Global Information Assurance Certification Paper

SSH Security Shell for Solaris Servers

GIAC Security Essentials Certification Practical

Philip J. Mannino, Jr.

Version 1.4b (Option 2)
Submitted April 25, 2003

Table of Contents

Abstract

This paper is a case study in upgrading system security. The purpose of the upgrade was to meet management requirements for more stringent data protection and to provide better security for the end users in our network. After reviewing our environment we came up with specific changes that would make our security criteria stronger. Our workstations run Windows '98 and Windows XP, while our servers consist of Solaris 2.8 (Unix), Windows NT 4.0 and Windows 2000. In examining the operating systems for potential security holes, we concluded that certain Unix service programs (TELNET, RSH, RLOGIN, and FTP) were very insecure and would have to be replaced. The scope of this paper is limited to the effort to replace these service programs while continuing to allow Windows' users to access Unix servers.

Our research for a software replacement for these programs led us to Secure Shell (SSH). We evaluated various implementations of SSH and arrived at a commercial product from SSH Communications Security Corp., URL: http://www.ssh.com, to replace the service programs. This paper contains a detailed description of the functions inherent in SSH that made it a suitable alternative. The planning and installation of version 3.5p1 of OpenSSH are described. Additional steps of our effort are also elaborated, including choosing complimentary client software, configuring and customizing SSH, and troubleshooting problems encountered in the installation process. I have in the summary provided the reader with some pros and cons to our implementation of SSH.

The Environment

We have a typical Unix Solaris 2.8 and Windows NT 4.0/2000 environment. We have several development, test, production and standby Unix servers and over 1200 users. All changes made to our servers are made in off-hours and during scheduled maintenance time. The changes are staged first to our least significant servers, and, then, to the server which is at the next higher level of importance. To complete changes on all servers can take two or three maintenance cycles. The only exceptions are emergency security patches. These are installed sooner on all systems due to the critical nature of the patch.

Like many other companies we were first a mainframe shop and converted to a Sun Solaris shop. Most of our conversion effort was migrating applications and making sure they worked correctly. Initially, security was not our biggest concern because our Unix systems are secure. We have firewalls, intrusion detection systems, and a good password policy. Only our system administrators are allowed on the production and standby servers. Transferring files between Solaris servers or moving data between a Solaris server and other platforms was still vulnerable to interception or corruption. Transferring files and logging on

from one system to the next was easy using FTP, TELNET or one of many other packages.

The Problem and the Solution

Our management raised the priority on security. To bring our systems security up to the level Management required we determined that we would have to disable FTP, RLOGIN, RSH AND TELNET, which have big security holes. However, they are all service programs that our users depended on heavily to support their application environments. Consequently, we needed a security product to replace these functions while simultaneously maintaining equivalent service to our users. Our research found that SSH was the best choice to replace the offending service programs and provide equivalent service to our users. SSH could be installed and tested in parallel with the production environment to allow the users simultaneous access to both the old service programs and the SSH replacements. After parallel testing, if we discovered any major problem with SSH (after we disabled), we could temporarily enable the service programs until the problem was resolved.

SSH, replaces TELNET, RSH, and RLOGIN. Tatu Ylönen developed the original SSH in 1995, and Ylönen also founded SSH Communication Security, Ltd. Transparent to the user, SSH automatically encrypts and decrypts the information sent between the sender and the recipient and guarantees the integrity of the data. SSH cannot stop break-in attempts or denial-of-service attacks, but it can give you the assurance that an attacker cannot get your USERID and password or corrupt your data. SSH includes both client and server products for several different platforms, e.g. Solaris, Linux and Windows.

Basically SSH allows a user to log on remotely to a computer and transfer data between the local computer (client) and the remote computer (server) over a secure connection that SSH establishes. By providing strong authentication and cryptography SSH creates a secure connection for communicating over an unsecured channel. Once a server is contacted and authentication checked and accepted, a secure connection is established, and all transmitted data is encrypted including the USERID and password.

SSH consists of three components: the SSH protocol, the SSH server programs, and the SSH client programs. The SSH protocol takes care of:

Authentication – making sure you are who you say you are;

Encryption – encrypting and decrypting the data passed over the network using secure keys;

Integrity – assuring the data arrives unaltered.

To expand on the basics let us look at the details of establishing a secure connection. When a client initially contacts the server, the client and server disclose the SSH protocol versions that they each support. Once the protocol version is decided, they both switch to a packet-based protocol. At this point the data being exchanged is still unencrypted. What happens next depends on which SSH protocol was negotiated, but, basically, keys are exchanged to turn on encryption, to establish the server authentication, and to provide integrity checking.

The keys consist of two parts--a public key and a private key, both generated by the server. The private key is secret, and the public key is not. The first time a client contacts the server the user will be told that authentication of the host cannot be established. An option is presented to the user to continue and accept the new key. A positive user response will allow the authentication to continue so that the server can be added to the user's list of known hosts. Although there is a slight performance degradation due to the overhead of this additional security, the actual process is performed in the background totally transparent to the end user.

Our Process

The first thing we did was install and configure SSH (OpenSSH version 3.4) on our current Solaris servers without disabling any services. At the same time that we installed SSH we had to provide a client/workstation version (Putty release 0.52) for the users, because soon TELNET and RLOGIN would no longer be available. We evaluated several client software products and Putty was very easy to install, use, and seemed to be widely used and well supported. With Putty installed, our users would be able to familiarize themselves with the new software, knowing they could fall back to their old methods if they had any problems. OpenSSH and Putty are freeware and can be downloaded from URLs: http://www.sunfreeware.com and http://www.chiark.greenend.org.uk/~sgtatham/putty/, respectively.

We also planned to install Jumpstart Architecture and Security Scripts (JASS), but we knew that running the JASS script on a production system could cause other problems. JASS is currently known as Solaris Security Toolkit and was developed by Sun's Enterprise Server Products and Professional Services teams to simplify and automate the process of securing Solaris systems. JASS can be downloaded free from Sun at URL: http://wwws.sun.com/software/security/jass/. The JASS script modifies all the configuration files so that no services outside the

general OS services are started. After deciding which services are safe and required, modifications are made to the system configuration files. JASS would facilitate customizing the SSH software which requires changes to the configuration files. The JASS script had a downside that we had to be careful off. Its default setting was to turn off all services and could cause downtime for our users if we missed turning on those services that are required to start the system.

We knew that new hardware upgrades were near, so we planned on installing JASS as each system was moved to the new hardware and upgraded. Pairing the JASS install with the hardware upgrade gave us plenty of system time for testing. We needed time to test the necessary changes to the system configuration files and to test the system security.

Additional Problems

We discovered that there was a production job using FTP (a Perl program using the Perl FTP function) to transfer logs from the production system to the test system. Another group was using a FTP client to transfer data from their Windows workstation to our Unix systems. Until we could replace our Windows FTP client with a secure Windows client we allowed FTP traffic to some of the Unix servers within our firewall. We evaluated several SSH Windows GUI clients and decided on the commercial product SSH Secure Shell, version 3.1.0, from SSH Communications Security Corp., URL: http://www.ssh.com. We selected SSH Secure Shell because it is easy to use and configure. It has a simple drag-and-drop feature for moving files from one Unix server to the next. The workstation upload/download feature is very effective and augmented with both a file transfer client and a session client.

After installing and configuring the SSH Secure Shell GUI client on the workstations that required the ability to transfer files, we disabled the FTP service. There was still a problem with the program that used FTP to transfer the log files. We had the systems administrators manually transfer the logs when needed until the problem was fixed. This procedure allowed for equivalent function for application programmers while FTP services remained disabled during the time we worked on the Perl program problem. The fix was to transfer the log files using secure copy (scp) by setting up a “trust” from the production server to the test server using the USERID for job control. A “trust” between two servers for a particular USERID is like giving the key to your house to a friend. He can enter your house without getting your permission each time (logging in with USERID/password).

To set up the trust, I generated the public and private keys using the production server, and then I cut and pasted the public key to the test server. I checked that the public key on both servers was the same by comparing their fingerprints, which I will explain shortly. It is important to verify that the keys are identical after the cut and paste because the process could have added spaces. A feature of

SSH computes a short value for a key, called a fingerprint, allowing two keys to be easily compared without the lengthy process of a byte-by-byte comparison. Fingerprints are not unique, but the odds of a false match are negligible. Following are the commands I used to generate the keys, and then to verify that they were identical:

Command executed on the production server to generate the key pair:
ssh-keygen -t dsa -f ~/.ssh/id_dsa -N ""

Command executed on the production and test servers to compare fingerprints:
ssh-keygen –l –f ~/.ssh/id_dsa.pub

Concerns

At first we allowed both SSH1 and SSH2 clients to access our server. However, after further research we discovered that SSH1 had some security holes that were closed in SSH2. Based on this research we decided to allow only SSH2 protocol to communicate with our servers. We initially made the changes to restrict SSH1 to one of our test servers. We could then document any changes that our users would have to go through and test the procedures. We felt the users would not have any problems making the necessary changes using the documentation. Their only change is to the Putty software configuration on their workstation computer. At the same time that we would restrict SSH1 protocol we would also upgrade the OpenSSH software to version 3.5p1.

Installing OpenSSH version 3.5p1 and restricting SSH1

As part of our security upgrade project, I successfully installed and customized a new version of OpenSSH. I followed a procedure, which I modified for our requirements from http://www.sunfreeware.com/openssh8.html, written by Steven M. Christensen and Associates, Inc. See Appendix A for the complete text of the procedure. The general steps I used to install OpenSSH were:

- First I downloaded the five required OpenSSH files and the patch for a random number generator, which is needed to install OpenSSH and does not come with Solaris 8.

- The random number generator package had to be installed first. Then the system was rebooted using the command, “init 0”, and then, “boot –r”, to build the device table with the random number generator before continuing with the normal boot process.

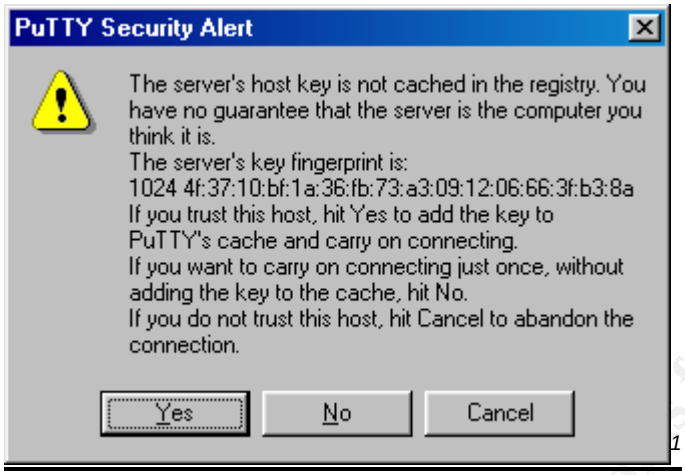
- Next I unzipped and installed OpenSSH packages.

- Once the OpenSSH packages were added I created the sshd user and the /var/empty directory to turn on the privilege separation function. Also I created the /etc/hosts.deny file containing the line sshd:ALL and the /etc/hosts.allow file with a list of servers/workstations which can issue the ssh command to this server.

- Next I generated three key pairs for the server. Messages from re-generating the keys for the new release should look like:

```
# /usr/local/bin/ssh-keygen -t rsa1 -f /usr/local/etc/ssh_host_key -N ""
Generating public/private rsa1 key pair.
/usr/local/etc/ssh_host_key already exists.
Overwrite (y/n)? y
Your identification has been saved in /usr/local/etc/ssh_host_key.
Your public key has been saved in /usr/local/etc/ssh_host_key.pub.
The key fingerprint is:
4f:37:10:bf:1a:36:fb:73:a3:09:12:06:66:3f:b3:8a root@myserver
# /usr/local/bin/ssh-keygen -t dsa -f /usr/local/etc/ssh_host_dsa_key -N ""
Generating public/private dsa key pair.
/usr/local/etc/ssh_host_dsa_key already exists.
Overwrite (y/n)? y
Your identification has been saved in /usr/local/etc/ssh_host_dsa_key.
Your public key has been saved in /usr/local/etc/ssh_host_dsa_key.pub.
The key fingerprint is:
96:84:8a:8b:42:e7:c0:0f:2b:83:72:19:cb:37:d6:57 root@myserver
# /usr/local/bin/ssh-keygen -t rsa -f /usr/local/etc/ssh_host_rsa_key -N ""
Generating public/private rsa key pair.
/usr/local/etc/ssh_host_rsa_key already exists.
Overwrite (y/n)? y
Your identification has been saved in /usr/local/etc/ssh_host_rsa_key.
Your public key has been saved in /usr/local/etc/ssh_host_rsa_key.pub.
The key fingerprint is:
dc:78:b1:64:82:a9:be:b3:d3:66:da:7a:3a:61:f4:b0 root@myserver
#
```

- Users logging on subsequent to regenerating the keys for a server would use Putty and would see the following pop-up during their login process:



[1]

  Note the key fingerprint, 4f:37:10:bf:1a:36:fb:73:a3:09:12:06:66:3f:b3:8a, matches the DSA key fingerprint from the messages in the above the bullet.

- I updated and carefully checked the /usr/local/etc/sshd_config file, making sure the options were correctly set. This is a very important and time-consuming step. One should be careful not to over look options and not to make changes until you are sure you understand the option being changed.

- I created the start up script used to start OpenSSH version 3.5p1 and saved it in file /etc/init.d/opensshd.

- Next, I created a link to the start up directory to start OpenSSH automatically during boot up.

- Finally, I started OpenSSH version 3.5p1. After testing the OpenSSH, I rebooted the server to verify that the OpenSSH start up script worked correctly.

Installation Results

To test the successful installation of Openssh 3.5p1, I logged on to one of our other servers, running our old release of SSH (3.4). I then repeated the process logging on from the SSH 3.4 server to my test server. I used the ssh debug

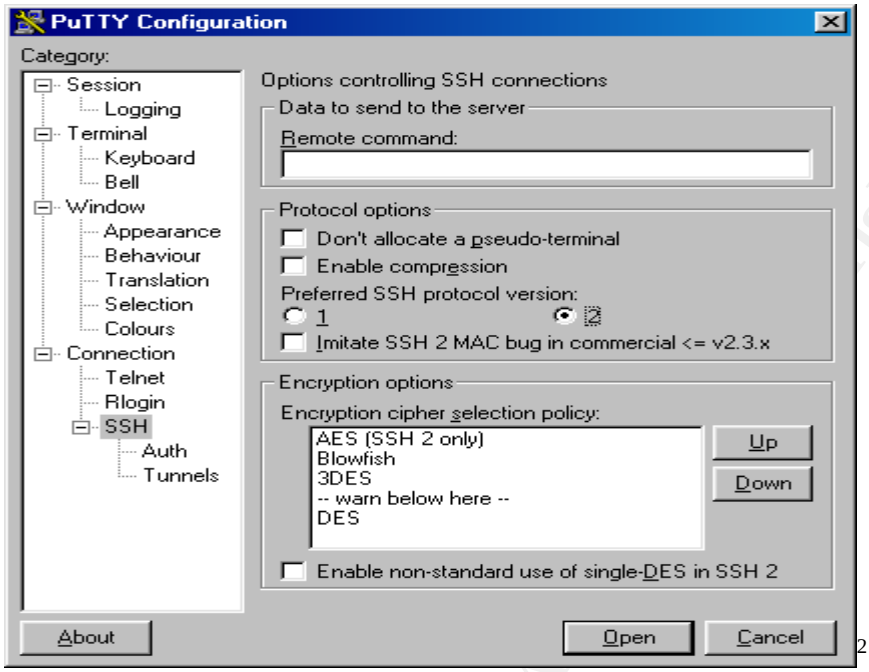
[1] http://www.chiark.greenend.org.uk/~sgtatham/putty/

option, -v, to document the results. The resulting messages from the ssh command follow:

```
myserver% ssh -v othersrv
OpenSSH_3.5p1, SSH protocols 1.5/2.0, OpenSSL 0x0090607f
debug1: Reading configuration data /usr/local/etc/ssh_config
debug1: Rhosts Authentication disabled, originating port will not be trusted.
debug1: ssh_connect: needpriv 0
debug1: Connecting to othersrv [123.456.78.51] port 22.
debug1: Connection established.
debug1: identity file /opt/abcagent/.ssh/id_rsa type -1
debug1: identity file /opt/abcagent/.ssh/id_dsa type -1
debug1: Remote protocol version 1.99, remote software version
OpenSSH_3.4p1
debug1: match: OpenSSH_3.4p1 pat OpenSSH*
debug1: Enabling compatibility mode for protocol 2.0
debug1: Local version string SSH-2.0-OpenSSH_3.5p1
debug1: SSH2_MSG_KEXINIT sent
debug1: SSH2_MSG_KEXINIT received
debug1: kex: server->client aes123-cbc hmac-md5 none
debug1: kex: client->server aes123-cbc hmac-md5 none
debug1: SSH2_MSG_KEX_DH_GEX_REQUEST sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_GROUP
debug1: dh_gen_key: priv key bits set: 122/256
debug1: bits set: 1639/3191
debug1: SSH2_MSG_KEX_DH_GEX_INIT sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_REPLY
The authenticity of host 'othersrv (123.456.78.51)' can't be established.
RSA key fingerprint is 30:cd:d4:2e:cb:52:c0:78:fd:ad:3d:da:0b:23:68:7d.
Are you sure you want to continue connecting (yes/no)?yes
Warning: Permanently added 'othersrv,123.456.78.51' (RSA) to the list of known
hosts.
debug1: bits set: 1634/3191
debug1: ssh_rsa_verify: signature correct
debug1: kex_derive_keys
debug1: newkeys: mode 1
debug1: SSH2_MSG_NEWKEYS sent
debug1: waiting for SSH2_MSG_NEWKEYS
debug1: newkeys: mode 0
debug1: SSH2_MSG_NEWKEYS received
debug1: done: ssh_kex2.
debug1: send SSH2_MSG_SERVICE_REQUEST
debug1: service_accept: ssh-userauth
debug1: got SSH2_MSG_SERVICE_ACCEPT
debug1: authentications that can continue: publickey,password,keyboard-
interactive
```

```
debug1: next auth method to try is publickey
debug1: try privkey: /opt/abcagent/.ssh/id_rsa
debug1: try privkey: /opt/abcagent/.ssh/id_dsa
debug1: next auth method to try is keyboard-interactive
debug1: authentications that can continue: publickey,password,keyboard-
interactive
debug1: next auth method to try is password
abcagent@othersrv's password:
debug1: ssh-userauth2 successful: method password
debug1: channel 0: new [client-session]
debug1: send channel open 0
debug1: Entering interactive session.
debug1: ssh_session2_setup: id 0
debug1: channel request 0: pty-req
debug1: channel request 0: shell
debug1: fd 4 setting TCP_NODELAY
debug1: channel 0: open confirm rwindow 0 rmax 32768
Last login: Mon Apr  7 10:46:09 from myserver.xyz.com
othersrv% exit
othersrv% logout
debug1: channel 0: rcvd eof
debug1: channel 0: output open -> drain
debug1: channel 0: obuf empty
debug1: channel 0: close_write
debug1: channel 0: output drain -> closed
debug1: client_input_channel_req: channel 0 rtype exit-status reply 0
debug1: channel 0: rcvd close
debug1: channel 0: close_read
debug1: channel 0: input open -> closed
debug1: channel 0: almost dead
debug1: channel 0: gc: notify user
debug1: channel 0: gc: user detached
debug1: channel 0: send close
debug1: channel 0: is dead
debug1: channel 0: garbage collecting
debug1: channel_free: channel 0: client-session, nchannels 1
Connection to othersrv closed.
debug1: Transferred: stdin 0, stdout 0, stderr 31 bytes in 39.2 seconds
debug1: Bytes per second: stdin 0.0, stdout 0.0, stderr 0.8
debug1: Exit status 0
myserver%
```

We decided to limit SSH1 access, before users are allowed to connect to the upgraded servers. This policy only involves changing the user's panel, "Putty Configuration." Under, "Connection", "SSH," one selects, "2," for "Preferred SSH protocol version," as shown in the screen image below:



[2]

Over all, I felt the installation on the test server of OpenSSH 3.5p1 was successful. I will document my results for my management. If the change ticket is approved for the production servers, I will install this version of OpenSSH on those other servers during our next maintenance weekend.

Lessons Learned

One concern that I have not yet addressed is the previously mentioned case of a user's first login to the OpenSSH server after the key has been changed. When the client does not find the host, the user is prompted to accept the new key. This process leaves us open to man-in-the-middle attacks. Loading each other's keys between all our Unix systems is manageable, but similar updates to all workstations are not feasible. We might be able to educate our users so they will better understand the processes that create and use keys. At least then they would know what they are accepting whenever they login and get the message that the host key has changed. The same is not true for the system administrator, who should be aware when keys are updated. The new key fingerprint should even be compared the first time an administrator does a logon after the keys have been updated.

[2] http://www.chiark.greenend.org.uk/~sgtatham/putty/

We also noticed that the file transfer throughput rate dropped when we changed to SSH2, which is consistent with documentation on this issue.

Initially we did not try to control the Host Access and Host Deny features because we assumed that our firewall was sufficient. Subsequently, we added a list of servers to be controlled when we upgraded the OpenSSH.

Lastly, it is important to protect all the SSH configuration files. Changes were made to several configuration files during the OpenSSH installation. These files have to have their access restricted and monitored so that we are notified if any are changed. If the SSH configuration files, such as, /etc/host files, are not protected and monitored, the point of upgrading OpenSSH and restricting SSH 1 is lost.

Conclusion

When I started researching SSH I had no idea there was such a vast amount of information on SSH. The more sensitive our data becomes and the larger and more widespread our computer networks grow, the more computer security plays an increasingly important role in our systems.

Adding SSH-2 is just one step in securing our servers, but with each step to provide more security our overall “defense in depth” is enhanced. SSH-2 is the standard for our Solaris servers. To keep them running properly the servers must be configured properly and regularly updated with any new security vulnerability patches. Also, there is a throughput and performance cost to higher security. The transfer rates of FTP are greater than SSH-1 and are much greater than SSH-2, but without data integrity and identity protection, speed does not matter, especially as system attacks get more sophisticated and more destructive. Finally, adding SSH-2 and disabling TELNET, FTP, RLOGIN, and other traditional services adds a great level of protection for remote login and data transfers, but care must also be taken to remove the disabled products to ensure that an attacker cannot enable the replaced services.

Appendix A: Installing OpenSSH version 3.5p1

The following is the procedure from http://www.sunfreeware.com/openssh8.html that I modified for our installation. It was written by Steven M. Christensen and Associates, Inc.

Installing Openssh 3.5p1 packages for Solaris 8

Preparing for installation

- Download from http://www.sunfreeware.com/ :
  - openssh-3.5p1-sol8-sparc-local.gz
  - openssl-0.9.6g-sol8-sparc-local.gz
  - zlib-1.1.4-sol8-sparc-local.gz
  - libgcc-3.2-sol8-sparc-local.gz
  - tcp_wrappers-7.6-sol8-sparc-local.gz
- Openssh requires that each machine have some sort of random (really pseudo-) number generation. Solaris 8 does not come with /dev/random and /dev/urandom built-in, but patches have been released to correct this.
- Creating the random number generator device.
  - Download from http://sunsolve.sun.com/pub-cgi/retrieve.pl :
    - Patch 112438-01
  - Unzip 112438-01.zip
  - Install the 112438-01 patch.
    # cd /opt/patch/112438-01
    # pkgadd -d
  - Bring the system down to OK prompt (init 0)
  - Reboot the system using boot -r.

Installing the packages

- gunzip the packages:
  # /usr/local/bin/gunzip openssh-3.5p1-sol8-sparc-local.gz
  # /usr/local/bin/gunzip openssl-0.9.6g-sol8-sparc-local.gz
  # /usr/local/bin/gunzip zlib-1.1.4-sol8-sparc-local.gz
  # /usr/local/bin/gunzip libgcc-3.2-sol8-sparc-local.gz
  # /usr/local/bin/gunzip tcp_wrappers-7.6-sol8-sparc-local.gz
- Install the packages:
  # /usr/local/bin/pkgadd -d openssh-3.5p1-sol8-sparc-local
  # /usr/local/bin/pkgadd -d openssl-0.9.6g-sol8-sparc-local
  # /usr/local/bin/pkgadd -d zlib-1.1.4-sol8-sparc-local
  # /usr/local/bin/pkgadd -d libgcc-3.2-sol8-sparc-local (if you don't already have gcc 3.2 installed)
  # /usr/local/bin/pkgadd -d tcp_wrappers-7.6-sol8-sparc-local

Setting up the sshd user and the /var/empty directory

- In openssh 3.5p1, a new security method is setup called privilege separation. When privsep is enabled, during the pre-authentication phase

sshd will issue chroot(2) to "/var/empty" and change its privileges to the "sshd" user and its primary group. sshd is a pseudo-account that should not be used by other daemons and must be locked and should contain a "nologin" or invalid shell.[3]

For more information see http://sunfreeware.com/README.privsep.

- Setup sshd user and /var/empty directory
  # mkdir /var/empty (/var/empty should not contain any files)
  # chown root:sys /var/empty
  # chmod 755 /var/empty
  # groupadd sshd
  # useradd -g sshd -c 'sshd privsep' -d /var/empty -s /bin/false sshd

Setting up tcp_wrappers

- Create /etc/hosts.deny and add the follow line:
  sshd:ALL
- Create /etc/hosts.allow and put a list of servers/workstations which can ssh to this server. Something similar to
  sshd:
  123.456.78.129,123.456.78.128,123.456.78.127,123.456.78.126,123.456.78.125

Installing ssh and sshd

- Generate keys for the server
  # ssh-keygen -t rsa1 -f /usr/local/etc/ssh_host_key -N ""
  # ssh-keygen -t dsa -f /usr/local/etc/ssh_host_dsa_key -N ""
  # ssh-keygen -t rsa -f /usr/local/etc/ssh_host_rsa_key -N ""

- Carefully check the /usr/local/etc/sshd_config file and make sure the options are correctly set. The /usr/local/etc/sshd_config should looks like:

  # $OpenBSD: sshd_config,v 1.59 2003/03/25 11:17:16 deraadt Exp $

  # This sshd was compiled with
  PATH=/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/openssh-3.5p1/bin

  # This is the sshd server system-wide configuration file. See sshd(8)
  # for more information.

  Port 22
  Protocol 2
  #Protocol 2,1
  #ListenAddress 0.0.0.0

[3] http://sunfreeware.com/README.privsep

```
#ListenAddress ::
HostKey /usr/local/etc/ssh_host_key
HostKey /usr/local/etc/ssh_host_rsa_key
HostKey /usr/local/etc/ssh_host_dsa_key
ServerKeyBits 768
LoginGraceTime 600
KeyRegenerationInterval 3600
PermitRootLogin no
AllowGroups ssh sysadmin
StrictModes yes
UsePrivilegeSeparation yes
RSAAuthentication yes
PubkeyAuthentication yes
#
# Don't read ~/.rhosts and ~/.shosts files
IgnoreRhosts yes
# Uncomment if you don't trust ~/.ssh/known_hosts for
RhostsRSAAuthentication
#IgnoreUserKnownHosts yes
StrictModes yes
#X11Forwarding yes
X11DisplayOffset 10
PrintMotd no
#PrintLastLog no
KeepAlive yes

# Logging
SyslogFacility AUTH
LogLevel INFO
#obsoletes QuietMode and FascistLogging

RhostsAuthentication no
#
# For this to work you will also need host keys in /etc/ssh_known_hosts
RhostsRSAAuthentication no
# similar for protocol version 2
HostbasedAuthentication no
#
RSAAuthentication yes

# To disable tunneled clear text passwords, change to no here!
PasswordAuthentication yes
PermitEmptyPasswords no

# Uncomment to disable s/key passwords
ChallengeResponseAuthentication no
```

```
# Uncomment to enable PAM keyboard-interactive authentication
# Warning: enabling this may bypass the setting of
'PasswordAuthentication'
#PAMAuthenticationViaKbdInt yes

# To change Kerberos options
#KerberosAuthentication no
#KerberosOrLocalPasswd yes
#AFSTokenPassing no
#KerberosTicketCleanup no

# Kerberos TGT Passing does only work with the AFS kaserver
#KerberosTgtPassing yes

#CheckMail yes
UseLogin yes

#MaxStartups 10:30:60
#Banner /etc/issue.net
#ReverseMappingCheck yes

Subsystem       sftp    /usr/local/libexec/sftp-server
```

- Create /etc/init.d/opensshd

```
#!/bin/sh
pid=`/usr/bin/ps -e | /usr/bin/grep sshd | /usr/bin/sed -e 's/^  *//' -e 's/ .*//'`
case $1 in
'start')
/usr/local/sbin/sshd
;;
'stop')
if [ "${pid}" != "" ]
then
/usr/bin/kill ${pid}
fi
;;
*)
echo "usage: /etc/init.d/sshd {start|stop}"
;;
esac
```

- Create a link to /etc/rc3.d

```
cd /etc/rc3.d
ln -s /etc/init.d/opensshd S25opensshd
```

- Start the sshd

```
 /etc/init.d/opensshd start
```

References

Jason Reid and Keith Watson, “Building and Deploying OpenSSH for the Solaris™ Operating Environment”. July 2001.
URL: http://www.sun.com/solutions/blueprints/0701/openSSH.pdf

Gregory, H. Peter, Solaris Security. Prentice-Hall, Upper Saddle River, New Jersey, 2000. Pages 159-179, 233-234, 241-248, 265-269

Daniel J. Barrett and Richard E. Silverman, SSH The Secure Shell. O’Reilly & Associates, Inc., California, February 2001

SANS Institute, “Hardening Solaris Systems”.
URL: http://www.sans.org/resources/hard_solaris.htm

Finland Helsinki, “SSH statement regarding the vulnerability of SSH1 protocol”, November 7, 2001. URL: http://www.ssh.com/company/newsroom/article/210/

Koenig, Thomas, “Ssh Basics - What is ssh”. 6 June 1997.
URL: http://www.dreamwvr.com/ssh-faq/ssh-faq-2.html#ss2.1

Jason Reid, “Configuring OpenSSH for the Solaris™ Operating Environment”, January 2002. URL: http://www.sun.com/solutions/blueprints/0102/configssh.pdf

Terena, “Security and Encryption: SSH”,
URL: http://www.terena.nl/library/gnrt/security/s6.html

SecurityFocus, “Mailing Lists”,
URL: http://online.securityfocus.com/cgi-bin/sfonline/subscribe.pl

SANS Institute, “SANS Newsletter Subscription Service”,
URL: http://www.sans.org/sansnews

Simon Tatham, PuTTY: A Free Win32 Telnet/SSH Client,
URL: http://www.chiark.greenend.org.uk/~sgtatham/putty/

Kimmo Suominen, Getting started with SSH,
URL: http://kimmo.suominen.com/ssh/