



Integrating the Secure Shell Software

Jason Reid, Solaris™ System Test

Sun BluePrints™ OnLine—May 2003



<http://www.sun.com/blueprints>

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

Part No. 817-2821-10
Revision 03, 5/5/03
Edition: May 2003

Copyright 2003 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, California 95045 U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at <http://www.sun.com/patents> and one or more additional patents or pending patent applications in the U.S. and in other countries.

This document and the product to which it pertains are distributed under licenses restricting their use, copying, distribution, and decompilation. No part of the product or of this document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any.

Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and in other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, Sun BluePrints, WebNFS, and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the US and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

U.S. Government Rights—Commercial use. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the Far and its supplements.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 2003 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, Californie 95045 Etats-Unis. Tous droits réservés.

Sun Microsystems, Inc. a les droits de propriété intellectuels relatants à la technologie qui est décrit dans ce document. En particulier, et sans la limitation, ces droits de propriété intellectuels peuvent inclure un ou plus des brevets américains énumérés à <http://www.sun.com/patents> et un ou les brevets plus supplémentaires ou les applications de brevet en attente dans les Etats-Unis et dans les autres pays.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a.

Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Des parties de ce produit pourront être dérivées des systèmes Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le Sun logo, Sun BluePrints, WebNFS, et Solaris sont des marques de fabrique ou des marques déposées, ou marques de service, de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays.

Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun™ a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.



Please
Recycle



Adobe PostScript

Integrating the Secure Shell Software

Note – This article is the fifth chapter of the Sun BluePrints™ book *Secure Shell in the Enterprise* by Jason Reid, which is scheduled to be available in July 2003 through www.sun.com/books, amazon.com, and Barnes & Noble bookstores.

Secure Shell was designed as a replacement for the Berkeley r-protocols. With the exception of key management, it can be used as a simple drop-in replacement. More advanced features are also offered that may be appropriate to your environment.

This chapter discusses integrating Secure Shell into your environment. It covers replacing `rsh(1)` with `ssh(1)` in scripts, using proxies to bridge disparate networks, limiting privileges with role-based access control (RBAC), and protecting legacy TCP-based applications. After integrating Secure Shell, insecure legacy protocols can be disabled.

Scripting Secure Shell

Automating actions with the Secure Shell client, `ssh(1)`, is mostly a straightforward replacement of either the `rsh(1)` or `rlogin(1)` commands. There are some key differences in how `ssh(1)` performs compared to its counterparts. Additionally, there are the matters of alternate authentication credentials and host keys.

rsh(1) Versus ssh(1)

In basic usage, the `ssh(1)` command is a direct replacement for the `rsh(1)` command. The major difference between them is authentication. If an `ssh(1)` command is issued and a password or passphrase is needed, it will be prompted for, then the command will be executed. In this case, the `rsh(1)` command will fail with a *permission denied* error.

```
$ rsh host -l user cat /etc/passwd
permission denied
$ ssh host -l user cat /etc/passwd
user@host's password: password
root:x:0:1:Super-User:/:/sbin/sh
daemon:x:1:1:/:
bin:x:2:2:/:usr/bin:
sys:x:3:3:/:
adm:x:4:4:Admin:/var/adm:
lp:x:71:8:Line Printer Admin:/usr/spool/lp:
uucp:x:5:5:uucp Admin:/usr/lib/uucp:
nuucp:x:9:9:uucp Admin:/var/spool/uucppublic:/usr/lib/uucp/uucico
smmsp:x:25:25:SendMail Message Submission Program:/:
listen:x:37:4:Network Admin:/usr/net/nls:
nobody:x:60001:60001:Nobody:/:
noaccess:x:60002:60002:No Access User:/:
nobody4:x:65534:65534:SunOS 4.x Nobody:/:
```

For background jobs, `ssh(1)` also supports the `-n` option to set standard input to `/dev/null`. Alternately, `-f` sets standard input to `/dev/null` after password or passphrase requests, but before command execution. If no remote execution is required and only port forwarding requested, the `-N` option can be used (Protocol 2 only).

rsh(1) Versus scp(1)

The `rsh(1)` command has the same authentication problem as the `rsh(1)` command. As with `ssh(1)`, the `scp(1)` command will prompt for passphrases and passwords as needed. Unlike the `rsh(1)` command, `scp(1)` displays a progress meter as it copies the file. This can be disabled with the `-q` option. The `scp(1)` command can also optionally compress the data stream using the `-C` option.

The following example shows the `rcp` authentication problem and `scp` command's progress meter.

```
$ rcp user@host:/etc/passwd /tmp
permission denied
$ scp user@host:/etc/passwd /tmp
user@host's password: password
passwd 100% |*****| 1044 00:00
```

telnet(1) Versus ssh(1)

The `telnet(1)` command is occasionally used to automate connections to systems in situations in which the `rlogin(1)` and `rsh(1)` commands cannot be used. Automating a `telnet(1)` connection requires the script to pass the login, password, and command to the `telnet(1)` command to execute. The following is a Korn shell script fragment that automates the `telnet(1)` session:

```
(
  sleep 2
  echo login^M
  sleep 2
  echo password^M
  sleep 2
  echo command^M
  sleep 2
) | telnet hostname
```

Unlike `telnet(1)`, this will not work with `ssh(1)`. The Secure Shell client was programmed to ignore passwords that are passed in this manner, as shown in the following example.

```
$ ( sleep 2 ; echo "password^M"; sleep 2; ls; sleep 2 ) | ssh host -l user
Pseudo-terminal will not be allocated because stdin is not a terminal.

user@host's password: password
Permission denied, please try again.
Unable to find an authentication method
```

The `ssh(1)` command can be tricked around this limitation by using Expect. For more information on Expect, refer to the Expect FAQ at: <http://expect.nist.gov/FAQ.html>

The following is an example of a short Expect script to automate logins with passwords when using the Secure Shell:

```
#!/usr/local/bin/expect
#
#
spawn /usr/bin/ssh host -l user
expect {*password:}
send "password\r"
#
expect {*home*} exit
send "command\r"
sleep 2
#
expect {*home*} exit
```

Automated Logins

Automating logins to a system requires the script to either possess or have access to the needed authentication credentials. The problem is protecting the destination host from compromised credentials. This requires safe guarding the credentials. A balance must be struck between security and cost in terms of scalability and maintenance. There is no perfect solution.

Secure Shell provides several choices of credentials:

- Embedded passwords
- Unencrypted identities (discussed in Chapter 6)
- Embedded passphrases for encrypted user identities (discussed in Chapter 6)
- Agents

Each choice has a drawback (see TABLE 0-1). The first three choices can be defeated with read access to the secret. The fourth, agents, can be defeated with access to the machine's memory.

TABLE 0-1 Automated Login Issues

Method	Problem
Embedded passwords	Compromised by reading the script source code
Unencrypted user identities	Compromised by copying the private identity key
Embedded passphrases for encrypted user identities	Compromised by reading the script source code
Agents	Requires loading of the agent

The most resistant solution is to use agents with manually-loaded keys. The problem here is twofold: maintenance and scalability. Humans are neither completely reliable nor completely dependable. The operator must be present to reload keys into the agent in the case of a failure (for example, a system crash or power loss). The operator does not scale well either. This solution requires a central machine or small cluster of machines from which remote jobs are started. A potential single-point-of-failure exists.

There is no easy, secure solution to the problem of automated remote access. A compromise solution is to use encrypted user identities in conjunction with RBAC. Secure Shell secures the network connection. The user identity limits the authorization points (the private key and passphrase must be copied first). RBAC also limits the privileges of the account.

Host Keys

The major difference in Secure Shell being a drop-in replacement for the Berkeley r-commands is host key management. Before the user can be authenticated, the destination host's Secure Shell daemon must be authenticated to the client. This is done by matching a locally-stored host key copy to what the Secure Shell daemon offers as its host key.

When there is no locally-stored copy, both the Solaris Secure Shell software and OpenSSH default to asking whether or not to accept the newly encountered key. This adds complexity to a script, as shown in the following example.

```
$ ssh host
The authenticity of host 'host' can't be established.
RSA key fingerprint in md5 is:
7a:71:ff:d9:6d:19:d6:d9:ef:f9:4d:3f:92:7a:77:7b
Are you sure you want to continue connecting(yes/no)?
```

The least secure method to remove this complexity is to turn off host key checking by setting `StrictHostKeyChecking` to `no` in `~/.ssh/config`. The most secure solution is to acquire all of the host keys ahead of time and place them in `~/.ssh/known_hosts`. See “Managing Keys and Identities” on page 71 for more information on the difficulties of key management.

Proxies

Proxies allow application-layer connections without allowing direct network connectivity. This allows applications to bridge otherwise inaccessible networks. Proxy support can be implemented in either of two ways: using alternate networking shared libraries (for instance, the `runsocks` command) or using the internal proxy support (`ProxyCommand` keyword).

The `ProxyCommand` interface is an external command executed with `/bin/sh`. The command should read from standard input and write to standard output. See `ssh_config(4)` and the OpenSSH source code for more details on the interface. The Solaris Secure Shell software provides support for SOCKS 5 servers through the `ssh-socks5-proxy-connect(1)` command and HTTP proxy servers through the `ssh-http-proxy-connect(1)` command. Both commands use the `ProxyCommand` interface and are located in the `/usr/lib/ssh` directory.

Note – General-usage protocols, such as HTTP and SSH, can be used to allow almost anything across the proxy, including encapsulated IP traffic. Monitor the proxy for suspicious behavior. The encrypted Secure Shell traffic hides the network content but not the endpoints or the amount of bandwidth consumed.

The following code example shows the proxy access by using the `runsocks` command. Some sites require `SOCKS_SERVER` and `LD_LIBRARY_PATH` to be explicitly set.

```
$ /usr/bin/env SOCKS_SERVER=sockserver:1080 \  
LD_LIBRARY_PATH=/usr/local/socks/lib /usr/local/socks/bin/runsocks \  
/opt/OBSDssh/bin/ssh remote.host.com
```

The following code example shows the proxy access by using the `ProxyCommand` interface:

```
$ ssh -o'ProxyCommand=/usr/lib/ssh/ssh-socks5-proxy-connect \  
-h socks-gw -p 1080 dmz.foo.com 22' dmz.foo.com  
user@dmz's password: password  
Last login: Thu Dec 10 23:03:04 2002 from foo.bar.com  
Sun Microsystems Inc. SunOS 5.8 Generic May 2001  
$
```

The proxy method to access a remote host can be specified on a per-host basis with the `ProxyCommand` and `Host` keywords. The shared library method cannot be used in this manner. Note that the entire command string must be on one line.

```
Host Teach  
ProxyCommand /usr/lib/ssh/ssh-socks5-proxy-connect -h sockserver -p 1080  
teach.foo.com 22
```

Role-Based Access Control

First appearing in the Solaris 8 OE release, role-based access control (RBAC) is an alternative to the all-or-nothing superuser privilege model. In the superuser (`root`) model, the power to do anything to the system—be it deleting files, adding users, starting or stopping daemons—is granted after the superuser privilege level is obtained. RBAC provides the least privilege model.

The least privilege model breaks up the capabilities of the superuser into roles. A role is a special type of user account from which privileged applications might be run. A user assumes a role with the `su(1M)` command. For example, operators might have the backup role assigned to them that allows `ufsdump(1M)` to be run with privilege. In addition, users might have one or more roles. Even `root` can be made into a role to prevent anonymous `root` logins.

In the Solaris 8 and 9 OE releases, RBAC is optional. It is not on by default because no default roles are created. For sites with earlier Solaris OE releases, an alternative to RBAC is `sudo`, maintained by Todd Miller. `sudo` provides the capabilities of `su(1M)` on a per-command per-user assignment basis.

TABLE 0-2 Pros and Cons of Using RBAC, `sudo`, and `root` Shell

Name	Pro	Con
RBAC	Limits privilege by either granting limited <code>root</code> level privileges or restricting what a user account can do. It is an integral feature of the Solaris OE.	Requires a change in the mind set away from the superuser model. Is not available in the Solaris 2.6 or 7 OE.
<code>sudo</code>	Works on non-RBAC-capable Solaris OE releases.	Must be downloaded, built, and tested. Is not Sun supported software. Grants only limited access to <code>root</code> level privilege.
<code>root</code> Shell	Most power in using a system.	Any direct <code>root</code> login can do anything to the system. Console logins defeat auditing.

▼ To Restrict a User to Only Copying Files Using RBAC

1. Become the superuser.
2. Add the execution attributes of the role.

```
# cat <<_EOM_ >> /etc/security/exec_attr
> Restricted Secure Shell:suser:cmd:::/usr/bin/scp:
> _EOM_
```

3. Add the name of the role.

```
# cat <<_EOM_ >> /etc/security/prof_attr
> Restricted Secure Shell:::scp access only:
> _EOM_
```

4. Comment out `PROFS_GRANTED=Basic Solaris User` in the `/etc/security/policy.conf` file.

5. Assign the user the role.

```
# cat <<_EOM_ >> /etc/user_attr
> user:::profiles=Restricted Secure Shell
> _EOM_
```

6. Change the user's shell to a profile shell.

```
# usermod -s /usr/bin/pfksh user
```

7. Restart the `nscd` daemon.

```
# /etc/init.d/nscd stop
# /etc/init.d/nscd start
```

The user will only be able to execute built-in shell commands and `scp(1)`, as in the following example.

```
localhost$ ssh remotehost -l user
user@remotehost's password: password
Last Login: Thu Dec 12 21:51:44 2002 from someplace
remotehost$ pwd
/home/user
remotehost$ cd /
remotehost$ ls
pfksh: ls: not found
remotehost$ pwd
/
remotehost$ cat /etc/passwd
pfksh: cat: not found
remotehost$ exit
localhost$ ssh remotehost -l user cat /etc/passwd
Last Login: Thu Dec 12 21:51:44 2002 from someplace
pfksh: cat: not found
localhost$ scp index.html user@remotehost:index.html
user@remotehost's password: password
index.html      100% |*****| 526 00:00
localhost$
```

For more information on RBAC, refer to the following documents:

- *Solaris 9 OE System Administration Guide*, specifically the Security Services chapter
- “Solaris Operating Environment Security: Updated for the Solaris 9 Operating Environment,” December 2002, by Alex Noordergraaf and Keith Watson
- `su(1M)`
- `roles(1)`
- `policy.conf(4)`
- `exec_attr(4)`
- `prof_attr(4)`
- `usermod(1M)`
- `roledel(1M)`
- `rolemod(1M)`
- `roleadd(1M)`

For information on `sudo`, refer to: <http://www.courtesan.com/sudo/>

Port Forwarding

Secure Shell can encapsulate TCP-based application data streams and then tunnel them across its secure connection to and from the client and server. This is referred to as port forwarding. Port forwarding is useful to secure communications with legacy platforms or internal systems such as the internal web site, internal Internet relay chat (IRC) network, or email access.

Port forwarding is not transparent to the application. The application requires some configuration to use the forwards. For situations requiring transparency, a network level solution such as IPsec or VPNs must be used. Port forwarding will not work for UDP-based protocols or for protocols, such as IRC DCC channels, that dynamically allocate a second data stream on a separate port.

Forwards can either be specified on the command line or in the client configuration file (recommended for a system with multiple forwards). Forwards can also be local (from client to server) or remote (server to the client). Port forwarding and RBAC can be used to provide secure access to an IMAP mail server while preventing the users from having access to the server itself.

The following two examples show the local forwarding of port 8080 on the client to port 80 on the server.

This example shows forwarding using the `ssh(1)` command:

```
$ ssh -L8080:server:80 server
```

This example shows forwarding using the client's configuration file:

```
Host server
LocalForward 8080 server:80
```

Note – The Solaris Secure Shell software disables port forwarding by default. See “Connection and X11 Forwarding” on page 46 for more details.

▼ To Secure WebNFS Mounts With Port Forwarding

1. **Choose an unused local port and forward it to the WebNFS™ port on the server.**

A different local port will be needed for each server. This connection must be maintained for the life of the mount.

```
$ ssh -f -N -L3030:server:2049 server
```

2. **Become the superuser.**
3. **Mount the file system using the forwarded port.**

```
# mount nfs://localhost:3030/export/home /mnt
```

Note – This procedure provides transport-level protection only for the WebNFS traffic. Using Secure Shell in this manner does not provide additional WebNFS authentication.

Disabling Insecure Services

Insecure services can be disabled by commenting them out of `inetd.conf`. The comment character is a hash (`#`). Consider making a backup copy of `inetd.conf` before editing. For information on `inetd(1M)` and `inetd.conf(4)`, consult their respective man pages.

Remove any service not needed for your environment. In particular, remove `ftp`, `telnet`, `shell`, `login`, and `exec`. Consider removing `echo`, `discard`, `daytime`, `chargen`, `comsat`, and `talk` services as well. These are normally not needed.

▼ To Disable Insecure Services

1. **Become the superuser.**
2. **Edit `/etc/inetd.conf` and comment out insecure services.**
3. **Use the `kill(2)` command to send the `HANGUP` signal to `inetd(1M)`.**

```
# ps -ef | grep inetd | grep -v grep
  root   153      1  0   Dec 09 ?           0:02 /usr/sbin/inetd -s
# kill -HUP 153
```

4. **Ensure that the services have been disabled.**

```
$ telnet localhost
Trying 127.0.0.1...
telnet: Unable to connect to remote host: Connection refused
$ rsh localhost
localhost: Connection refused
```

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.

References

- Barrett, Daniel J. and Richard E. Silverman. *SSH: The Secure Shell*. Sebastopol, CA: O'Reilly, 2001.
- Haines, Michael and Joep Vesseur. "Extending Authentication in the Solaris 9 Operating Environment Using Pluggable Authentication Modules (PAM): Part I." September 2002.
- Haines, Michael and Joep Vesseur. "Extending Authentication in the Solaris 9 Operating Environment Using Pluggable Authentication Modules (PAM): Part II." October 2002.
- Miller, Todd. "SUDO." 2002. <http://www.courtesan.com/sudo/>
- Powell, Brad, Dan Farmer and Matthew Archibald. "Titan Frequently Asked Questions." 2002. <http://www.fish.com/titan/FAQ.html>
- Reid, Jason, "Building OpenSSH—Tools and Tradeoffs," Sun BluePrints OnLine, January 2003, at: <http://www.sun.com/blueprints/0103/817-1307.pdf>

Ordering Sun Documents

The SunDocsSM 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>