Securing CVS on OpenBSD

Tillman J. Hodgson
Regina, Saskatchewan, Canada
http://www.hodgsonhouse.com/tillman/opensource.html

Revision Date: May 10, 2001

Contents

1 Introduction **1**
- 1.1 Foreword . . . 1
- 1.2 Disclaimer . . . 1
- 1.3 Copyright . . . 1
- 1.4 Acknowledgements . . . 2
- 1.5 What is CVS? . . . 2

2 Methodology **2**
- 2.1 The Importance of Secure Passwords . . . 2
- 2.2 The Importance of Secure Users . . . 3
- 2.3 SSH Tunnels . . . 3
- 2.4 Chroot Jails . . . 3

3 Chrooting CVS **3**
- 3.1 CVS User Setup . . . 3
- 3.2 Populating the Jail . . . 3
- 3.3 The run-cvs Wrapper . . . 3
- 3.4 CVS Tools . . . 4
- 3.5 CVS Administrative Files . . . 4
- 3.6 Notifications . . . 4
- 3.7 Sending Mail out of the Jail . . . 4

4 SCVS: SSH Tunnel **4**
- 4.1 Where to get SCVS . . . 4
- 4.2 How to modify SCVS . . . 4
- 4.3 Disabling SCP . . . 4

5 Users Manual: Basic Operations **4**

A Resources **4**

B scvs listing **4**

C Change Log **6**

D GNU Free Documentation License **6**

1 Introduction

1.1 Foreword

This document is meant as a basic introduction to securing an Internet-accessible CVS repository on an OpenBSD server. It was prepared by Tillman Hodgson after his experience putting together the public CVS server for LOSURS (`http://www.losurs.org`). OpenBSD 2.8 with the -PATCHES CVS branch applied was used while developing this document, though these techniques should be reasonably portable to most ∗NIX systems.

1.2 Disclaimer

The author of this document is *not responsible* for any damages incurred due to actions taken based on this document. CVS has traditionally been a very insecure service, especially when exposed to the Internet. If you do not feel comfortable taking responsibility for your own actions, you should stop reading this document and hire a qualified security professional to handle your CVS security for you.

1.3 Copyright

Please contact Tillman Hodgson if you wish to use this document under other terms or circumstances.

1.4 Acknowledgements

There's a few folks that the author would like to acknowledge for their aid in putting together this document:

- Anton Berezin for writing the FreeBSD-based *Chrooted tunnelled read-write CVS server HOWTO*[1], which is based on other works which also deserve a moment of appreciation.
- Heikki Korpela (from the `misc@openbsd.org` mailing list) for his many emails of assistance while I was tracking down the email notifiations problems. Cheers :-)
- Scott Wunsch (`http://www.wunsch.org`) and Gord Matzigkeit (`http://www.fig.org`), LOSURS founder and member-extraordinaire respectively, for helping me set up the original presentation (and not heckling me too much) and for letting me bounce ideas off of you at early hours of the morning.

1.5 What is CVS?

The *Cederqvist*[2], the official manual for CVS, has this to say about CVS and its uses:

> "CVS is a version control system. Using it, you can record the history of your source files.
>
> For example, bugs sometimes creep in when software is modified, and you might not detect the bug until a long time after you make the modification. With CVS, you can easily retrieve old versions to see exactly which change caused the bug. This can sometimes be a big help.
>
> You could of course save every version of every file you have ever created. This would however waste an enormous amount of disk space. CVS stores all the versions of a file in a single file in a clever way that only stores the differences between versions.
>
> CVS also helps you if you are part of a group of people working on the same project. It is all too easy to overwrite each others' changes unless you are extremely careful. Some editors, like GNU Emacs, try to make sure that the same file is never modified by two people at the same time. Unfortunately, if someone is using another editor, that safeguard will not work. CVS solves this problem by insulating the different developers from each other. Every developer works in his own directory, and CVS merges the work when each developer is done."

CVS is not:

- a build system
- a substitute for management
- a substitute fpr developer communications
- a change control system
- an automated testing program
- a process management system

CVS was also originally designed to provide version control for developers working off of the the same development server. The grafting on off network access has resulted in a reputation of bad security, and was the motivation for this document.

2 Methodology

When trying to secure a service for public access, a little bit of planning for your defensive strategy goes a long way towards building a more solid system. What you want to do is identify all the possible attack vectors, and then design your security to block them.

Naturally, if your methodology misses an attack vector, you are vulnerable.

Layered defenses help to alleviate this. By assuming that any one defense method is somehow still vulnerable and designing your security to offer redundent defense mechanisms you can often survive an exploit relatively unscathed. The methodology that was followed in this document works on a layered system of a strong secure password stance, encrypted tunnels to the CVS repository via SSH, and a "jailed" CVS environment to contain any intrusions. The appropriate use of the standard Unix permissions and host security also plays a role, but is not covered in this document.

2.1 The Importance of Secure Passwords

Passwords are the keys to your system.Having a username/password combination allows an attacker to bypass all your security and obtain direction access. Thus, using passwords that are strong enough to prevent casual cracking are an effective and simple measure that *needs* to be part of the overall security plan.

The book *Practical UNIX & Internet Security*[3] has an in-depth discussion on password selection. They recommend passwords that:

- Have both uppercase and lowercase letters.
- Have digits and/or punctuation characters as well as letters.
- May include some control characters and/or spaces.

- Are easy to remember, so they do not have to be written down.
- Are seven or eight characters long.
- Can be typed quickly, so somebody cannot determine what you type by watching over your shoulder.

OpenBSD's `crypt()` function allows us to tighten password security considerably. By using Blowfish to encrypt CVS passwords[1] you can allow the use of longer passwords (up to 72 characters) and allow characters that traditional `crypt()` may have interpreted rathering than accepting at face value. This, combined with Blowfish greater resilience to cracking attempts, should greater increase the security of your CVS passwd file.

† *Note that you should still change all passwords in your passwd file if you even suspect that it has fallen into unfriendly hands. The use of good passwords and the Blowfish encryption method buys you more time to do this in, but does* not *provide a guarantee that some passwords won't be cracked.*.

2.2 The Importance of Secure Users

CVS has the convienient, but possibly insecure, feature of saving the users CVS password in `~/.cvspass` in essentially plain-text form[2].

The implications of this are that if a users home directory (and, by extension, their workstation in general) is not well secured, then obtaining the pasword is fairly easy.

Tools that you can use to combat this include:

- Frequent password changes by the CVS administrator, with the new password being transmitted to the uers non-electronically
- Frequent backups of the CVS repository so that damage can be repaired
- Not giving remote users the ability to modify CVSROOT[3]

Additionally, some of the options for the `admin` command are dangerous. It's use should be restricted by creating the `cvsadmin` user group, which prevents regular users from using all of the `admin` options except `-k`, which is needed to mark files as binary.

[1] That explanation is somewhat simplified. The actual passwd field entry is created by encrypting the string "OrpheanBeholderScryDoubt" with the Blowfish state 64 times.

[2] The password is mangled to make it harder to read, but the algorythm is well-known and reversible.

[3] This is inconvienient, as the CVS administrator has to do a lot of work on behalf of the developers, but can prevent an intruder from causing as much damage.

2.3 SSH Tunnels

2.4 Chroot Jails

The security of a network daemon (such as CVS) can be enhanced by using the `chroot()` system call. `chroot()` changes the effective root directory for a process to a specific subdirectory within the filesystem. For example, the CVS process would see `/chroot/cvs/` and it's subdirectories as `/` and it's subdirectories.

This greatly improves security because a very reduced piece of the system is available to the intruder if they successfully infiltrate the CVS daemon.

For example, the real `/etc/master.passwd` file is not viewable from within the chroot "jail", and instead a substitute (with only the CVS user) at `/chroot/cvs/etc/master.passwd` takes its place.

Under most Unix variants if the root user is able to execute arbitrary code (for example, if `perl` is available or a method to move new executables into the chrooted environment is utilitzed) they will be able to break out of the chroot.[4]

Carefully constructing your chroot environment so that the root account is well protected helps alleviate one of these concerns. Unfortunately, bringing executable code (precompiled static binaries, for example) into the chroot is very easy with a CVS daemon, since one of its functions is to hold a file repository. This requires us to emphasize even more securing the root account.

Because the CVS daemon will not be able to access anything outside of the chroot environment special measures must be taken to ensure that any needed devices are available (i.e. that `/chroot/cvs/dev` exists and has appropriate entries)

3 Chrooting CVS

3.1 CVS User Setup

3.2 Populating the Jail

Statically compiled binaries recommended because ...

3.3 The run-cvs Wrapper

vim run-cvs.c gcc -O2 run-cvs.c -o run-cvs cp run-cvs /usr/sbin/run-cvs

```
#include <stdlib.h>
#include <unistd.h>

/* change these #defines to suit your setup */
#define BASE "/chroot/cvs"
#define OWNER_UID 999
#define OWNER_GID 999

int main(int argc, char *argv[])
```

[4] For information on breaking out of a chrooted environment, see http://www.bpfh.net/simes/computing/chroot-break.html

```
{
  int res;

  res = chdir(BASE);
  if ( res ) exit(1);

  res = chroot(BASE);
  if ( res ) exit(2);

  res = setgid(OWNER_GID);
  if ( res ) exit(3);

  res = setuid(OWNER_UID);
  if ( res ) exit(4);

  /* there should be --allow-root string per
     repository you are allowing access to */
  execl("/bin/cvs", "cvs",
    "--allow-root=/TILLMAN",
    "--allow-root=/LOSURS",
    "--allow-root=/TOMGOULET",
    "pserver",
    NULL);
  exit(3);
}
```

† *Remember to modify* `run-cvs.c`*, re-compile, and* `cp` *it over the existing copy every time that you add a new repository or else the new repository will not be accessible via cvs.*

BlahBlahBlah ...

3.4 CVS Tools

3.5 CVS Administrative Files

3.6 Notifications

3.7 Sending Mail out of the Jail

4 SCVS: SSH Tunnel

4.1 Where to get SCVS

4.2 How to modify SCVS

The first line of the `scvs` file contains the full path to `perl`. If your copy of `perl` is installed in a different location you'll have to modify this line to match.

† *Note that you need to modify this for every client installation.* `scvs` *is not installed on the server, just on the clients. Thus you'll likely end up customizing each client installation individually unless you have identical clients.*

The next bit of `scvs` is intended to be modified by the user, and looks like this:

```
#--
#   tunable variables
#--

$tune_cvs_server_name = "cvs.losurs.org";
$tune_local_cvs_cmd = "/usr/bin/cvs";
$tune_remote_cvs_port = 2401;
$tune_local_cvs_port = 2401;
# local ssh command to use
$tune_ssh_cmd = "ssh";

# the user on the server side cvs runs as
$tune_ssh_user = "cvs";
```

4.3 Disabling SCP

5 Users Manual: Basic Operations

A Resources

B scvs listing

The complete `scvs` listing is presented here, with comments stripped after the configuration section (excepting copyright notice). This in provided in case the reader is unable to find a working mirror of `scvs` on the Internet.

```
         1         2         3         4         5
12345678901234567890123456789012345678901234567890123456

#!/usr/bin/perl

# modified for ssh+chroot setup by Anton Berezin <tobez@plab.ku.dk>

#--
#   tunable variables
#--

# change this!
$tune_cvs_server_name = "loki.hodgsonhouse.com";

# where cvs program is located on the client system
$tune_local_cvs_cmd = "/usr/bin/cvs";

# remote pserver port to use
# for explanation, see
#    http://www.prima.eu.org/tobez/cvs-howto.html#inetd
#$tune_remote_cvs_port = 2410;
$tune_remote_cvs_port = 2401;

# local pserver port;  you probably don't want to change this
$tune_local_cvs_port = 2401;

# ssh command to use; the default is good for UNIX clients
$tune_ssh_cmd = "ssh";

# the user on the server side cvs runs as
$tune_ssh_user = "cvs";

#--
#   end of tunable variables
#   there is no need to modify anything below
#--

#
# $Id: scvs,v 1.6 1999/02/09 16:37:04 tim Exp $
#
# (c) 1999, Tim Hemel <tim@n2it.net>
#
# SCVS - "secure cvs"
#
# scvs [ -d cvsroot ] [ cvsoptions ] cvscommand [ cvscommandoptions ]
#
# This program executes a cvs client and lets it run its traffic thro
# encrypted SSH tunnel.
#
# The remote repository can be specified on the commandline, or in th
# environment variable. This should be done with -d cvsroot, where cv
# the remote repository. For example: :pserver:cvs@cvs.n2it.net:/cvs.
# option MUST be the FIRST option to scvs if it is given.
#
# After having checked out a file, the CVS/Root file contains the fak
# cvs server on the localhost. This saves you from using the -d optio
```

```
# time. Be careful however when using both scvs and cvs on the same client.
# Preferably all cvs traffic should be done with scvs.
#
# TODO:
#
# * Implement better error detection and recovery.
# * Better parsing of the repository (it will not detect strange syntaxes)
# * Better command line option parsing
# * Add more options that are now still environment variables. For example:
#   -S should contain $SSH_DEFAULT_HOST. for example: -S ssh@cvs.n2it.net:22,
#   -S ssh@, -S :22, -S cvs.n2it.net, or -S ssh@:22.
#
#
# Note: there is no way to specify a different port number to cvs. This means
# that all scvs clients on the same machine need to share port 2401.
# This is possible, only the ssh tunneling will fail, so our program should
# detect that and continue anyway.
# A successful connection is then only possible if the other users know the
# password for $SSH_USER@$SSH_HOST, or if there is no password.
#

##############################################################################
# CVS settings

$CVS_CMD=$tune_local_cvs_cmd;

$CVS_PORT=$tune_remote_cvs_port;
$CVS_LOCAL_PORT=$tune_local_cvs_port;

# the values below will be needed only if the repository is specified via the
# command line or the CVSROOT environment variable, and in those cases $HOST
# will be extracted from there. However, for funny results you can uncomment
# these two lines.
# $CVS_HOST="cvs.n2it.net";
# $CVS_USER="tim";

##############################################################################
# SSH settings

# ssh2 does not seem to work with our -L port:host:hostport argument, so make
# sure we will use ssh1.
$SSH_CMD=$tune_ssh_cmd;

# This should be the user on whose behalf the tunneling is made. It is
# typically a user that cannot do any harm, has no password and uses a program
# like nologin (but one that will wait) as a shell.
$SSH_USER=$tune_ssh_user;

# This value should also automagically be set from the repository string.
# However, if that host is not running sshd, you may want to tunnel through
# another host and modify and uncomment the line below.
# $SSH_HOST=$CVS_HOST

# This value is used if the repository cannot be determined from the
# commandline or the CVSROOT variable. Modify this for your local situation.
$SSH_DEFAULT_HOST=$tune_cvs_server_name;

# Port at which the sshd on the remote server runs. Default is 22.
# $SSH_PORT=22;

# This should be left unmodified, as it makes no sense changing this. Unless
# some future version (or perhaps even the current version) of cvs allows you
# to specify the remote repository's port.
$SSH_LOCAL_PORT=$CVS_LOCAL_PORT;

##############################################################################
# & parse_repository ({{rep}})
# . extracts the method, user, host, port and directory from {{rep}}.
# . There are four possibilities:
#   - /path/to/repository
#   - :method:/path/to/repository
#   - :user@hostname:/path/to/repository
#   - :method:user@hostname:/path/to/repository
#
# . This function is far from perfect and will produce strange results with
#   non-standard repositories. Has only been tested for :pserver:
#
sub parse_repository
{
  my $rep = $_[0];
  my ($dir,$user,$host,$method);

  # determine the directory
  $rep =~ s/:(\/.*)$// && do { $dir=$1; };
  if (not $dir)
    { $rep =~ s/(\/.*)$// && do {$dir = $1; }; }

  # determine the hostname and the username
  $rep =~ s/:([^:]*)@(.*)$// && do { $user = $1; $host = $2; };
  if (not $host)
    { $rep =~ s/:([^:]+)$// && do { $host = $1; }; }

  # all that is left now is the method
  $rep =~ s/^:([^:]*)// && do { $method = $1; };

  # If there is still anything left, we have an error, warn the user
  if ($rep)
    { print STDERR "Warning: repository parsed wrong ('$rep' ignored)\n"; }

  #print STDERR "DEBUG: dir=$dir, user=$user, host=$host, method=$method\n";
  return ($method, $user, $host, $dir);
}

##############################################################################
# main

# should be changed to a general cmdline parsing routine.
# get the repository's name from the commandline or the CVSROOT environment
# variable.
if ($ARGV[0] eq "-d")
  {
  $rep = $ARGV[1];
  shift; shift;
  }
else
  {
    $rep = $ENV{'CVSROOT'};
  }

#print STDERR "DEBUG: rep = $rep\n";
# parse the repository
($method, $user, $host, $dir) = parse_repository $rep;
# print "met: $method, user: $user, host: $host, dir: $dir\n";
# construct the local fake cvs server name.
if ($method) { $cvs_serv = ":$method:"; }
if ($user)   { $cvs_serv .= "$user\@"; }
if ($rep)    { $cvs_serv .= "localhost:"; }
if ($dir)    { $cvs_serv .= $dir; }
# print STDERR "DEBUG: cvs_serv = $cvs_serv\n";

# construct the tunneling command
$SSH_HOST |= $host;
#print STDERR "DEBUG: SSH_HOST=$SSH_HOST\n";
$SSH_HOST |= $SSH_DEFAULT_HOST;
#print STDERR "DEBUG: SSH_HOST=$SSH_HOST\n";
if ($SSH_USER)
  { $ssh_serv="$SSH_USER\@$SSH_HOST"; }
else
  { $ssh_serv="$SSH_HOST"; }

# print "ssh_serv: $ssh_serv\n";

$tunnel_cmd = "$SSH_CMD $ssh_serv -q -x -f"
            . ( ($SSH_PORT) ? " -p $SSH_PORT" : "" )
            . " -L $SSH_LOCAL_PORT:$SSH_HOST:$CVS_PORT open";
# print "tunnel_cmd: $tunnel_cmd\n";

# execute the tunneling, and read the response from the server
open (TUNNELSH,"$tunnel_cmd |") or die "Could not execute $tunnel_cmd\n";
chomp ($magicword = <TUNNELSH>);
#print STDERR "magicword = $magicword!\n";
# now we can call system to execute the cvs command.
  # print STDERR "Doing: $CVS_CMD|", ( ($cvs_serv) ? ('-d |', "$cvs_s
$exitcode =
  system "$CVS_CMD", ( ($cvs_serv) ? ('-d', $cvs_serv) : () ) , @ARGV
```

```
220 if ($exitcode) { print STDERR "Could not execute CVS command!\n"; }

    # close the tunnel
    #print STDERR "DEBUG: ", ( "$SSH_CMD $ssh_serv -q -x -f"
224 #            . ( ($SSH_PORT) ? " -p $SSH_PORT" : "" )
    #            . " $magicword" );
    # print STDERR "before close\n";
    system ( "$SSH_CMD $ssh_serv -q -x -f"
228             . ( ($SSH_PORT) ? " -p $SSH_PORT" : "" )
                . " $magicword" );

    # print STDERR "right after close\n";
```

C Change Log

D GNU Free Documentation License

References

[1] Anton Berezin, *Chrooted tunnelled read-write CVS server HOWTO*, http://www.prima.eu.org/tobez/cvs-howto.html

[2] Per Cederqvist et al, *The Cederqvist*, http://cvs.home.org/docs/manual/cvs.html

[3] Simson Garfinkel & Gene Spafford, *Practical UNIX & Internel Security*, ISBN 1-56592-148-8