Global Information Assurance Certification Paper

Name: Matthew Geiger
Date: 19 December 2001
Assignment: GSEC v1.2f

SSHell Game - Stacking the odds in your favor

> "Those you trust the most can steal the most"
> - Lawrence Lief, industrial security analyst
> *Newsweek* 12/26/83

Introduction

For remote shell access and server administration, the SSH secure shell protocol represents a quantum leap in security over predecessors like telnet, rlogin and rsh. SSH provides strong encryption of the communications channel and cryptographic authentication of b oth server and client. Unsurprisingly, SSH has been enthusiastically adopted as the new de facto standard, with commercial and open -source versions appearing for pretty much every platform.

Indeed, it may be that SSH's impressive feature -set and solid credentials lulled its converts into a sense of complacency. Whatever the case, some recent vulnerabilities and a major working exploit have spotlighted some pretty universal lessons about secure practices and how they apply to SSH. This paper takes a look at those lessons in the context of SSH's implementation and deployment, with an emphasis on its configuration. Unless otherwise noted, specific references are to OpenSSH (www.openssh.org), the most widely used implemen tation of the protocol – and still the fastest growing.

What happened?

Since its first application in 1995, SSH has earned a reputation for stability, well -designed security and an impressive set of features that allow, among other things, a variety of port-forwarding / connection -tunneling options. As a result of this flexibility, SSH has been incorporated into a wide range of open source and commercial products that need to set up secure communications channels with strong encryption and authentication. The range encompasses everything from network appliances to mainframes. SSH has also inspired a budding family of derivative protocols such as SFTP, a secure replacement for FTP, and even an scp program that replaces the remote file copying features of rcp .

The introduction of the open -source OpenSSH package gave a big boost to the standard's adoption. To give some measure, consider this: OpenSSH's first release was at the end of 1999, yet by December 2001 OpenSSH accounted for 52% of servers accepting SSH connections. SSH's active and relatively rapid development has been a boon to network security. But the pace of development – marked by a variety of implementations, two incompatible protocol versions and the need to support a number of different OS -based authentication schemes – clearly presents a steep challenge in maintaining flawlessly secure code.

Indeed, the OpenSSH security page lists 14 known security vulnerabilities in various

implementations. And recent bugs have been among the most severe – potentially awarding their exploiters root privileges.

In February 2001, Michal Zalewski of BindView's RAZOR team discovered a weakness in the system used under the SSH1 protocol – ironically in a function used to help detect a crypto-cracking attack on SSH. The weakness – a buffer overrun – allows an attacker to inject malicious code onto an SSH-running system and have it executed without needing to be authenticated as a user.

Fixed versions of the vulnerable SSH1 implementations were issued on the heels of the announcement. But the upgrading cycle was relatively slow, considering that a successful exploit of the vulnerability would give the attacker root access in most cases. At the time the vulnerability was published, between 65% and 80% of SSH-running servers accessible from the Internet were vulnerable, according to a study by the University of Michigan's Center for Information Technology Integration. In July 2001, more than 50% continued to run the vulnerable service.

From theoretical to real

It might be that the unhurried adoption of the SSH upgrade was partly because the lack of a working exploit for the vulnerability. But that changed sometime around September 2001, when the first reports of exploits appeared in security community forums. In November, Dave Dittrich of the University of Washington published a detailed analysis of the attack and compromise of a server running the vulnerable SSH1 configuration. The attack routine had already been scripted and packaged by then – and at least two prepackaged versions of the exploit have certainly entered wide circulation since.

CERT and other incident monitoring bodies have reported a marked rise in scanning targeting SSH's default port 22. For the past month, up to 17 Dec 2001, scans for port 22 have been consistently in the six most active reported by DShield, the collaborative, distributed scan detection organization (www.dshield.org). This jibes with anecdotal evidence from firewall logs I review, which show port 22 scans among the top five in the past month (the other four [excluding NetBios scans] are: SMTP, FTP, Squid and DNS).

Yet, according to the latest survey data from U Mich's CITI, more than 30% of servers offering SSH service to the Internet remained vulnerable to the attack as recently as early December 2001. With packaged, no-expertise-needed attacks widely available in the cracker underground, along with automated scanning tools that can fingerprint vulnerable servers, that's a lot of servers ready to be rooted out there.

Another potentially serious vulnerability, this one exclusive to OpenSSH, came to light more recently. At the beginning of December, members of the FreeBSD and OpenBSD team reported that OpenSSH servers set up to use the underlying OS' login system could incorrectly execute instructions with the privileges of the SSH daemon, usually root. Although the use of login is disabled by default in OpenSSH, which has its own authentication scheme, it's nonetheless fairly common in systems that want the local user control that login provides.

It's worthwhile noting that this was OpenSSH's second root-yielding vulnerability related to accommodating the login function. The previous flaw, disclosed in mid-2000, allowed users to issue arbitrary commands with SSH's privileges.

Lessons

Hmm. Working root exploits. A pretty healthy flow of serious updates and/or patches. Does this mean SSH is not such a good thing after all – maybe even damaged goods? The consensus is a definite no.

The real message is much less dramatic, but no less important. SSH – like every critical package running on any computer – needs attention. It needs attention during configuration; it needs to be patched or upgraded diligently, especially in reaction to serious vulnerabilities; it should be routinely monitored.

In addition, although SSH-protected communications may be ridiculously tough to crack, it's unwise to treat your SSH *server* as impregnable – even when it's patched and carefully configured. Like any critical, privileged service, it should be shielded to the greatest extent practical and be integrated into a holistic, layered security strategy.

The first step

First, a review of some key features of the default configuration file, sshd_config, for OpenSSH – usually found in the SSH configuration and key storage directory. Lines that start with # are commented out in the default OpenSSH configuration. I've omitted a number of lines that don't have significant security implications or where it's not a good idea to fiddle with default settings. My comments are set off with \\.

```
Port 22 \\ Setting the server to listen on a different port will
        \\ evade some scanners but can be inconvenient and doesn't
        \\ provide much real security.

#Protocol 2,1    \\ The cheapest, easiest security enhancement:
                 \\ uncomment this line and delete the 1. Removing
                 \\ support for the outdated SSH1 would make
                 \\ this server not vulnerable to the most serious
                 \\ documented SSH exploits. Six of the 14 known
                 \\ vulnerabilities listed by OpenSSH apply
                 \\ specifically to the SSH1 protocol.

#ListenAddress 0.0.0.0  \\ By default, SSH listens on all interfaces.
                        \\ On multi-homed machines that only need to
                        \\ offer SSH connections on one interface,
                        \\ specify that interface's IP address here.
                        \\ This is particularly relevant for machines
                        \\ with an Internet-facing interface but that
                        \\ only need to allow SSH connections on the
                        \\ private interface.

PermitRootLogin yes    \\ If you're offering SSH service exclusively
                       \\ to ordinary users, it's a no-brainer
                       \\ to specify "no" here. Better yet, with a
```

```
                        \\ little fiddling, SSH can be set up to run
                        \\ as a user other than root. See  man sshd.
```
(Of related interest is the AllowGroups keyword, which is followed by a list of group names, in which the * and ? wildcards are allowed. Only members of the specified groups will be allowed login. The AllowUsers keyword functions the same way but allows granular, user-by-user control for those s ysadmins who tie cans to their bedroom doorknobs when they sleep ... if they sleep. Neither keyword appears in the default OpenSSH configuration file; they must be added to use these features. There are also DenyGroups and DenyUsers options, but it's bette r practice to specify only those authorized and deny the rest.)

```
# Don't read ~/.rhosts and ~/.shosts files
IgnoreRhosts yes // Almost always should stay yes. It's hard to
                 // imagine a more ironic way to undermine the
                 // security of SSH than to rel y on .rhosts
                 // authentication and run the rlogin or rsh
                 // facilities alongside it. Anyone who can alter
                 // the .rhosts file could authenticate to ssh.

#IgnoreUserKnownHosts yes  // Do you trust (or need) users to set up
                           // host verification for SSH
                           // connections under their own login
                           // authority. If the answer is no, a
                           // common alternative is to uncomment the
                           // line and set up a global list of known
                           // hosts. Authentication based on
                           // the public k ey of the connecting host,
                           // for which the known_hosts file is also
                           // used, introduces further risk. If this
                           // is enabled, allowing users to set
                           // their own known_hosts file should be
                           // carefully considered. See
                           // HostbasedAuthentication below.

X11Forwarding yes     // Note that because SSH supports a wide range
                      // of port-forwarding strategies, setting this
                      // to no won't stop an intent user from
                      // setting up his own X forwarding system.

SyslogFacility AUTH PRIV     // It's a good idea to log at least at
LogLevel  INFO               // this level of detail, and check the
                             // logs for anomalies in SSH access.


RhostsAuthentication no      // If set to yes, this would allow
                             // insecure authentication by IP or
                             // hostname contained in the rhosts or
                             // /etc/hosts.equiv files.

HostbasedAuthentication no  // This would allow authentication based
                            // on entries in rhosts or
                            // /etc/hosts.equiv together with the
                            // client host's public key's entry in
```

```
                              // the relevant known_hosts file. This
                              // confers login authority to the remote
                              // computer, not the user, and is not a
                              // good idea. This is also affected by
                              // setting the IgnoreUserKnownHosts
                              // keyword (see above). The S SH1 protocol
                              // equivalent keyword for this method is
                              // RhostsRSAAuthentication.

PubkeyAuthentication yes      // One of the standard and more secure
                              // ways to authenticate a login. This is
                              // based on the connecting client
                              // supplying the correct RSA or DSA
                              // cryptographic credentials that match a
                              // corresponding public key stored on the
                              // SSH-providing server, in the relevant
                              // user's authorized_keys2 file. The
                              // connecting user needs both the priva te
                              // key and the passphrase to
                              // authenticate.
(Sounds perfect and the method is preferred - but there are
weaknesses. A big one is that there is no way to evaluate the
quality of the passphrase from the SSH server; it could be blank
[and oftе n is when users are scripting automated secure tunneling
sessions]. That degrades the security to something resembling more
the above host-based methods, depending on how well -controlled user
access is to the client. On the plus side, a variety of key -specific
options [including barring TCP forwarding] can be set in the
authorized_keys2 file.
The SSH1 equivalent keyword is RSAAuthentication. Default value for
both keywords is yes.)


PasswordAuthentication yes    // Allows remote users to authenticate
                              // based on the password of the user
                              // they want to log on as. This doesn't
                              // have the brute-forcing protection that
                              // key-based authentication does because
                              // there's no private key needed, but it
                              // does allow control and audit ing of the
                              // password quality. If this is the only
                              // means of authentication and access
                              // control, it places all the security
                              // emphasis on the quality of the
                              // passphrase and the soundness of the
                              // login mechanism.

PermitEmptyPasswords no       // No explanation needed. Really.

#PAMAuthenticationViaKbdInt yes   // This allows the use of PAM
                              // access-control & authentication
                              // mechanisms for login; that almost
                              // always means plain user passwords
                              // are allowed - even if
                              // PasswordAuthentication is set to
```

```
                              // no.

#KerberosAuthentication no  // OpenSSH allows Kerberos authentication
                            // by default. Integrating authentication
                            // means the SSH server needs to verify
                            // and communicate with a Kerberos domain
                            // server - setting that relationship up
                            // is another story and requires careful
                            // configuration.

#UseLogin no     // This feature, disabled by default, is the source
                 // of two potential root -yielding vulne rabilities
                 // (see the What Happened? section above). Avoid
                 // unless critical.

#MaxStartups 10:30:60  // This sets the maximum number of concurrent
                       // unauthenticated connections accepted by the
                       // SSH daemon. The default is 10, which shoul d
                       // be plenty for all but the busiest servers.
(For a busier server, the 10:30:60 setting commented out above would
specify that after 10 unauthenticated concurrent connections, the
server starts dropping 30% of new connection attempts, with the drop
rate rising to 100% as concurrent  unauthenticated connections
approach 60. Tweaking these settings should allow busy servers to
handle many SSH connections, while still making life hard for brute  -
force-based attempts.)

#ReverseMappingCheck yes    // Checks h ostname and corresponding DNS
                            // records with actual IP address of
                            // originating connection. May be useful
                            // in some environments, but is probably
                            // more trouble than it is worth in most
                            // - especially ones in which SSH clien ts
                            // connect from dynamically assigned
                            // addresses. Offers less security
                            // against spoofing than a properly used
                            // client or user RSA key.
```

Weaving SSH into the defense infrastructure

In the practical application of most critical s ervices, the tightest, most secure configuration can be hard to achieve. More often than not, a compromise is struck in the interest of functionality or compatibility. Other times, an unanticipated vulnerability puts a hole in an otherwise solid setup. SSH is clearly no exception.

Beyond judicious configuration and the diligent application of patches or upgraded packages, SSH services should be integrated into system defenses. Even if a multi - interface SSH server is configured to listen only on the private interface, it's wise to firewall port 22 on the public interface(s) anyway. Similarly, if host -based access rules are applied in setting up SSH, they should be mirrored at the firewall or in TCP Wrappers, if appropriate. And, of course, network IDS monito rs need to include SSH - attack signatures. SSH log monitoring should be given the same attention as afforded the logs of any critical service.

In other words, even though SSH is an excellent line of defense itself, it needs to be meshed tightly into the la yers of a good defense -in-depth strategy. That process only starts with careful configuration. SSH is too powerful a tool to put it in the wrong hands.

References

OpenSSH's list of SSH security issues
http://www.openssh.org/security.html

CERT
SSH CRC32 attack detection code contains remote integer overflow
http://www.kb.cert.org/vuls/id/945216
Exploitation of vulnerability in SSH1 CRC -32 compensation attack detector
http://www.cert.org/incident_notes/IN -2001-12.html
The recent UseLogin vulnerability outlined
http://www.kb.cer t.org/vuls/id/157447

Razor Bindview's advisory on SSH CRC32 vulnerability
http://razor.bindview.com/publish/advisories/adv_ssh1crc.html

Dave Dittrich's analysis of a succes sful CRC32 attack
http://staff.washington.edu/dittrich/misc/ssh -analysis.txt

Center for Information Technology Integration's scan -based survey of SSH servers
http://www.citi.umich.edu/u/provos/ssh/
and details in PDF format: http://www.citi.umich.edu/techreports/reports/citi -tr-01-13.pdf

A second, more efficient version of the CRC32 attack discovered
http://www.incidents.org/diary/diary.php?id=118

SSH, The Secure Shell: The Definitive Guide
published by O'Reilly (http://www.oreilly.com/catalog/sshtdg/)

And, of course, OpenSSH's man pages for ssh, sshd, and sftp.
Available on a system near you ... or at: http://www.openssh .org/manual.html

Last Updated: July 20th, 2017

Upcoming Training

SANSFIRE 2017	Washington, DC	Jul 22, 2017 - Jul 29, 2017	Live Event
SANS Boston 2017	Boston, MA	Aug 07, 2017 - Aug 12, 2017	Live Event
SANS Prague 2017	Prague, Czech Republic	Aug 07, 2017 - Aug 12, 2017	Live Event
Community SANS Omaha SEC401*	Omaha, NE	Aug 14, 2017 - Aug 19, 2017	Community SANS
SANS Salt Lake City 2017	Salt Lake City, UT	Aug 14, 2017 - Aug 19, 2017	Live Event
SANS New York City 2017	New York City, NY	Aug 14, 2017 - Aug 19, 2017	Live Event
Community SANS Trenton SEC401	Trenton, NJ	Aug 21, 2017 - Aug 26, 2017	Community SANS
Virginia Beach 2017 - SEC401: Security Essentials Bootcamp Style	Virginia Beach, VA	Aug 21, 2017 - Aug 26, 2017	vLive
SANS Virginia Beach 2017	Virginia Beach, VA	Aug 21, 2017 - Sep 01, 2017	Live Event
SANS Chicago 2017	Chicago, IL	Aug 21, 2017 - Aug 26, 2017	Live Event
SANS Adelaide 2017	Adelaide, Australia	Aug 21, 2017 - Aug 26, 2017	Live Event
Community SANS Pasadena SEC401 @ NASA	Pasadena, CA	Aug 23, 2017 - Aug 30, 2017	Community SANS
Mentor Session - SEC401	Minneapolis, MN	Aug 29, 2017 - Oct 10, 2017	Mentor
SANS San Francisco Fall 2017	San Francisco, CA	Sep 05, 2017 - Sep 10, 2017	Live Event
SANS Tampa - Clearwater 2017	Clearwater, FL	Sep 05, 2017 - Sep 10, 2017	Live Event
Mentor Session - SEC401	Edmonton, AB	Sep 06, 2017 - Oct 18, 2017	Mentor
SANS Network Security 2017	Las Vegas, NV	Sep 10, 2017 - Sep 17, 2017	Live Event
Mentor Session - SEC401	Ventura, CA	Sep 11, 2017 - Oct 12, 2017	Mentor
Community SANS Albany SEC401	Albany, NY	Sep 11, 2017 - Sep 16, 2017	Community SANS
Community SANS Columbia SEC401	Columbia, MD	Sep 18, 2017 - Sep 23, 2017	Community SANS
Community SANS Dallas SEC401	Dallas, TX	Sep 18, 2017 - Sep 23, 2017	Community SANS
Rocky Mountain Fall 2017	Denver, CO	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS London September 2017	London, United Kingdom	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS Baltimore Fall 2017	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS Copenhagen 2017	Copenhagen, Denmark	Sep 25, 2017 - Sep 30, 2017	Live Event
Community SANS Boise SEC401	Boise, ID	Sep 25, 2017 - Sep 30, 2017	Community SANS
Baltimore Fall 2017 - SEC401: Security Essentials Bootcamp Style	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	vLive
Community SANS New York SEC401	New York, NY	Sep 25, 2017 - Sep 30, 2017	Community SANS
Community SANS Charleston SEC401	Charleston, SC	Oct 02, 2017 - Oct 07, 2017	Community SANS
Community SANS Sacramento SEC401	Sacramento, CA	Oct 02, 2017 - Oct 07, 2017	Community SANS
SANS DFIR Prague 2017	Prague, Czech Republic	Oct 02, 2017 - Oct 08, 2017	Live Event