Global Information Assurance Certification Paper

One Incident Of Remediating The CRC 32 sshd1 Vulnerability

Rebecca Sander v1.2f

Introduction

The purpose of this paper is to document the process I used to respond to the CRC32 sshd1 vulnerability. My operating environment is primary Solaris and Linux, with a small percentage of HPUX and OpenBSD. Most systems are behind a corporate firewall, but a few are on the internet and used as data transfer points. This is only the process I used, and not the only acceptable response. I will document the four steps I used and my results. The four steps consisted of;

1) Being informed of the vulnerability.
2) Researching the vulnerability and possible ways to remediate it.
3) Deploy the fix.
4) Test the fix.

Step 1

I subscribe to several e-mail sources to keep up to date on current vulnerabilities. I prefer e-mail because I don't have to surf the net for up to date security information, it comes to me, and the sources are reliable. They are listed below with subscription information.

SANS Security Alert Consensus

http://www.sans.org/sansnews/
Subscribe to the newsletters at this URL. You can select which OS flavors you're interested in, which helps keep information overload at bay. You can find archived copies on this website also.

HP Security Bulletins Digest

http://www.itresourcecenter.hp.com
Hewlett Packard contract customers can subscribe to the security update digest with their IT Resource Center username and password. This is also available without an HP Support Contract, but you'll have to register. This digest notifies me of vulnerabilities in HPUX and gives information about applicable patches or work arounds. Bulletins are archived at this site, also.

Sun Security Bulletin

To subscribe to the Sun Security Bulletin mailing list, send email to security-alert@sun.com with a subject line containing subscribe. No Sun support contract is needed. This digest notifies me of vulnerabilities in Solaris and gives information about applicable patches or work arounds. There is a link to the digest

archives at http://sunsolve.sun.com.

CERT Advisories

http://www.cert.org/
To subscribe to the CERT mailing list for advisories and bulletins, send email to majordomo@cert.org. Please include in the body of your message subscribe cert-advisory. Advisories are archived at http://www.cert.org/advisories.

Step 2

The CRC32 vulnerability first came to my attention in February 2001 in a CIAC Bulletin forwarded to me from another sys admin. L-047: OpenSSH SSH1 Coding Error and Server Key Vulnerability, February 13, 2001. (1) It was described as a possible remote root compromise, but difficult to exploit. The suggested work around was to upgrade to ssh2.3.0 and disable ssh1 compatibility. Older versions of ssh2 were still vulnerable, so it was important to upgrade to at least version 2.3.0. Some of my considerations before doing that were; Openssh does not support kerberos, which our site uses. Data flow between our remote sites and us would be disrupted while we coordinated a joint upgrade to ssh2. Ssh version 2 supports kerberos, but there are licensing issues to be considered. These considerations combined with the fact that I was not seeing any log entries that looked unusual, and the fact that the vulnerability was difficult to exploit, all made me decide that this was not a threat big enough to do any thing other than be aware of it for now.

In November 2001, I began seeing log entries for connections to sshd from unfamiliar IP addresses. They were single connections, sometimes from IP's that would resolve, and sometimes not. None of the connections were authorized. This appeared to me to be reconnaissance probes. The system that was reporting these connections is by necessity on the internet and not behind a corporate firewall, so it's not unusual to see stray connections. I continued to make a note of this information. Some log entries are listed below as examples of what I was seeing.

Nov 9 02:11:44 host1 sshd[30467]: log: Connection from xxx.xxx.xxx.xxx port 4181
Nov 17 07:48:29 host1 sshd[732]: log: Connection from xxx.xxx.xxx.xxx port 1666
Nov 22 02:22:15 host1 sshd[14693]: log: Connection from xxx.xxx.xxx.xxx port 4598
Nov 29 03:41:31 host1 sshd[6725]: log: Connection from xxx.xxx.xxx.xxx port 1054
Nov 30 06:28:52 host1 sshd[24683]: log: Connection from xxx.xxx.xxx.xxx port 1426
Nov 30 10:59:32 host1 sshd[11961]: log: Connection from xxx.xxx.xxx.xxx port 3813
Dec 1 21:25:17 host1 sshd[12919]: log: Connection from xxx.xxx.xxx.xxx port 3228

Also in November 2001, another administrator forwarded to me an entry from the Handlers Diary on www.incidents.org dated November 12th, 2001.(2) It reported that a script had been written and turned loose in the wild that would exploit the ssh1 CRC32 vulnerability, with the results of analysis of the exploit by David Dittrich, Washington

University. (3) This analysis pointed out one very interesting fact to me. In order for this exploit to work, attacker and victim had to exchange keys. Therefore, vulnerable ssh1 implementations could not be exploited if they were configured with IP filters or hostallow configurations. This information was critical for me, as all of the problems preventing me from upgrading to ssh2 quickly were still present, but now the threat seemed much more imminent. I found several documents on the internet as I researched the validity of using sshd1 configuration settings to prevent my sshd1 systems from being exploited. These documents all said the same thing. Preventing a key exchange would prevent this vulnerability from being exploited. The full text can be seen at the URL's listed in the references. I include these excerpts, as they are the basis for the decision I made.

Excerpt from 11/12/01 handlers diary (2):

> In order for the exploit to work, the attacker must be able to engage the victim system in a cryptographic key exchange dialog. Thus, servers protected by packet filters or access control restrictions will be immune to attacks from restricted addresses.

Excerpt from paper at Washington University, referenced in the handler's diary (3):

> The exploit does not work against systems that use access control restrictions (e.g., SSH.com's "AllowHosts" or "DenyHosts" settings) or packet level filters (e.g., ipchains, iptables, ipf) which would prevent a host from attempting to exchange public keys. The vulnerability requires being able to enter cryptographic key exchange negotiation with the server to properly manipulate the stack.

Excerpt from http://razor.bindview.com/publish/advisories/adv_ssh1crc.html (4):

> Update(12-06-01): There are at least three exploits being used in the wild for mass defacements of Linux systems. We urge all administrators to upgrade their SSH daemons as soon as possible or to verify that their installations do not suffer from SSHv2 -> SSHv1 fallback problem. Note that the attacker needs to make a TCP connection from an IP address for which sshd will enter into a key-exchange dialogue. If the attacker's packets have a source IP address that is disallowed by (for example) DenyHosts in the sshd configuration file, the key exchange will not happen and the attacker will not have an opportunity to compose the required exploit data.

Excerpt from 11/12/01 Handlers Diary(2):

> "The successful exploit causes an authentication attempt to pause while the shell code back door becomes active. You can connect to the shell and do whatever you want. Only problem is, the original SSH daemon (at least with SSH.com

> 1.2.31) will timeout when the authentication doesn't complete, and the shell will be terminated. This gives a window of ten minutes (at least with SSH.com 1.2.31) before the listening shell's parent dies and another exploit attempt must be started. (That is plenty of time to fully root the box eight ways from Sunday, unfortunately.)"

Excerpt from CERT Advisory (5):

> If you cannot disable the service (ssh), you can limit your exposure to these vulnerabilities by using a router or firewall to restrict access to port 22/TCP (SSH). Use tcp wrappers or a program that provides similar functionality, or use the key-based IP restriction offered by your implementation. Note that this does not protect you against attackers from within your network.

Step 3

Based on the above information, I chose a course of action. I chose not to upgrade to ssh2 without ssh1 fallback until it could be coordinated with remote sites. I communicated with the sys admins at our remote sites so that we would all deploy and test these changes together, so the data flow would not be interrupted. I made three changes to the sshd_config files on all machines, not just the internet visible ones. Then, I sent a SIGHUP to the sshd process. The three options I changed are listed below:

```
HostsAllow xxx.xxx.*
SilentDeny yes
Timeout 120
```

HostsAllow xxx.xxx.*
By setting this option, the sshd daemon won't allow a client connection unless it matches the addresses defined with this option. This will accept IP addresses, host names and wildcards. I used this argument to limit allowed IP addresses to ClassB addresses in our corporate network. External internet systems I set to specific host IP addresses of our remote sites.

SilentDeny
If the connecting system isn't allowed to connect, sshd silently drops the connection. It gives no useful reconnaissance information about sshd version, etc.

Timeout 120
Although I've seen no analysis that says how long it might take to push a root kit onto my system, I thought 10 minutes was excessive, so I've decreased this timeout to two minutes.

Step 4

Step 4 is to test the fix. In this case, I wanted to verify that the sshd1 daemon will allow the clients that are specified on the HostsAllow option to connect. Also, I wanted to be sure that all other clients would be denied and the server wouldn't give out any information about it's configuration to denied clients. NOTE: With the exception of test 3, the sshd version shown in the following tests is the client version, not the server being probed.

Test1:
This is the verbose output, as seen from the connecting client, of a connection to an sshd1 server with all three options set. This is run from a client that is listed as an allowed client in the HostsAllow option. (I wanted to be sure it still works for the good guys!)

```
host1% ssh -v host2
SSH Version 1.2.27 [hppa1.0-hp-hpux11.00], protocol version 1.5.
Compiled with RSAREF.
host1: Reading configuration data /etc/ssh_config
host1: ssh_connect: getuid 1111 geteuid 1111 anon 1
host1: Connecting to host2 [xxx.xxx.xxx.xxx] port 22.
host1: Connection established.
host1: Remote protocol version 1.5, remote software version 1.2.26
host1: Waiting for server public key.
host1: Received server public key (768 bits) and host key (1024 bits).
host1: Host 'host2' is known and matches the host key.
host1: Initializing random; seed file /home/user/.ssh/random_seed
host1: Encryption type: idea
host1: Sent encrypted session key.
host1: Installing crc compensation attack detector.
host1: Received encrypted confirmation.
host1: Remote: Server does not permit empty password login.
host1: No agent.
host1: Trying RSA authentication with key 'user@host1'
host1: Server refused our key.
host1: Doing password authentication.
user@host2's password:
host1: Requesting pty.
host1: Requesting X11 forwarding with authentication spoofing.
host1: Remote: X11 forwarding disabled in server configuration file.
Warning: Remote host denied X11 forwarding, perhaps xauth program could not be run
on the server side.
host1: Requesting shell.
host1: Entering interactive session.
```

In test 1, the sshd1 server checked the connecting clients IP address against the HostsAllow list, found the address, and allowed the connection.

Test 2:
This is the verbose output, as seen from the disallowed connecting client, with all three options set on the server:

```
host1% ssh -v host2
SSH Version 1.2.26 [i386-unknown-openbsd2.5], protocol version 1.5.
Compiled with RSAREF.
host1: Reading configuration data /etc/ssh_config
host1: ssh_connect: getuid 1111 geteuid 0 anon 0
host1: Connecting to host2 [xxx.xxx.xxx.xxx] port 22.
host1: Allocated local port 605.
host1: Connection established.
Connection closed by foreign host.
```

In test 2, the sshd1 server has checked the connecting client IP address against the HostsAllow list, has not found the connecting IP, and drops the connection. The sshd1 server does not reveal any information as to it's own version because the SilentDeny option is set to yes. Compare Test 2 to Test 3. Test 3 shows a connection from a disallowed client with the SilentDeny option not set.

Test 3: This is the verbose output, as seen from the disallowed connecting client, to an sshd1 server with HostsAllow option set but without SilentDeny set:

```
host1:user {124} ssh -v host2
SSH Version 1.2.26 [i386-unknown-openbsd2.5], protocol version 1.5.
Compiled with RSAREF.
host1: Reading configuration data /etc/ssh_config
host1: ssh_connect: getuid 1111 geteuid 0 anon 0
host1: Connecting to host1 [xxx.xxx.xxx.xxx] port 22.
host1: Allocated local port 602.
host1: Connection established.
host1: Remote protocol version 1.5, remote software version 1.2.26
host1: Waiting for server public key.
Sorry, you are not allowed to connect.
```

In test 3 the sshd1 server reveals the server version on the Remote protocol line because the SilentDeny option isn't set to yes.

Based on the results of testing the fix, I feel reasonable confident that our ssh1 servers will not be exploited via the CRC32 vulnerability. Now we can plan an upgrade to ssh2 when it won't interfere with production activities.

Conclusion

I chose this incident for the topic of my paper because it illustrates well several concepts that are presented in the GIAC class. First, have a plan. I use specific, reliable tools to stay informed of current security threats. I use those tools to quickly decide which threats I must respond to, and which can be safely disregarded. The internet is the most current information source for the rapidly changing security scene, and I used those sources extensively while dealing with this vulnerability. I believe that as system administrators, it is essential that we understand each new threat and respond in an informed manner. Simply applying a patch isn't always the best, or only, answer for every site. This incident also illustrated the point that each vulnerability needs to be assessed to decide how high the risk is in comparison to the effort to eliminate it. This particular vulnerability changed from low threat to high threat. To date, I am continuing to see probes on my internet systems. I will continue to monitor my logs and the available information on this exploit, and respond accordingly.

References:

(1) Unknown, L-047: OpenSSH SSH1 Coding Error and Server Key Vulnerability, February 13, 2001. http://www.ciac.org/ciac/bulletins/l-047.shtml

(2) Unknown, Handlers Diary, Nov. 12, 2001. http://www.incidents.org/diary.php?id=16

(3) David A. Dittrich, Analysis of SSH crc32 compensation attack detector exploit, Nov. 15, 2001. http://staff.washington.edu/dittrich/misc/ssh-analysis.txt

(4) Michal Zalewski, Remote vulnerability in SSH daemon crc32 compensation attack detector, February 8, 2001.
http://razor.bindview.com/publish/advisories/adv_ssh1crc.html

(5) Unknown, CERT Advisory CA-2001-35 Recent Activity Against Secure Shell Daemons, December 13, 2001. http://www.cert.org/advisories/CA-2001-35.html