Building OpenSSH—Tools and Tradeoffs

Jason Reid, Solaris™ System Test

Sun BluePrints™ OnLine—January 2003

http://www.sun.com/blueprints

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95045 U.S.A.
(650) 960-1300

Part No. 817-1307-11
Revision 08, 5/28/03
Edition: January 2003

Please
Recycle

Building OpenSSH—Tools and Tradeoffs

OpenSSH is a free, BSD style license, implementation of the Secure Shell protocols. OpenSSH is designed for strong authentication, for improved privacy, for secure X11 sessions, and to not trust the network. OpenSSH is developed on and for the OpenBSD operating system by the OpenBSD group. The OpenSSH portability team then transforms the OpenBSD version into the portable version that supports many UNIX™ derived operating systems including the Solaris™ Operating Environment (Solaris OE) and the Linux operating system.

This article updates much of the information in “Building and Deploying OpenSSH for the Solaris Operating Environment,” Sun BluePrints™ Online article, July 2001. This article contains information about gathering the needed components, deciding the compile-time configuration decisions, building the components, and finally assembling OpenSSH. Things change quickly in the open source world, so the versions mentioned in this article might have changed. Use the latest version, and test it in your environment. Despite version changes, the basic build process should remain the same.

Components

OpenSSH requires several components to be present before you can build it. These components must either be installed individually or as part of the Solaris OE. The following lists the components that are needed:

- Solaris OE build machine
- `gzip`
- ANSI C compiler
- Perl

- Zlib
- Entropy source
- OpenSSL
- OpenSSH
- TCP Wrappers (optional)

See “Resources” on page 24 for information on how to obtain the individual software components.

The instructions in this article use software package names and filenames that do not reference the version number of the software packages. Always use the latest versions. The consequence is that the software build process might change in the future. The instructions were written using the following specific versions:

- Solaris 9 OE for SPARC™ processors
- `gzip` 1.3 (included with the Solaris 9 OE)
- Forte Developer 7 C 5.4 (rebranded as Sun™ ONE Studio 7, Complier Collection)
- Perl v5.6.1 (included with the Solaris 9 OE)
- Zlib 1.1.4
- `/dev/urandom` (Solaris 9 OE feature and entropy source)
- PRNGD 0.9.26 (entropy source)
- TCP Wrappers 7.6
- OpenSSL 0.9.6g
- OpenSSH 3.5p1

Consult the installation documentation to prevent build problems.

Before Building OpenSSH

Before you build OpenSSH, you must consider the issues discussed in this section. Compiling code is output intensive. The build and compiler output are not included in the examples.

Using Static Versus Dynamic Libraries

Zlib, OpenSSL, and TCP Wrappers can be built as either static or dynamic libraries. The default is static. The advantage of static libraries is performance and integration. A statically linked binary is faster to start up, and the executable can be installed as a standalone component. It depends on no supporting libraries (other than the required system dynamic libraries such as `libc.so`.) The disadvantage is that changes to a library require replacing and relinking the executable. Dynamic

libraries allow just the library to be replaced and the program restarted. The Solaris OE 8 and 9 versions ship only dynamic libraries for this reason. Unless you foresee the need to replace libraries frequently, use the default of static libraries. This simplifies the configuration, build, and deployment processes.

Install Versus Build Location

OpenSSH requires its components to be installed in the `/usr/local` directory. This is only for building, not installation of OpenSSH. On the deployed machines, OpenSSH can be installed in either the `/opt/OBSDssh` directory, the `/usr/local` directory, or some other location. You must chose the location before building because the location is compiled into the executables.

About $PATH

The component configure scripts expect the programmer (developer) utilities in the `/usr/ccs/bin` tree to be in the `$PATH` environment variable. If `/usr/ucb/bin` is in `$PATH`, it must be after the compilers and the programmer utilities to prevent the wrong `cc` command from being called.

▼ To Add /usr/ccs/bin to the $PATH Variable

1. **Check the** `$PATH` **variable.**

```
$ echo $PATH
/usr/bin:/usr/sbin
```

2. **Add** `/usr/ccs/bin` **to the** `$PATH` **variable.**

```
$ PATH=/usr/ccs/bin:$PATH
$ echo $PATH
/usr/ccs/bin:/usr/bin:/usr/sbin
```

Checking MD5 Hashes and GNU Privacy Guard Signatures

Before you build the software packages, verify that they have been downloaded correctly by either checking their GNU privacy guard (GPG) signature or MD5 hash. If it differs, do not use the package. You can obtain the MD5 software at:

`http://sunsolve.sun.com/md5/md5.tar.Z`

The compressed TAR file contains both SPARC and x86 binaries. Note that the file permissions on the extracted binaries need to be changed to executable. Consult the GNU Privacy Guard documentation for building it and checking signatures.

▼ To Install the MD5 Software

1. **Download the software into the `/tmp` directory.**
2. **Become the superuser, and change the directory to the `/opt` directory.**
3. **Use the `zcat`(1) command to uncompress the TAR file.**

```
# zcat /tmp/md5.tar.Z | tar xvf -
x md5, 0 bytes, 0 tape blocks
x md5/md5-x86, 23452 bytes, 46 tape blocks
x md5/md5-sparc, 23892 bytes, 47 tape blocks
```

4. **Change the permissions and ownership of the files.**

```
# chmod -R 755 /opt/md5
# chown -R root:bin /opt/md5
# exit
```

▼ To Generate an MD5 Hash

1. **Generate the software's hash.**

```
$ /opt/md5/md5-sparc prngd-x.x.x.tar.gz
MD5 (prngd-x.x.x.tar.gz) = f63c06d96d9610619f702e234a660544
```

2. **Use the `cat`(1) command to inspect the distributed hash.**

```
$ cat prngd-x.x.x.tar.gz.md5
MD5(prngd-x.x.x.tar.gz)= f63c06d96d9610619f702e234a660544
```

Component Descriptions

This section contains descriptions of the OpenSSH components.

Solaris OE Build Machine

The build machine needs to have the same base architecture as the targeted deployment machines. The Solaris OE is currently available for two platforms: SPARC and x86. You can check the architecture using the uname(1) command. If you deploy both, you will be required to build OpenSSH twice.

The following table will help you determine which build machine architectures are compatible.

TABLE 1 OpenSSH Compatible Architectures Examples

Build Architecture	Target Architecture	Comments
Ultra 1 SPARC	Netra™ T1 SPARC	These architectures are compatible.
Ultra 1 SPARC	LX-50 x86	These architectures are not compatible.
LX-50 x86	LX-50 x86	These architectures are compatible.
LX-50 x86	Sun Fire™ 15K SPARC	These architectures are not compatible.

Solaris OE Release

You must build OpenSSH on the oldest Solaris OE release that you plan on supporting. Newer releases of the Solaris OE are backwards compatible. This might require that new features not be used to maintain compatibility across releases. Building a single package reduces build-time costs and prevents a wrong package from being installed. You can check the release version with the uname(1) command.

Metaclusters

The build machine needs to have one of the following metaclusters installed:

- `SUNWCprog` (developer metacluster)
- `SUNWCall` (entire Solaris OE distribution)
- `SUNWCXall` (entire Solaris OE distribution plus OEM support)

The programmer utilities located in the `/usr/ccs/bin` directory are required to build OpenSSH, and `/var/sadm/system/admin/CLUSTER` contains the metacluster software installed on the machine. If the metacluster is not one of the above three, the build machine will need to be reinstalled with the correct metacluster.

▼ To Check the Installed Metacluster

- **Use the `cat`(1) machine to check the metacluster software.**

```
$ cat /var/sadm/system/admin/CLUSTER
CLUSTER=SUNWCall
```

Caution – Do not build on the intended deployment machines. This is particularly critical for machines installed with a minimized approach. Building the software requires a compiler and interpreters that could provide leverage for an attacker. Build the software and package it on the build machine, then deploy it to other machines.

Gzip

The component source software packages are distributed in the Gzip format (for example, *package_name*`.tar.gz`). This is a Gzip compressed TAR file. The file must be uncompressed before it can be extracted. Neither `uncompress`(1) nor `unzip`(1) will be able to uncompress the file. Gzip comes with the Solaris 8 and 9 OE releases. For previous releases, you will have to download the Gzip software and build it from the source. Alternatively, prebuilt binaries can be downloaded at:

`http://www.sunfreeware.com/`

▼ To Extract a Gzip Compressed Software Package

In the following example, `foo.tar.gz` is the name of the software package.

- **Use the `gzip`(1) command to uncompress and the `tar`(1) command to extract the file.**

```
$ gzip -dc foo.tar.gz | tar xvf -
x foo/bar, 0 bytes, 0 tape blocks
x foo/bar/ChangeLog, 10963 bytes, 22 tape blocks
x foo/bar/INDEX, 1138 bytes, 3 tape blocks
```

Compilers

An ANSI C compliant compiler is needed to build the various components. Either the Forte C or GNU C compiler will work. Forte C has the advantage of being able to produce more optimized executables, particularly with the relevant flags being used. The optimization flag usage becomes a factor when building the math-intensive OpenSSL cryptographic library. The Forte complier has the disadvantage of being a separate product. Consult your sales representative for more information on obtaining it. The GNU compiler is available free of charge.

Note – Make sure the build system has the appropriate patches applied, particularly the necessary patches for the Forte C compiler, if you are using it.

To build `gcc`, refer to its documentation. To obtain prebuilt versions of `gcc`, go to:

`http://www.sunfreeware.com/`

Perl

The Practical Extraction and Reporting Language (Perl) is needed to configure and install OpenSSL and OpenSSH. Specifically, version five of the language is needed. Perl version five comes with the Solaris 8 and 9 OE releases. For previous releases, you must download it and build from source. To obtain prebuilt binaries, go to:

`http://www.sunfreeware.com/`

Zlib

Zlib is a lossless data-compression library. Optionally, OpenSSH uses it to compress data as it is transmitted and received to reduce bandwidth consumption. Although the feature is optionally used, Zlib is needed for compilation of OpenSSH. Zlib comes with Solaris 8 and 9 OE releases in dynamic library form.

Note – Per Sun Alert 43541, Solaris 8 OE systems should apply the Zlib patch (patch ID 112611 for SPARC and 112612 for x86). The Zlib patch fixes a security bug detailed in CERT Vulnerability VU#368819.

For the Solaris 2.6 and 7 OE releases, to statically link OpenSSH or for minimized machines without the Zlib dynamic libraries, Zlib will need to be built. To build a dynamic Zlib library, consult the documentation.

Note – Do not use versions previous to `zlib-1.1.4` because there is an exploitable vulnerability.

▼ To Build Zlib

- To configure Zlib to use the Forte C compiler:

1. **Change directories to the** `zlib-`*x.x.x* **directory.**
2. **Use the** `env`**(1) command to set the options and execute the** `configure` **script.**

```
$ env CC=cc \
CFLAGS="-xO5 -xdepend -xprefetch -dalign -xlibmil -xunroll=5 " \
./configure
```

Note – If the target machines are without an UltraSPARC II or III processor, omit the `-xprefetch` flag.

---

3. **Use the** `make`**(1S) command to build the Zlib software.**

```
$ make
```

4. **Use the** `make`**(1S) command to test the build.**

```
$ make test
hello world
uncompress(): hello, hello!
gzread(): hello, hello!
gzgets() after gzseek: hello!
inflate(): hello, hello!
large_inflate(): OK
after inflateSync(): hello, hello!
inflate with dictionary: hello, hello!
                *** zlib test OK ***
```

5. **Install the Zlib software by executing the following commands:**

```
$ su
Password: password
# PATH=/usr/ccs/bin:$PATH
# export PATH
# make install
# ls -l /usr/local/lib/libz.a
-rwxr-xr-x   1 root  other     104308 Oct 10 14:03 libz.a
```

- To Configure Zlib to Use the GNU C Compiler:

1. **Change the directory to the** `zlib-`*x.x.x* **directory.**
2. **Execute the** `configure` **script.**

```
$ ./configure
```

3. **Use the** `make`**(1S) command to build the Zlib software.**

```
$ make
```

4. **Use the `make`(1S) command to test the build.**

```
$ make test
hello world
uncompress(): hello, hello!
gzread(): hello, hello!
gzgets() after gzseek: hello!
inflate(): hello, hello!
large_inflate(): OK
after inflateSync(): hello, hello!
inflate with dictionary: hello, hello!
                *** zlib test OK ***
```

5. **Install the Zlib software by executing the following commands:**

```
$ su
Password: password
# PATH=/usr/ccs/bin:$PATH
# export PATH
# make install
# ls -l /usr/local/lib/libz.a
-rwxr-xr-x   1 root  other     104308 Oct 10 14:03 libz.a
```

Entropy Sources

Entropy is the measurement of available randomness. A source of randomness is needed to generate cryptographic keys. The keys cannot be predictable because an attacker would be able to guess the key and break the encryption. The problem is that computers are deterministic machines, so they are very unsuited to the task of random number generation. Computers can only produce pseudo random numbers that are, at best, very close to random. True random numbers can only be generated with hardware measuring stochastic natural phenomena, such as radioactive decay.

Hardware-based random number generators are often expensive and have slow bit rates of entropy production. Instead, software-based pseudo random number generators are used. Randomness is approximated by measuring a series of partially random events such as the timing between key strokes, mouse positioning, or arrival of network packets. All of the entropy is collected into a pool and *stirred* (a mathematical process to improve randomness).

The standard interface for entropy requests is to provide two sources: `random` and `urandom`. The `random` source provides processed entropy from the pool. If the pool is empty or not enough entropy is present to fulfill a request, `random` source will block (wait until completion) until enough entropy becomes available. The `urandom` source provides processed entropy from the pool if available. If not enough entropy is available, a cryptographic hash of the available entropy is returned instead. The `urandom` source will never block.

The `random` source always provides the highest quality of entropy with the performance penalty of requests being nondeterministic. The `urandom` source avoids the penalty by providing lower-quality entropy when the pool is low. The interface can be implemented either by two character pseudo devices, FIFOs, or by UNIX domain sockets.

The criteria for choosing an entropy source for OpenSSH are:

- That the source supports the intended Solaris OE release (2.6, 7, 8, or 9)
- That the source supports either thirty-two bit or sixty-four bit kernel mode
- That the source supports the SPARC and Intel platforms
- That the source is self-contained

The choices for entropy sources are:

- OpenSSH's internal entropy collection
- Kernel-level random number generator
- Entropy gathering daemon
- `ANDIrand`
- `SUNWski`
- Pseudo random number generator daemon

OpenSSH Internal Entropy Collection

The internal entropy collection is the default when no other option is provided when OpenSSH is configured. At the invocation of OpenSSH, entropy is gathered by running user-level commands, such as `ps`(1). OpenSSH will block until enough entropy is gathered. This gives the appearance that OpenSSH has hung, particularly on lightly-loaded systems. Internal entropy gathering is not recommended due to its performance.

Kernel-Level Random Number Generators

Kernel-level random number generators implement the standard entropy interface as two character pseudo devices: `/dev/random` and `/dev/urandom`. A kernel implementation has access to all internal state information such as process context and device driver intrinsics. This provides a larger quantity of and a finer grained source of entropy than user-level sources. The Solaris 9 OE and the Linux operating system provide a kernel-level random number generator. With the Solaris 8 OE, it is provided in a patch (patch ID 112438 for SPARC and 112439 for Intel). Kernel-level random number generators are the recommended entropy source.

`ANDIrand`

`ANDIrand` is a kernel-level random number generator kernel module developed by Andreas Maier. It provides the `/dev/random` and `/dev/urandom` character pseudo devices. This module is not supported by Sun, so it is not recommended for systems requiring Sun support services.

`SUNWski`

`SUNWski` is a user-level daemon for the Solaris 2.6 OE. It provides the `random` entropy source interface as a FIFO special file. It is not available for other Solaris OE releases, so it is not recommended.

Entropy Gathering Daemon (EGD)

The entropy gathering daemon (EGD) is a user-level daemon written in Perl by Brian Warner for GNU Privacy Guard. It provides only the `random` entropy source interface through a UNIX domain socket. This source will block, causing performance problems, so it is not recommended. EGD also requires the installation of `perl`(1), which is not recommended for minimized systems.

Pseudo Random Number Generator Daemon

The pseudo random number generator daemon (PRNGD) is a user-level daemon written in C by Lutz Jaenicke. It provides both the `random` and `urandom` entropy sources through a UNIX domain socket. It conforms to the EGD protocol for entropy requests. PRNGD is recommended for systems without a kernel-level random number generator.

Recommendations

Whenever possible use a kernel-level random number generator. It provides the highest quality of pseudo random numbers, has access to the private state information in the kernel, and is difficult for an attacker to determine the inner state. If you cannot use a kernel-level random number generator, use the PRNGD daemon. The following table contains entrophy recommendations based on the operating environment.

TABLE 2 Entropy Source Recommendations

Solaris OE Release	Source
Solaris 9 OE	`/dev/random`
Solaris 8 OE	`/dev/random` (patch 112438 or 112439)
Solaris 2.6 or 7 OE	PRNGD

Building PRNGD Software

PRNGD must be configured manually because there is no configure script. Configuration and building occur at the same time. PRNGD does not need to be installed on the build machine because it is packaged later for deployment.

▼ To Build PRNGD Using the Forte C Compiler

- For the Solaris 7, 8, or 9 OEs

1. **Change the directory to the** `prngd-`*x.x.x* **directory.**
2. **Use the** `make`**(1S) command to build the software package.**

```
$ make CC=cc CFLAGS="-xO5 -DSOLARIS" SYSLIBS="-lsocket -lnsl"
```

- For the Solaris 2.6 OE

1. **Change the directory to the** `prngd-`*x.x.x* **directory.**
2. **Use the** `make`**(1S) command to build the software package.**

```
$ make CC=cc CFLAGS="-xO5 -KPIC -DSOLARIS26 -D__EXTENSIONS__" \
SYSLIBS="-lsocket -lnsl"
```

▼ To Build PRNGD Using the GNU C Compiler

- For the Solaris 7, 8, or 9 OEs

1. **Change directories to the** `prngd-`*`x.x.x`* **directory.**
2. **Use the** `make`**(1S) command to build the software package.**

```
$ make CC=gcc CFLAGS="-O3 -DSOLARIS" SYSLIBS="-lsocket -lnsl"
```

- For the Solaris 2.6 OE

1. **Change directories to the** `prngd-`*`x.x.x`* **directory.**
2. **Use the** `make`**(1S) command to build the software package.**

```
$ make CC=gcc CFLAGS="-O3 -DSOLARIS26 -D__EXTENSIONS__" \
SYSLIBS="-lsocket -lnsl"
```

TCP Wrappers

TCP wrappers provides limited, connection-oriented host-based firewall functionality with which connections can be denied or accepted based on the originating host. Connection attempts are logged using `syslog`(3C). OpenSSH uses this functionality by linking in the `libwrap` library. TCP wrappers is dependent on the name and IP address information returned by the name services, such as DNS. It cannot stop low-level network-based attacks, such as port scanning, IP spoofing, or denial of service. For those, a packet-based firewall solution such as SunScreen™ software is necessary. The Solaris 9 OE has TCP wrappers integrated into it, package `SFWtcpd`, which is located in the `/usr/sfw` directory. For the Solaris 8 OE, TCP wrappers can be found on the Software Companion CD (starting in the Solaris 8 10/00 release). For the Solaris 2.6 and 7 OE releases, TCP wrappers must be downloaded and built from the source. TCP wrappers is not required to build OpenSSH.

Building TCP Wrappers

This section contains procedures for building the TCP wrappers software.

▼ To Build TCP Wrappers

- For the Forte C Compiler

1. **Change directories to the** `tcp_wrappers_`*x*`.`*x* **directory.**
2. **Use the following command to build the TCP wrappers software.**

```
$ make REAL_DAEMON_DIR=/usr/sbin sunos5 \
STYLE="\"-xO5 -xdepend -xprefetch -dalign -xlibmil -xunroll=5 \""
```

Note – If the target machines are without an UltraSPARC II or III processor, omit the `-xprefetch` flag.

- For the GNU C Compiler

1. **Change directories to the** `tcp_wrappers_`*x*`.`*x* **directory.**
2. **Use the following command to build the TCP wrappers software.**

```
$ make REAL_DAEMON_DIR=/usr/sbin sunos5
```

▼ To Install TCP Wrappers

TCP wrappers does not have an automated install script. OpenSSH requires only two files from the distribution: `libwrap.a` and `tcpd.h`

1. **Become the superuser.**
2. **Copy the** `libwrap.a` **file to the** `/usr/local/lib` **directory.**
3. **Copy the** `tcpd.h` **file to the** `/usr/local/include` **directory.**
4. **Change the ownership and permissions with the following commands.**

```
# chown root:other /usr/local/lib/libwrap.a /usr/local/include/tcpd.h
# chmod 755 /usr/local/lib/libwrap.a /usr/local/include/tcpd.h
```

OpenSSL

OpenSSL is a general purpose cryptographic library that also implements the Secure Sockets Layer (SSL) protocols. This is the component that does all the cryptographic work for OpenSSH.

Note – The OpenSSL library contains patented cryptographic algorithms; however, OpenSSH does not use them. The `README` file lists the patents that might apply. Consult your legal counsel as to whether this is an issue.

The `config` script attempts to build a library optimized for the specific build machine. This is not distributable or portable, particularly if the build machine is not identical to the intended target machines. Instead, use the `Configure` Perl script to build a more general library. In selecting the designated support, choose the lowest common denominator platform.

TABLE 3 OpenSSL Configure Architecture Designations

Supported Architectures	Forte C Compiler	GNU C Compiler
`sun4c`, `sun4d`, `sun4m`, `sun4u`	`solaris-sparcv7-cc`	`solaris-sparcv7-gcc`
`sun4d`, `sun4m`, `sun4u`	`solaris-sparcv8-cc`	`solaris-sparcv8-gcc`
`sun4u`	`solaris-sparcv9-cc`	`solaris-sparcv9-gcc`
`i86pc`	`solaris-x86-cc`	`solaris-x86-gcc`

Note – Avoid all of the designations in the following list.

- Designations that start with `debug`, for example `debug-solaris-sparcv8-gcc`

  This will cause performance problems because it is meant only for debugging problems with the library.

- `solaris64-sparcv9-gcc31` or `solaris64-sparcv9-cc`

  These designations will not link with the 32-bit Zlib and OpenSSH components.

- `solaris-sparc-sc3`

  This compiler is not supported on the Solaris 2.6, 7, 8, or 9 OE releases.

▼ To Build and Test OpenSSL

1. **Change directories to the** `openssl-`*`x.x.x`* **directory.**
2. **Run the** `Configure` **script with the appropriate designation.**

```
$ ./Configure designation
```

3. **Use the** `make`**(1S) command to build and test the OpenSSL software.**

```
$ make
$ make test
```

▼ To Install OpenSSL

1. **Become the superuser.**
2. **Add** `/usr/ccs/bin` **to the** `$PATH` **variable, and export the variable.**

```
# PATH=/usr/ccs/bin:$PATH
# export PATH
```

3. **Use the** `make`**(1S) command to install the software.**

```
# make install
```

OpenSSH

OpenSSH is the OpenBSD group's implementation of the Secure Shell protocols: one and two. It is based on Tatu Ylönen's original Secure Shell implementation. Before building OpenSSH, all of the required and optional components must be built and installed on the build machine.

Configuring OpenSSH

The `configure` script includes many arguments that influence the compilation and installation process. OpenSSH needs to be configured based on the installation targets, compiler choice, and entropy source usage.

▼ To Obtain the List of Arguments in the Configure Script

1. **Change directories to the** `openssh-`*x*`.`*x*`p`*x* **directory.**
2. **Execute the** `configure` **script with the** `-help` **option to obtain the argument list.**

```
$ ./configure -help
```

As a best practice, you should build OpenSSH with the following arguments:

- `--with-pam`

  This argument enables the use of pluggable authentication modules (PAM).

- `--disable-suid-ssh`

  Do not install OpenSSH with the `setuid` bit. This prevents a local `root` compromise if a vulnerability is found with the `ssh` command. The `setuid` bit is only needed for regression to the `rsh` protocol, which is disabled by the following option.

- `--without-rsh`

  Do not regress to the insecure `rsh` protocol if you are unable to connect by using the Secure Shell protocol.

- `--with-lastlog=/var/adm/lastlog`

  Defines the `lastlog` file location for the Solaris OE.

- `--sysconfdir=/etc/openssh`

  This argument establishes the location for the OpenSSH configuration files. Make it a standard location, but avoid `/etc/ssh` to prevent a collision with the Solaris Secure Shell software. The location can also be: `/etc` or `/usr/local/etc`

- `--prefix=/opt/OBSDssh`

  This argument establishes the top-level installation directory. The `/opt/OBSDssh` directory is for package generation. You can also use the `/usr/local` directory. The top-level installation directory is where OpenSSH looks for its various components.

- `--without-privsep-user`

  This argument disables privilege separation due to PAM interactions.

Caution – If you receive a `Privilege separation user does not exist error`, add the `UsePrivilegeSeparation no` entry to the `sshd_config` file. You can prevent this error from occurring by adding the entry to the `sshd_config.out` file before you generate the package.

- `--without-privsep-path`

  This argument also disables privilege separation due to PAM interactions.

- `--with-prngd-socket=/var/run/egd-pool`

  For systems using PRNGD, add this argument. It is the location of the entropy pool socket.

- `--without-prngd`

  For systems using `/dev/random`, without PRNGD, add this argument. Do not use PRNGD.

- `--without-rand-helper`

  For systems using `/dev/random`, add this argument. Do not use the subprocess entropy gatherer.

Note – The `configure` script will report `Random number source: OpenSSL internal ONLY`; disregard this message.

- `--with-tcp-wrappers=/usr/local`

  For TCP wrappers support, add this argument. If you are using the integrated Solaris 9 OE version, use the `/usr/sfw` directory instead of the `/usr/local` directory.

- `--with-cflags="-x05 -xdepend -dalign -xlibmil -xunroll=5 -xprefetch "`

  For the Forte C compiler, add this argument.

Note – If the target machines are without an UltraSPARC II or III processor, omit the `-xprefetch` flag.

▼ To Configure OpenSSH

- For package creation, `/dev/random` usage, and the Forte C compiler

- **Execute the following command with the appropriate flags.**

```
$ ./configure --with-pam --disable-suid-ssh --without-rsh \
--with-lastlog=/var/adm/lastlog --sysconfdir=/etc/openssh  \
--prefix=/opt/OBSDssh --without-privsep-user --without-privsep-path \
--without-prngd --without-rand-helper \
--with-cflags="-xO5 -xdepend -dalign -xlibmil -xunroll=5 -xprefetch "
```

- For package creation, `/dev/random` usage, and the GNU C Compiler

- **Execute the following command with the appropriate flags.**

```
$ ./configure --with-pam --disable-suid-ssh --without-rsh \
--with-lastlog=/var/adm/lastlog --sysconfdir=/etc/openssh  \
--prefix=/opt/OBSDssh --without-privsep-user --without-privsep-path \
--without-prngd --without-rand-helper
```

- For package creation, PRNGD usage, and the Forte C Compiler

- **Execute the following command with the appropriate flags.**

```
$ ./configure --with-pam --disable-suid-ssh --without-rsh \
--with-lastlog=/var/adm/lastlog --sysconfdir=/etc/openssh  \
--prefix=/opt/OBSDssh --without-privsep-user --without-privsep-path \
--with-prngd-socket=/var/run/egd-pool \
--with-cflags="-xO5 -xdepend -dalign -xlibmil -xunroll=5 -xprefetch "
```

- For `/usr/local` installation, PRNGD Usage, and the GNU C Compiler

- **Execute the following command with the appropriate flags.**

```
$ ./configure --with-pam --disable-suid-ssh --without-rsh \
--with-lastlog=/var/adm/lastlog --sysconfdir=/etc/openssh  \
--prefix=/usr/local --without-privsep-user --without-privsep-path \
--with-prngd-socket=/var/run/egd-pool
```

Building OpenSSH

Build OpenSSH by executing the `make`(1S) command, as in the following procedure. Installation is not needed because OpenSSH is packaged later for deployment.

▼ To Build OpenSSH

1. **Change the directory to the** `openssh-`*`x.x`*`px` **directory.**
2. **Execute the** `make` **(1S) command.**

```
$ make
```

Deploying OpenSSH

Efficient distribution requires that all of the OpenSSH components (client, server, configuration files, and documentation) be combined into a single entity, either a TAR file or preferably a System V package. TAR files are easy to create and transfer, but they require manual installation. System V packages are the standardized process for installing, updating, and removing software on the Solaris OE.

The components of the Solaris OE are installed using packages. A package enables automated installation of OpenSSH, as needed or at install time, using the JumpStart™ software technology. The binaries, configuration files, and documentation are copied to their respective locations, and any needed symbolic links are created.

For more information on packages, refer to the *Application Packaging Developer's Guide* at `http://docs.sun.com` and the following manual pages:

- `pkgadd`(1M)
- `pkginfo`(1)
- `pkgmk`(1)
- `pkgparam`(1)
- `pkgproto`(1)
- `pkgtrans`(1)
- `installf`(1M)
- `pkgask`(1M)
- `pkgrm`(1M)
- `removef`(1M)

Packaging OpenSSH

The `makeOpenSSHPackage.ksh` script creates a Solaris OE package. It takes the OpenSSH executables, configuration files, documentation, and optionally, PRNGD's executables and configuration files, and creates the `OBSDssh` package. The script will need some editing based on your local environment. The `makeOpenSSHPackage.ksh` script configures `openssh.server` (the provided `init` script) based on the configuration of `makeOpenSSHPackage.ksh`. The `init` script is included in the generated package. The OpenSSH Tools TAR file containing an `init` script and a packaging script is available from Sun BluePrints Online, Scripts and Tools page at:

`http://www.sun.com/solutions/blueprints/tools/index.html`

The `makeOpenSSHPackage.ksh` script uses `sshd_config.out` and `ssh_config.out` as the default configuration files for the OpenSSH installation. Place your local configuration into these files before generating the package.

The `OBSDssh` package does not generate host keys at install-time. Keys are generated at first boot. This is to prevent WebStart Flash installed hosts from having duplicate host keys.

▼ To Generate the `OBSDssh` Package

1. **Use the `cp`(1) command to copy the `makeOpenSSHPackage.ksh` script to the openssh-*x*.*x*p*x* directory.**

```
$ cp makeOpenSSHPackage.ksh openssh-x.xpx
```

2. **Copy `openssh.server` to the parent directory of `openssh-`*x*`.`*x*`p`*x*.**

```
$ cp openssh.server .
```

3. **Change the directory to the `openssh-x.xpx` directory.**

```
$ cd openssh-x.xpx
```

4. **Use the `ksh`(1) command to make the package.**

```
$ ksh ./makeOpenSSHPackage.ksh
```

5. **Verify the presence of the package.**

```
$ ls *pkg
OBSDssh.pkg
```

MD5 Hashes

For packages that are distributed for downloading, generate and distribute the MD5 hashes.

▼ To Generate the OpenSSH package MD5 Hash

- **Generate the MD5 hash with the following command.**

```
$ /opt/md5/md5-sparc ./OBSDssh.pkg
MD5 (./OBSDssh.pkg) = f63c06d96dcefd919f702e234a660544
```

Solaris Security Toolkit

To facilitate the deployment of OpenSSH, the Solaris™ Security Toolkit software includes a `finish` script, `Finish/install-openssh.fin`, to automate the installation of the `OBSDssh` package. The `finish` script needs to be edited to include the version of OpenSSH, the package location, and the package name. The script will not install OpenSSH onto a Solaris 9 OE system because the Solaris Secure Shell software is provided. The script must be altered to force the installation onto a Solaris OE 9 system.

- For more information, refer to the following Sun BluePrints OnLine articles:
  - "The Solaris™ Security Toolkit – Quick Start: *Updated for Toolkit Version 0.3*"
  - "The Solaris™ Security Toolkit – Release Notes: *Updated for Toolkit Version 0.3*"

- "The Solaris™ Security Toolkit – Installation, Configuration, and Usage Guide: *Updated for Toolkit Version 0.3*"
- "The Solaris™ Security Toolkit – Internals: *Updated for Toolkit Version 0.3*"

Note – For minimized installations of the Solaris OE, the `SUNWcsl` package is required for OpenSSH. The `SUNWzlib` package is also required if the dynamic Zlib library is used to build OpenSSH.

About the Author

Jason Reid (`jason.m.reid@sun.com`) is in the Solaris System Test group. Prior to his present position, he was an SQA engineer in the Developer Tools Group. Before joining Sun, Jason worked at the Purdue University Computing Center as an UNIX systems administrator, while obtaining his B.S. in Computer Science.

Resources

This section contains lists of references you can use to obtain more information about the OpenSSH components.

- Prebuilt versions of the software (except for the Sun™ ONE Studio software): `http://www.sunfreeware.com/`
- GNU compiler collection: `http://www.fsf.org/software/gcc/gcc.html`
- GNU privacy guard: `http://www.gnupg.org/`
- Gzip: `http://www.gzip.org/`
- OpenSSL: `http://www.openssl.org/`
- OpenSSH: `http://www.openssh.com/`
- Perl: `http://www.perl.com/`
- PRNGD: `http://ftp.aet.TU-Cottbus.DE/personen/jaenicke/postfix_tls/prngd.html`
- Sun ONE Studio Compiler Collection (formerly the Forte Compiler Collection): `http://forte.sun.com/s1scc/index.html`
- Solaris Security Toolkit: `http://www.sun.com/security/jass/`

- TCP Wrappers: `ftp://ftp.porcupine.org/pub/security/index.html`
- Zlib: `http://www.gzip.org/zlib/`

References

Computer Systems, Solaris Productization and Marketing, “Building Longevity into Solaris Operating Environment Applications,” Sun BluePrints OnLine, April 2000, at:
`http://www.sun.com/blueprints/0400/s8long.pdf`

Dasan, Vasanthan, Alex Noordergraaf, and Lou Ordorica, “The Solaris Fingerprint Database – A Security Tool for Solaris Operating Environment Files,” Sun BluePrints OnLine, May 2001, at:
`http://www.sun.com/blueprints/0501/Fingerprint.pdf`

Elling, Richard, “Static Performance Tuning,” Sun BluePrints OnLine, May 2000,
`http://www.sun.com/blueprints/0500/sysperfnc.pdf`

Englund, Martin, “Securing Systems With Host-Based Firewalls,” Sun BluePrints OnLine, September 2001, at:
`http://www.sun.com/blueprints/0901/sunscreenlite.pdf`

Lindh, Börje, “Application Performance Optimization,” Sun BluePrints OnLine, March 2002, at:
`http://www.sun.com/blueprints/0302/optimize.pdf`

Howard, John and Alex Noordergraaf, *JumpStart Technology Effective Use in the Solaris Operating Environment*, Sun Microsystems Press, Palo Alto, 2002

Howard, John and Alex Noordergraaf, “WebStart Flash,” Sun BluePrints OnLine, November 2001, at:
`http://www.sun.com/blueprints/1101/webstart.pdf`

McGraw, Gary and John Viega, “Make Your Software Behave: Playing the Numbers,” IBM DeveloperWorks, April 2000, at:
`http://www.ibm.com/software/developer/library/playing/index.html`

McGraw, Gary and John Viega “Make Your Software Behave: Beating the Bias”, IBM DeveloperWorks, April 2000, at:
`http://www.ibm.com/software/developer/library/beating.html`

McGraw, Gary and John Viega, “Make your software behave: Software Strategies,” IBM DeveloperWorks, April 2000, at:
`http://www.ibm.com/software/developer/library/beating.html`

OpenBSD Group, “OpenSSH Frequently Asked Questions,” OpenBSD Group, April 2002, at:
http://www.openssh.com/faq.html

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

Schneier, Bruce, *Applied Cryptography Protocols, Algorithms, and Source Code in C*, 1994, New York, John Wiley & Sons

Solaris Security Toolkit software web site at:
http://www.sun.com/security/jass/

SunSoft™ Developer Engineering, *Solaris Porting Guide: Second Edition*, SunSoft Press, Mountain View, 1995.

Watson, Keith, “The Solaris Operating Environment, OpenSSH, and PRNGs,” Sun Microsystems, Usenix BOF presentation, 2001

Ordering Sun Documents

The SunDocs℠ program provides more than 250 manuals from Sun Microsystems, Inc. If you live in the United States, Canada, Europe, or Japan, you can purchase documentation sets or individual manuals through this program.

Accessing Sun Documentation Online

The docs.sun.com web site enables you to access Sun technical documentation online. You can browse the docs.sun.com archive or search for a specific book title or subject. The URL is http://docs.sun.com/

To reference Sun BluePrints OnLine articles, visit the Sun BluePrints OnLine Web site at: http://www.sun.com/blueprints/online.html