib/libcrypto.so.18.0
    038f0000   238f5000   rlib 0    1    0    /usr/lib/libdes.so.9.0
    02175000   221ae000   rlib 0    1    0    /usr/lib/libc.so.51.0
    08d7e000   08d7e000   rtld 0    1    0    /usr/libexec/ld.so
```

Copy these files into their appropriate places within */cvs*. You'll also need */bin/sh* and some basic devices. Copy *MAKEDEV* from */dev* and place it in */cvs/dev*, then run `./MAKEDEV std` to create the basic devices that most programs need. You can then remove the *MAKEDEV* script for security, since you won't be needing it anymore.

One more standard directory needs to be created, which is pretty easy to overlook: */tmp*. Create this inside the chroot and make it `chmod 1777`. The *cvs* program needs this for reasons I'm not certain of.

The last important step in setting up your chroot directory is to create the location where the read locks will be put during checkout. CVS locks directories for reading to prevent people from changing a directory while you're checking it out into your sandbox. We're going to use */var/lock/cvs*, which means that we need to create this both outside the chroot (for normal usage) and inside the chroot (for anonymous checkout). In both cases, it needs to be owned by somebody, I picked root, and writable by the *cvs* group. Set the `LockDir` option in the *CVSROOT/config* file to make *cvs* start using it.

And now you should be good to go. Make sure you sent *sshd* a `SIGHUP` to get it to reload its configuration, and try this from some other machine:

```
$ cvs -d anoncvs@host:/cvs co -d test .
```

Also make sure you try checking in, and checking out the real CVS directory (in my case this was */var/cvs/public*) to ensure that they fail.