GIAC
CERTIFICATIONS

Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Hacker Tools, Techniques, and Incident Handling (Security 504)"
at http://www.giac.org/registration/gcih

Dealing with an SSH Compromise

An Idealized Incident Handling Scenario
Prepared for SANS GIAC Incident Handling Certification Practical Study, version 2.0
By Thomas Stocking, GCIA

2/2002

© SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Dealing with an SSH Compromise 1
Introduction 3
Part 1 - The Exploit 3
Name 3
CVE-2001-0144 3
Operating Systems 4
Unix and it's Variants 4
What is vulnerable 4
Description 4
Variants 4
Part 2 – The Attack 6
Network Description/Network Diagram 6
Protocol Description: SSH 7
How SSH works 7
How the Exploit Works 8
Attack Diagram 9
Scanning 9
The Attack 10
Covering the tracks 12
Attack Signature/Attack Detection 13
Log files 13
Preventative Measures 14
Sshd updates 14
Sshd trust modifications 15
Obfuscation 15
Part 3 – Incident Handling Process 15
Preparations 15
Initial system hardening 16
Encrypted protocol – SSH only, no telnet 16
ACL Setup 16
Remote logging 16
Log file monitoring 17
Preparations: Policies and Procedures 17
Identification 17
Containment and Eradication 19
Recovery 23
Root Cause: 24
Proximate Cause: 24
Lessons Learned: 24
Follow-up and Corrective Actions 24
Conclusion 27
Sources List 28
Attachment A: Diff Listing and discussion 29

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Introduction

In this paper I will dramatize a fictitious incident. I will apply several concepts discussed in the SANS Incident Handling and Hacker training that to me seem especially significant; the combination of tools and methods involved in a successful intrusion, the points of decision and risk management in the response, the phenomenon of the false sense of security, and the truly essential nature of incident prevention. No incident I have yet handled has had all of these aspects. I also wish to explore the seriousness of the threat posed by the exploit I choose to research, and so I am choosing what seems to me to still be a serious threat: SSH crc32 vulnerabilities.

This threat has many aspects. At this writing, scanning for SSH is at a fever pitch on the Internet, port 22 is number 4 out of the top 10 ports scanned[1], and according to one source[2], almost 20% of all SSH daemons are still vulnerable to this exploit. In a documented exploit, even scanning for a single vulnerable version yielded a 5% vulnerability rate.[3] In addition, a set of tools has been discovered in the last few months that make this vulnerability very easy to exploit. The conditions are ripe for the same methods to be used in the propagation of worm code, and there is some evidence that this is already occurring[4]. This is all happening despite the fact that the vulnerability and associated patches have been know for nearly a year – a disturbing sign of neglect by many system administrators.

Many high-quality analyses of this vulnerability and it's associated exploit code already exist. I will summarize those findings here, and focus on the processes used by the attackers and defenders, and illustrate the decisions taken, and reasons for them. My hope is that by presenting this vulnerability, the exploit, and appropriate countermeasures and incident handling procedures, I will contribute to my own education as well as the learning of my colleagues in the Information Security profession.

In addition, I will tie in aspects of the security incident handling process with the more general Incident Management process as described in the ITIL[5], and link the best practices approach to quality with that of security.

Part 1 - The Exploit

Name

CVE-2001-0144

The Common Vulnerabilities and Exposures (CVE)[6] dictionary defines this vulnerability as "CORE SDI SSH1 CRC-32 compensation attack detector allows remote

---

[1] See http://www.dshield.org/topports.html for current list
[2] See http://www.citi.umich.edu/ssh/
[3] See http://staff.washington.edu/dittrich/misc/ssh-analysis.txt, in which 25,386 unique hosts were scanned for OpenSSH 2.1.1, and 1244 vulnerable hosts were found (approximately 5%).
[4] See http://www.vnunet.com/News/1127965
[5] See the Information Technology Infrastructure Library: http://www.itil-itsm-world.com/

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

attackers to execute arbitrary commands on an SSH server or client via an integer overflow."
In simpler terms, this means that some versions of Secure Shell version 1 (SSH1) are vulnerable to exploitation of a buffer overflow. When "arbitrary code" is executed, that is a bad thing.
The actual exploit of this vulnerability that I will characterize is known as x2. It is a coded program that is used to attack systems with this vulnerability.

Operating Systems

Unix and it's Variants

Since the vulnerability is in the ssh1 code itself, and is not specific to one OS, multiple Operating systems are potentially vulnerable. The vulnerability thus can exist in any OS version capable of running the vulnerable code, including, but not limited to:
Linux (Red Hat, Debian, SuSe, etc)
BSD
Open BSD
SCO, etc.
Solaris

What is vulnerable

Secure Shell version 1 (SSH1). There are several versions utilizing the vulnerable source code. This list is from the original vulnerability statement[7]:

- SSH 1.2.24 - 1.2.31 (ssh.com) -- all versions to
- F-SECURE SSH 1.3.5 - 1.3.10
- OpenSSH prior to 2.3.0 (unless SSH protocol 1 support is disabled)
- OSSH 1.5.7 (by Bjoern Groenvall) and other ssh1/OpenSSH derived daemons

Note that sshd (the daemon) is what is vulnerable. The client software is not, in itself, subject to this compromise, and is therefore not included in this list.

Description

The vulnerability this exploit uses was exposed by Michal Zalewski on the bindview website on February 8th, 2001. The exploit takes advantage of a programming error in an attack detection routine. A 16 bit integer was used where a 32 bit integer would have been appropriate. The result is a stack overflow error.

Variants

As many opensource (and even some commercial versions) of sshd leverage the vulnerable code, there are several possible variants of the same basic vulnerability. In addition, there are several possible ways to exploit the overflow, depending upon the code pushed onto the stack.

---

[6] See http://www.cve.mitre.org
[7] See http://razor.bindview.com/publish/advisories/adv_ssh1crc.html

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

While this is technically a challenge to perform in practice, the vulnerability has been made very easy to exploit through the use of automated tools, notably the x2 tool. This tool allows an intruder to simply specify the host to attack, the version of SSH (from a list specifiable as an input file), and a simple password. The attack proceeds and, if successful, presents the attacker with a root shell.

There are several variants of the tool in use in the wild. Some of these appear to be derivatives of a leaked version of a "research" coding project by the TESO Security group, judging by certain markers left in the code. It is likely that, as TESO's work is available to a group of coders, these individuals will apply the concepts and methods in the fabrication of new tools.

There is some evidence that semi-automated attacks are being launched, i.e., rootkits and associated tools are being used for target selection, and may be being modified to rapidly compromise multiple hosts.

One note: while I could find sample exploits of the noted vulnerability[8], I was unable to track down a version of x2 for download. It seems that the cracker community is hoarding the code, which is part of a new trend growing out of recent debate. There is also some evidence that it may be being sold or traded. It may be that the extreme ease with which the code may be used to exploit vulnerable systems is inhibiting the general release of the code to the "script kiddie" community, but it may also be part of a growing trend in obfuscation and secrecy among crackers. The x2 code itself is highly obfuscated and resistant to analysis[9].

[8] See http://packetstorm.widexs.nl/0102-exploits/ssh1.crc32.txt

[9] See http://www.incidents.org/papers/ssh_exploit.pdf for an excellent, if preliminary, discussion of x2, complete with packet traces and a hacking session in the lab, by Rob Lee. I have used his analysis in preparing this section.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Part 2 – The Attack

Network Description/Network Diagram

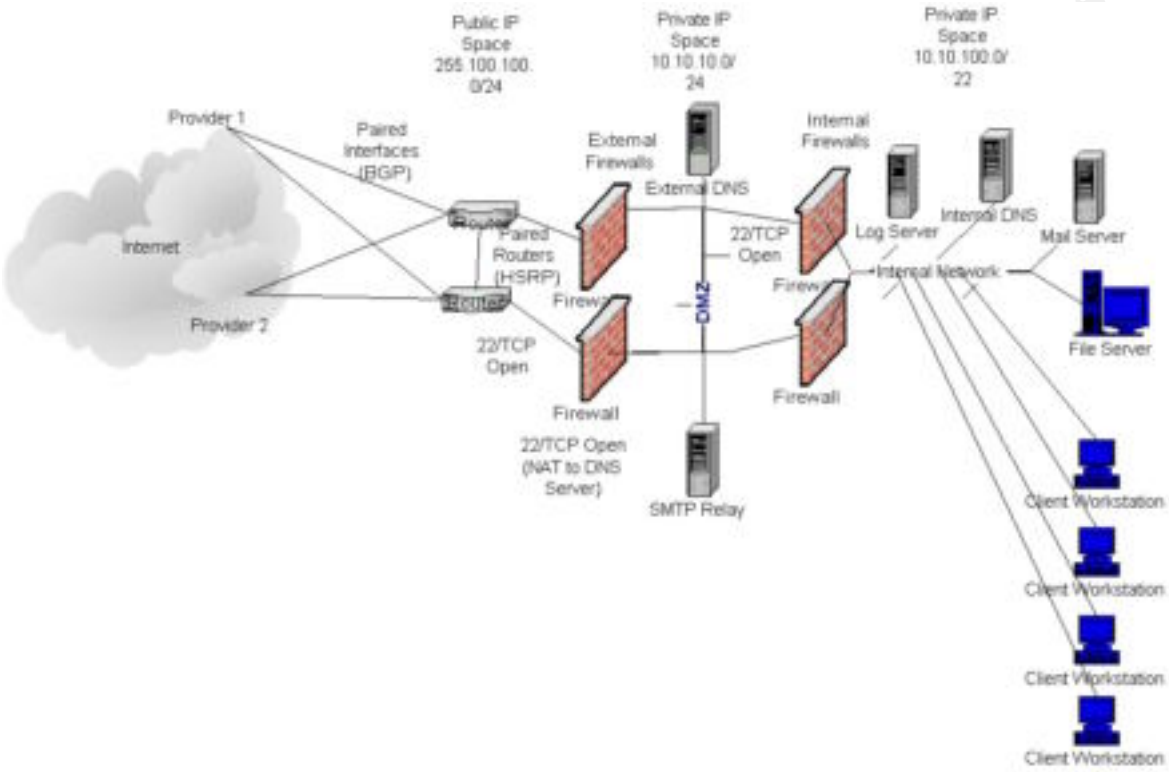

Figure 1. The Network

Figure one shows the company's network. It is set up for high-availability, with redundant network links, and external and internal router and firewall pairs. Functionally, each set of routers and firewalls can be considered a single device.

The routers are Cisco 7200 series, IOS 12.0. They are running HSRP between them, and BGP on the external interfaces, which is necessary to allow failover in the case of failure of a provider's infrastructure.

The four firewalls pictured are Cisco PIX, model 525. The software is version 5.2. They are running in failover mode (one is primary, secondary takes over when primary fails).

The servers are configured as follows:

External DNS, SMTP Relay:
- VA Linux Full-On 2u, running debian Linux, Potato, kernel 2.1 series

Log Server, Internal Mail Server
- Sun Ultra 80, Solaris 2.7

Internal DNS
- Generic 1u Intel Celeron, running debian Linux, Potato, kernel 2.1 series

The ACLs on the external routers and firewalls allow SSH inbound (TCP port 22). No restrictions on source IP address are in place. The only destination hosts allowed are the external DNS and SMTP relays, however. This is put in place to allow

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

management of the network components (routers and firewalls), which are accessible via telnet (port 23) on the internal interfaces only, or via direct serial connections to the DNS and SMTP relay hosts. Some network administrators need to work from home, and made ssh remote access a requirement.

Relevant External ACL (Cisco):
access-list 101 permit any 255.100.100.5 0.0.0.0 eq 22
access-list 101 permit any 255.100.100.6 0.0.0.0 eq 22

Relevant External Firewall Rules (Cisco PIX):
static (dmz,external) 255.100.100.5 10.10.10.5 netmask 255.255.255.255
static (dmz,external) 255.100.100.6 10.10.10.6 netmask 255.255.255.255
conduit permit tcp host 255.100.100.5 eq 22 any
conduit permit tcp host 255.100.100.6 eq 22 any

The internal firewall allows SSH to the servers in the DMZ from the internal network as well. In fact, SSH is allowed in both directions, with no restrictions across the internal firewalls.

Relevant Internal Firewall Rule (Cisco Pix):
conduit permit tcp any eq 22 any

The syslog service is running on a host located on the internal network (the Log Server in figure 1). This host is at IP address 10.10.100.6. The internal firewalls allow traffic to this server to be initiated from any host in the DMZ, thus allowing syslog information to be captured from the External DNS, SMTP Relay, and external and internal firewalls. Note that this is not set up on the external firewalls, so syslog on the boundary routers is not enabled. The necessity of opening this hole in the external firewall was deemed too great a risk.

Relevant Internal Firewall Rule (Cisco PIX):
conduit permit udp host 10.10.100.6 eq syslog 10.10.10.0 255.255.255.0

Protocol Description: SSH

How SSH works

Using TCP/IP, a transport layer called SSH-TRANS is used to make the initial connection from a client system to a host server running an sshd daemon. SSH then makes subsequent user authentication (SSH-USERAUTH) and finally connection (SSH-CONNECT) requests, with each request dependent upon the success of the previous request.

The communication is encrypted at all layers using asymmetric public-private key encryption. The server and client exchange session keys in the SSH-TRANS layer at connection time, and again every so often to support the encryption. The server's public key is used to support initial connection, and if it is available to the client a priori it need

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

not be transferred at connect time. Best practices and the relevant IETF draft[10] spec states that clients must know the server's public key before connecting, and have it associated with the server name, or have this pairing managed by a certificate authority. In practice, however, this is difficult to manage, and most servers utilize an option to disable the association checking the first time a user connects. This allows users to simply connect to a server with no knowledge of its key information, which is very convenient. This practice, however, defeats at least some of the purpose of using the key-pair method in the first place. Using ssh in this way contributes to a false sense of security – encryption alone is not more secure.

How the Exploit Works

The exploit makes use of a very specific vulnerability exhibited by specific versions of the sshd daemon. The vulnerability stems from what the bindview advisory calls "insufficient range control calculations" in a function called detect_attack(), ironically, a function designed to prevent cryptographic attacks on the daemon in the connection layer. The function uses a 16-bit integer variable in a calculation with a 32-bit variable. The result is a memory allocation error that allows an attacker to push code onto memory and eventually execute it.

As the results of the buffer overflow are dependent upon initial encryption settings, it takes several iterative passes to "hit" this vulnerability with very carefully crafted data. The attacker needs to make these passes, and record the results at each crash, using the information gained to alter the attack. Eventually, the attacker may break in, essentially disrupting the SSH-TRANS layer during the key exchange process.

All user authentication SSH-AUTH and connection layer SSH-CONNECT safeguards are bypassed. The attacker is given a root shell.

So, programmatically, we have:

- Attacker sends crafted code to target system.
- Vulnerable service crashes on target system, resets connection.
- Attacking system records result, changes settings to attempt exploit again at another address in memory, and repeats until success is detected.

There is an automated code package called x2 that carries out this process. Using it requires only the code and the password ("thisisnotyourexploit"). It is extremely easy to use. When it detects success, it sends a message ("You are in") to the attacker's console.

Note that at this writing there are no postings of the x2 code showing up in common search engines or exploit archives.

This exploit is very difficult to run manually, given the analysis required at each step, and the need to make multiple trial attempts before success is achieved. Without a programmed exploit, performing more than a trivial buffer overflow (i.e. doing more than crashing sshd) would probably not be practical.

Note that key exchange must be permitted for the attack to work. SSH can be configured to allow only known, trusted hosts to progress to this point in the SSH-

[10] See http://www.ssh.com/tech/archive/secsh/architecture.txt

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

TRANS layer, so restricting trust in the configuration of sshd is an effective countermeasure.

Attack Diagram

Scanning

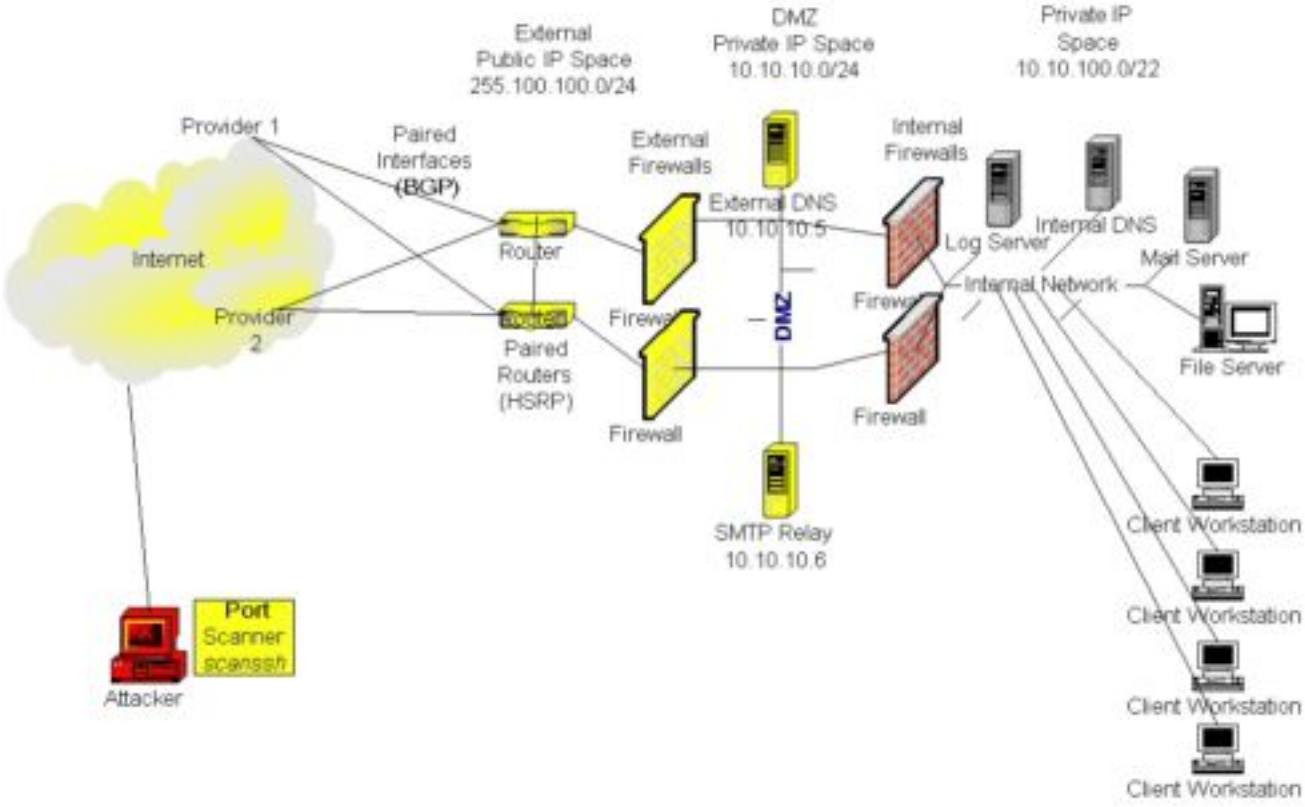

Figure 2: Scanning

The attacker scans the company's public IP address range[11], and discovers port 22 is open on the DNS server and SMTP relay. So far the attacker has done nothing illegal or harmful, and cannot really be distinguished from a legitimate user.

The attacker sends packets that simply open a connection on port 22 and read the header returned:

3-way handshake:

22:18:34.628175 attacker.32771 > victim.22: S 2493461027:2493461027(0) win 5840 <mss 1460,sackOK,timestamp 125605[|tcp]> (DF) (ttl 64, id 55430)

22:18:34.629615 victim.22 > attacker.32771: S 2504205473:2504205473(0) ack 2493461028 win 5792 <mss 1460,sackOK,timestamp 203961[|tcp]> (DF) (ttl 64, id 0)

22:18:34.629925 attacker.32771 > victim.22: . ack 2504205474 win 5840 <nop,nop,timestamp 125605 203961> (DF) (ttl 64, id 55431)

Header Info Returned:

[11] The attacker may use a number of programs, such as nmap, but in this case let us assume they are using scanssh, available at http://www.monkey.org/~provos/scanssh/

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

22:18:34.635442 victim.22 > attacker.32771: P 2504205474:2504205499(25) ack 2493461028 win 5792 <nop,nop,timestamp 203962 125605> (DF) (ttl 64, id 14363)
22:18:34.635809 attacker.32771 > victim.22: . ack 2504205499 win 5840 <nop,nop,timestamp 125605 203962> (DF) (ttl 64, id 55432)
22:18:34.635960 attacker.32771 > victim.22: P 2493461028:2493461056(28) ack 2504205499 win 5840 <nop,nop,timestamp 125605 203962> (DF) (ttl 64, id 55433)
22:18:34.635997 victim.22 > attacker.32771: . ack 2493461056 win 5792 <nop,nop,timestamp 203962 125605> (DF) (ttl 64, id 14364)
Tear Down connection:
22:18:34.636030 attacker.32771 > victim.22: F 2493461056:2493461056(0) ack 2504205499 win 5840 <nop,nop,timestamp 125605 203962> (DF) (ttl 64, id 55434)
22:18:34.639171 victim.22 > attacker.32771: F 2504205499:2504205499(0) ack 2493461057 win 5792 <nop,nop,timestamp 203962 125605> (DF) (ttl 64, id 14365)
22:18:34.639488 attacker.32771 > victim.22: . ack 2504205500 win 5840 <nop,nop,timestamp 125606 203962> (DF) (ttl 64, id 55435)
22:18:34.640501 victim.22 > attacker.32771: R 2504205500:2504205500(0) ack 2493461057 win 5792 <nop,nop,timestamp 203962 125606> (DF) (ttl 64, id 14366)

The attacker sees simply:
[root@attacker /root]# scanssh victim
victim SSH-1.99-OpenSSH_2.2.0p1

There is nothing here that would trigger an IDS or look suspicious in a log file. Some versions of sshd will not even log such an incomplete connection by default.

The attacker can determine from the sshd header[12] that the sshd is a vulnerable version, and select the victim for attack based on this. The attacker can now prepare code to use later, such as a replacement version of sshd. This procedure allows attackers to take the time to prepare very carefully if they need to, including downloading source code and modifying it to suit their purposes.

The Attack

All the attacker needs to do is run x2 against the DNS server. In a matter of minutes, the attacker has a root shell on this server. The attacking machine makes a large number of connections to the victim, eventually zeroing in on the appropriate offset to use, which contribute to it's being identifed (see signature section).

Running the exploit is simple. One must simply type:

sshd-exploit –t7 <victim IP address>

[12] Unmodified sshd daemons respond to connect requests with a version header (e.g. SSH-1.99-OpenSSH_2.2.0p1 as above). This header actually contains two interesting pieces of information: the first set of numbers (1.99 in this case) indicates that the ssh1 protocol will be supported (1.xx). If the number begins with a 2, then only ssh2 is supported. The second set is the version number, and is previous to Openssh 2.3.0, and as such vulnerable.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

password: thisisnotyourexploit

Eventually, you are presented with a root shell.

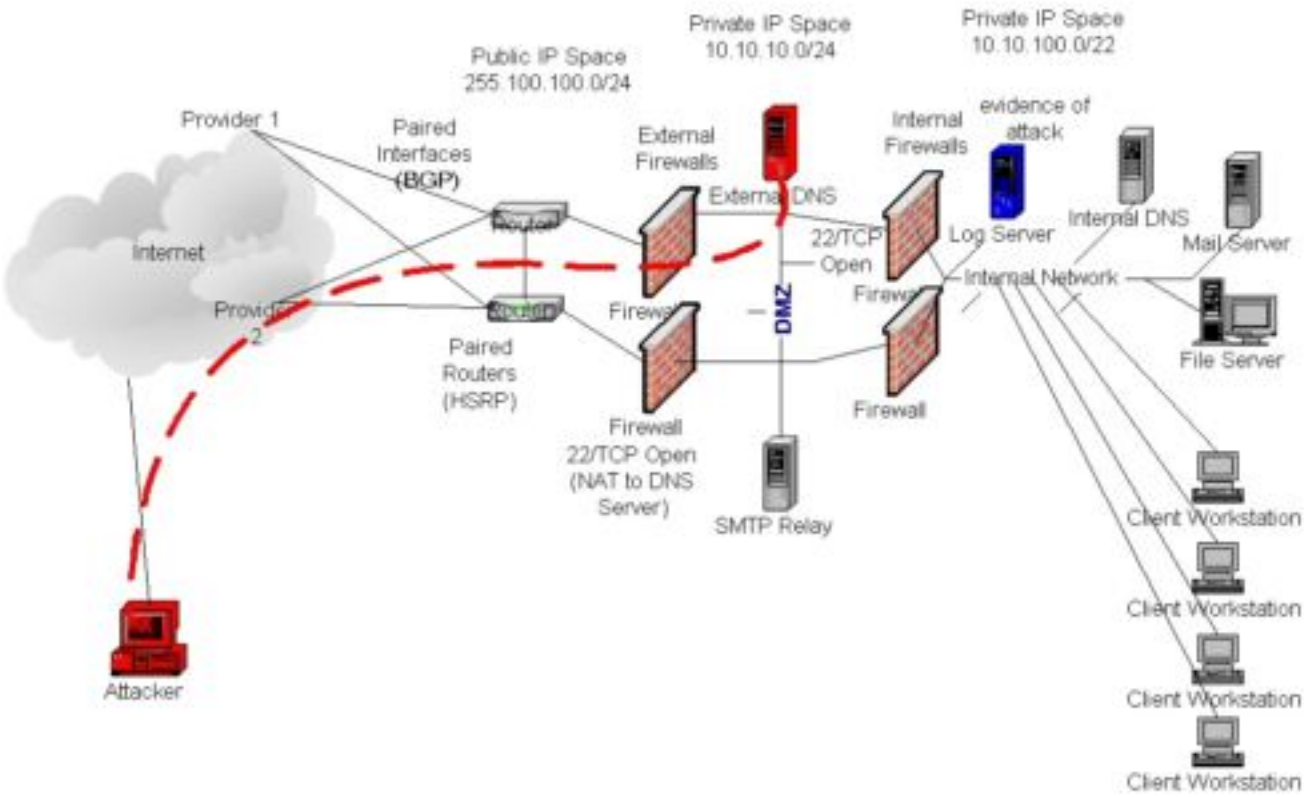

Figure 3: Compromise

The first thing the attacker does is to attempt to determine the version of UNIX or Linux installed. This is fairly obvious, as each OS has certain characteristics, but the presence of the file /etc/debian_version leaves the attacker in no doubt that the system is indeed running debian linux. This file is present on all debian systems, and it's content is the version and revision numbers.

The attacker must assume that the intrusion will be detected, or at least suspected, and that the vulnerability will eventually be removed. In order to keep access, therefore, the attacker must be able to log in again in a more normal and less detectable fashion. The external firewalls and routers are still blocking most useful ports except 22, however, and while it may be possible to change this, it is difficult to do so undetectably and quickly, especially without administrative passwords to the firewalls.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

The attacker instead installs code, which changes sshd to allow undetected access[13]. This ensures that s/he will be able to log in later without appearing suspicious.

Note: The attacker must themselves be well prepared. It is no easy feat to modify and recompile someone else's source code so that it actually works as you intend. The advent of rootkits has made the process easier, but the attacker still must have a selection of these rootkits and other software to choose from, and be intimately familiar with it's features in order to modify it for use during an attack. The software must be made available for download to the victim system, something that must be prepared ahead of time.

Covering the tracks

The attacker determines that the server is logging activity locally and on another host with syslog[14]. As the exploit does log signature lines to syslog, the attacker must expunge these entries.

To accomplish this, the attacker transfers a "cleaner" program[15], complied ahead of time to save time during the initial attack[16]. Running this deletes the telltale syslog entries locally.

This leaves the problem of the log server, with information still on it that the attacker wants to remove. A second intrusion is therefore necessary to complete the removal of telltale signs.

The attacker determines, through simple scanning[17], that the remote syslog server is accepting ssh, but is running a more recent version, and the x2 exploit will not work. The attacker attempts to gain a valid userID and password, hoping that the user names and passwords will be the same on both the compromised and new target systems.

---

[13] One such method is to use ssh replacement code, which may be found at: http://packetstorm.mirror.widexs.nl/UNIX/penetration/rootkits/ssh-1.2.27rk.diff. We will assume the attacker simply replaces the sshd code with a modified version, compiled for Linux. Note that the new sshd will still function normally, but will allow the attacker to enter a special "magic" password, gaining a root shell with no logging of the connection, or logging that obfuscates the fact that a connection was made (e.g. "connection closed" when connection is opened).

[14] The attacker can determine this by checking the /etc/syslog.conf file for a loghost entry (one starting with @).

[15] See Logcleaner zap3.c at: http://packetstorm.mirror.widexs.nl/UNIX/penetration/log-wipers/logcleaner-0.3.c

[16] The attacker can also get around the requirement to pre-compile software for specific versions of the target UNIX/Linux OS if the compiler (gcc in this case) is installed on the target system. This is a good reason to remove the compiler, though it will likely only frustrate a minority of attackers, forcing them to prepare more binaries in advance. Compiling programs often depends on library code being present, so removal of unused libraries is also prudent if gcc is removed.

[17] Telnet to port 22 to see the sshd version header.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

The attacker transfers the /etc/shadow file from the compromised host with ftp[18], and begins running a decryption algorithm[19] on their own system in an attempt to decipher the passwords.

I will assume that this is as far as the attacker gets before the response begins. If they had had more time, the attacker might have taken several additional steps, such as:

- Installing a sniffer on the server – useful for determining usernames and password if telnet is used
- Attempting to compromise another server such as the SMTP relay
- Scanning the DMZ from the compromised server
- Attempting to compromise the external or internal firewalls

Attack Signature/Attack Detection

Fortunately for the defense, the attacker's methods are detectable in several ways. I will summarize these here.

Log files

Syslog:
The buffer overflow in vulnerable versions of sshd can generate several syslog messages, notably:
Nov 1 18:46:47 victim sshd[9518]: fatal: Local: Corrupted check bytes on input.
Nov 1 18:48:08 victim sshd[9586]: fatal: Local: crc32 compensation attack: network attack detected[20]

In addition, if sshd is set to log connection attempts, numerous attempts will be recorded in the context of the attack, which may (if the system is not very busy) be possible to detect as unusual, or at least to recognize as such forensically.

SNORT:
The SNORT intrusion detection system lists several rules in the exploit rules database[21] for ssh crc32 buffer overflow detection. If the defenders had been running SNORT with a current rule set, the rules listed here would have generated alerts:

---

[18] ftp being installed and working, and the attacker having prepared and ftp server for use during the attack. This leaves no telltale executable file on the compromised server, for example netcat.
[19] For Unix/Linux, the attacker can user crack, or John the Ripper, for example. See http://www.deter.com/unix/#unix for a version of crack.
[20] Copied from http://staff.washington.edu/dittrich/misc/ssh-analysis.txt as an example.
[21] Available for download at: http://www.snort.org/downloads-other.html#5 See SNORT documentation on this site for implementation details.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

alert tcp $EXTERNAL_NET any -> $HOME_NET 22 (msg:"EXPLOIT ssh CRC32 overflow /bin/sh"; flags:A+; content:"/bin/sh"; reference:bugtraq,2347; reference:cve,CVE-2001-0144; classtype:shellcode-detect; sid:1324; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 22 (msg:"EXPLOIT ssh CRC32 overflow NOOP"; flags:A+; content:"|90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90|"; reference:bugtraq,2347; reference:cve,CVE-2001-0144; classtype:shellcode-detect; sid:1326; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 22 (msg:"EXPLOIT ssh CRC32 overflow"; flags:A+; content:"|00 01 57 00 00 00 18|"; offset:0; depth:7; content:"|FF FF FF FF 00 00|"; offset:8; depth:14; reference:bugtraq,2347; reference:cve,CVE-2001-0144; classtype:shellcode-detect; sid:1327; rev:1;)

Tripwire:
An integrity checker such as Tripwire[22], properly configured to include the sshd daemon code, would detect not the initial compromise, but would detect the modification to the sshd daemon itself. This software works by creating a database of hash codes from selected files on the system. Running a check of the database reveals all changed files. Thus the timely detection of unauthorized change is dependent upon frequent checks of the systems.

Preventative Measures

Sshd updates

Generally, systems and security administrators need to keep up on the security issues with the software they are using, and update it accordingly to keep ahead of issues of this kind. In this case the vendor response was prompt, and a new version was quickly produced and packaged. All that remained was for the administrators to upgrade and test the new package on their systems.
The debian linux distribution supports a simple to use update facility, apt-get. Once this facility is configured to use the appropriate code repositories, one need merely type "apt-get install sshd" from a root shell to install/upgrade ssh to the latest version.

Protocol version 2 only support

Setting sshd to support version 2 only, and not fall back to version 1, would remove vulnerability to this exploit even without upgrading the code. Note that many MS Windows ssh clients do not correctly support ssh2[23], however, so the step of allowing only ssh2 should be taken only after testing access from all client software versions.

---

[22] Commercial version, see: http://www.tripwire.com/, opensource version, see: http://www.tripwire.org/. Both are well-maintained, very useful software.
[23] One notable (and free) exception is the Putty client. See http://www.chiark.greenend.org.uk/~sgtatham/putty/

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Sshd trust modifications

While it may be somewhat painstaking, it is possible to restrict the hosts allowed to open a ssh (port 22) connection to a given instance of sshd. This is done with the /etc/hosts.allow listing of permitted hosts[24].
The main disadvantage of this approach is that the remote access to the restricted host is then only possible from hosts specifically allowed (which is kind of the point: it's a two-edged sword). Administrators attempting to log on from Internet cafes in Prague may well be frustrated, unless they are able to get their temporary IP address added to the allowed list. This is fairly easy to work-around, however, mostly by arranging appropriate coverage by support personnel, and requesting static IP addresses for the home office Internet connections of support personnel. Of course, there will be additional ongoing maintenance of the hosts.allow file.
Note: allowing only clients with appropriate certificates (turning on Public Key Authentication, turning off Password Authentication) would additionally guard against password guessing (brute force attacks). This is only an ssh2 option, though, so it does not apply to the exploit in question.

Obfuscation

One could simply set sshd to listen on a port other than 22, or forward a port other than 22 from the external firewall to port 22 on the target host running sshd. Note that the attacker in this case scans with a tool that looks only for port 22/TCP – a targeted scan. Simply moving the port elsewhere would have avoided detection, and hence the attack. Recall that the purpose of having the service running on an external interface in the first place is to allow a select group of administrators remote access. It should not be difficult to inform them of the new port to use, and adjust their client configurations appropriately. The disadvantages of modifications such as this is in maintenance and documentation, but this should be kept up in any case. Another point is that this will only stop the attacker who selects the target system somewhat at random via scanning, and will not necessarily do much to foil someone who is determined to get into a particular network. That person may scan all ports, for example.

Part 3 – Incident Handling Process

Preparations

This team did not do a complete job of preparation for handling of incidents such as this. This is a fact which came back to haunt them as they went through the incident handling process, and affected their ability to respond. They were somewhat off their guard, as so many of us seem to be, but as this is a fictitious situation, I will give them credit for taking some prudent steps, to make the learning opportunities more interesting as they discover what they did not do.

Note that in this section I will be drawing from some lab work, my own experience, and that of my colleagues. I will not relate any exact experiences in the

[24] See the manual entry for sshd (man sshd) for requirements, also tcpwrappers documentation.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

interests of maintaining anonymity of sources, but the reader should know that most of this stuff did, in fact, happen, albeit to several different people in several different situations.

Initial system hardening

Some basic system hardening steps such as removing unused software packages and closing unused service ports were done. Appropriate patch levels of all software at the time of install were used. However, there was no process in place to update the software except by functional necessity (i.e. only if a needed feature was not supported was the system upgraded). Backups of all initial configurations were made and archived on spare disks to facilitate quick recovery from disk failure.

Encrypted protocol – SSH only, no telnet

Allowing remote access was specified as a requirement, so accommodation to security was made at design time by planning for ssh access, and disabling telnet to all systems.
Remote access to the servers in the DMZ was set up with ssh, which was assumed to be secure enough in itself to allow remote network administration. The alternative of allowing telnet to the routers and systems from outside the firewall was rejected as too insecure. The assumption was made that ssh was secure, and very little was done to ensure this beyond installation of the package.

ACL Setup

The following basic guidelines were agreed upon in setting up the router ACLs and Firewall rule sets:

- No system should be generally accessible from outside it's own IP subnet
- If access is allowed to a system from outside it's subnet, then the access shall be:
    - Filtered at the port/service level
    - Restricted to and from known hosts

Note that the fact that this was not strictly followed contributed to the success of the attack. Had only trusted hosts been allowed to connect, the attacker would not have been able to use this particular exploit.

Remote logging

Following the rule of "prevention is ideal, but detection is a must", the defenders set up a syslog server to allow remote capture of syslog information, and had set all systems to write logs there, where possible. One notable exception to the list of systems sending data to the syslog server were the boundary routers. It was decided that the opening of the external firewall to allow syslog[25] was too dangerous, since it could allow attackers to perform denial of service attacks on the syslog server itself.

---

[25] This would be simply allowing UDP port 514 through the outer firewalls. This is not a significant risk IF the firewalls also restrict the source address to be that of the external

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Log file monitoring

In addition, a simple utility called swatch[26] was used to parse through the syslog files continuously, and to send email to system administrators when certain expressions were matched. This was done in lieu of creating a complete intrusion detection system using dedicated software such as SNORT. The swatch utility was set to match only very simple strings, (like "fatal") however. Only one administrator knew how to configure it, and they did not document or disseminate this knowledge. This led to a lack of understanding of what the resulting messages and alerts meant, and a corresponding lack of maintenance of the utility' s configuration.

Preparations: Policies and Procedures

Some basic policies were put in place, but best practices in this area had been largely ignored.
The following steps were taken before the incident occurred:

- Warning banners
  - Each system displays a banner stating that the machine in question is only for the use of company employees and contractors, and that privacy is limited on this machine. It also reminds employees that they are bound by the terms of the security policy.
- A security policy was in place
  - A corporate policy dictating the limits of the expectation of privacy, the terms of appropriate use, and legal liability of the users was written, and acknowledged by all employees.
- Security Procedures were in place
  - Certain procedures were enforced, such as the maintenance of separate passwords for internal and external systems, password complexity and length requirements.
  - Deployment of new systems always included a security check as part of release management.

Identification

The incident was identified via it's syslog signature. A system administrator came in early in the morning after the attacker got in, and read the email sent by the swatch utility, indicating a problem with the External DNS server. The administrator immediately checked the syslog file for the External DNS server and discovered that there were multiple lines of the form:
Nov 1 18:46:47 victim sshd[9518]: fatal: Local: Corrupted check bytes on input.
followed by two lines of the form:

---

router itself. NAT can also be a problem in this case if it is used to mask addresses in such a way that the syslog data coming from multiple devices all appears to have the same source IP address.
[26] See http://www.cert.org/security-improvement/implementations/i042.01.html

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Nov 1 18:48:08 victim sshd[9586]: fatal: Local: crc32 compensation attack: network attack detected[27]

The administrator then logged in to the External DNS Server. No problems were obvious, and access via ssh was certainly the same as always. The administrator checked the local syslog file, and found that the lines that had been flagged by the swatch utility were NOT present. This was the first indicator that something was seriously wrong, and that an incident may have taken place: the initial alarm was correlated with this anomaly. The administrator decided to check the size of the binaries in /usr/bin and /usr/sbin against those of the Internal DNS server, which was a mirror image of the External DNS, albeit with different DNS database entries and a different IP address. The output looked like this:

Internal DNS server:

```
indns:/usr/sbin# ls -la
total 2872
drwxr-xr-x   2 root     root        4096 Jan 21 15:21 .
drwxr-xr-x  13 root     root        4096 Dec 29 16:15 ..
-rwxr-xr-x   1 root     root        5027 Jul 10  2001 MAKEFLOPPIES
-rwxr-xr-x   1 root     root       11048 Sep 18 18:20 accessdb
lrwxrwxrwx   1 root     root           7 Oct  3 15:38 addgroup -> adduser
-rwxr-xr-x   1 root     root       25538 Sep 17 06:23 adduser
-rwxr-xr-x   1 root     root       16459 Sep  6 17:08 apt-setup
.
. (lots of files)
.
-rwxr-xr-x   1 root     root         877 Aug 22 09:33 shadowconfig
-rwxr-xr-x   1 root     root      209724 Jul  2  2001 sshd
-rwxr-xr-x   1 root     root        3984 Apr 25  2001 syslog-facility
```

External DNS Server

```
exdns:/usr/sbin# ls -la
total 2872
drwxr-xr-x   2 root     root        4096 Jan 21 15:21 .
drwxr-xr-x  13 root     root        4096 Dec 29 16:15 ..
-rwxr-xr-x   1 root     root        5027 Jul 10  2001 MAKEFLOPPIES
-rwxr-xr-x   1 root     root       11048 Sep 18 18:20 accessdb
lrwxrwxrwx   1 root     root           7 Oct  3 15:38 addgroup -> adduser
-rwxr-xr-x   1 root     root       25538 Sep 17 06:23 adduser
-rwxr-xr-x   1 root     root       16459 Sep  6 17:08 apt-setup
.
. (lots of files)
```

[27] These example lines copied from Analysis of SSH crc32 compensation attack detector exploit, by David Dittrich (http://staff.washington.edu/dittrich/misc/ssh-analysis.txt)

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

.
-rwxr-xr-x 1 root root 877 Aug 22 09:33 shadowconfig
-rwxr-xr-x 1 root root 212324 Nov 1 20:41:12 sshd
-rwxr-xr-x 1 root root 3984 Apr 25 2001 syslog-facility

A quick comparison revealed a difference in the size of sshd. The administrator did not immediately raise the alarm, or call for help, but proceeded to poke around on the External DNS server for a while, checking settings for sshd, looking at /etc/sshd.conf with the visual editor, vi, and running programs to see if they were working properly. Finally they returned to the sshd difference and checked the version number on the external and internal DNS servers to compare them. They each reported the same version. This confirmed the incident: both a missing set of suspicious syslog lines, and an altered sshd. At this point, the administrator started making calls to management and the rest of the systems administrators.

Containment and Eradication

The administrators and responsible managers convened a meeting to discuss the incident. There were several people called in from around the company, and some schedules were interrupted, as this meeting was made a high priority. One of the administrators was adamant that the system affected be shut down and replaced, since in his opinion a rootkit had been installed, while others were concerned about downtime (no external DNS would mean some Internet users and email might have difficulty reaching the site), and management wanted information to help make a decision about whether to attempt information gathering, and possibly apprehend the culprits and hand them over to law enforcement. In the end, the decision was made to replace the system, since the adamant administrator carried the day. Getting to this point took and hour and a half of wrangling, however.

This approach meant that the team would be tipping their hand, and exposing the fact that the compromise had been detected. This heightened the importance of preserving the existing evidence as carefully as possible, to keep open the possibility of apprehending the attacker, as no more evidence might be gathered.
The action items from the first meeting were:

- Rebuild and replace the DNS Server
- Analyze the compromised server offline for evidence of vulnerability and determine countermeasures
- Look for evidence of other systems having been compromised
- Come up with a timeline for the incident to assist in decision making

The administrators split up into three teams:

1) One team shut down the DNS server and began building a temporary replacement. This was a straightforward task, and not really pertinent to this review.
2) A second team began researching the possible methods that the attacker could have used, and analyzing the compromised server itself for clues, going with the working assumption that a rootkit had been installed on the server. The analysis was of some interest, and was done as follows:

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

a. A complete backup of the system was made using Linux: the server's hard disk was removed and placed in a temporary system with a blank hard drive of like capacity. This system was then booted from a set of Linux Boot floppies. The old source disk was installed as hda, and a blank as hdb. Then it was simply a matter of making a duplicate:

```
# cat  /dev/hda > /dev/hdb
```

The original disk was then stored for potential use as evidence.

b. A "clean" install disk containing the same Linux version as the compromised system was restored from a stock disk image that had been saved when the server was initially installed. A backup of a known good image could also have been used, provided a reasonable check that a backup had not already been compromised was made[28]. This disk was then installed as hdb, and the copy of the compromised disk was installed as hda. The root partitions on each of these were then mounted (read-only for safety) as local volumes. These were then compared at the file level. The procedure was as follows:

Mount the systems root directories (on partition5) as read only

```
# mkdir  /compromised
# mkdir  /clean
# mount –o ro /dev/hda5  /compromised
# mount –o ro /dev/hdb5  /clean
```

Now generate the (long) listings

```
# ls –lR  /compromised  > compromised.listing
# ls –lR  /clean  > clean.listing
# diff compromised.listing clean.listing
```

Note: The floppy boot environment is a little short on disk space. A large root (/) file system on the disks being checked might tend to fill the disk. To get around this, use a RAM disk:

```
# dd if=/dev/zero of=/dev/ram1 bs=1k count=10000
(write 10000 bytes of zeros to a ramdisk device – zero it out for use)

# mke2fs –N 20000 /dev/ram1
(create a file system  on the RAM disk with a lot of inodes)
```

[28] This can be done by comparing the critical binary file sizes and modification dates to those of know good media, such as distribution CDs or digitally signed binary archives.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
# mount –t ext2 /dev/ram1 /mnt
(mount the disk so you can use it)
```

Then run the same commands as above, but send the listings to the /mnt directory instead:

```
# ls –lR  /compromised  > /mnt/compromised.listing
# ls –lR  /clean  > /mnt/clean.listing
```

This yields listings of the files in compromised.listing that differ from those in clean.listing, and their counterparts in clean.listing. This output can be piped to the grep command to look for lines containing specific strings, should there be a large number of differences.

Practical Note: If you mount the disk partition with the root (/) filesystem on it, you will probably get the most interesting files. For completeness, however, you should check the /etc/fstab file to see what partitions the other parts of the filesystem are mounted on, notably /boot, and /opt, which may be of interest, since they are often on other partitions, and contain the kernel (in the case of /boot) and installed software (in /opt). In a multi-disk system, this will be a tedious process of going through each disk.

RAID systems other than RAID 1 (mirroring) are likely to be difficult to evaluate with this method, since one would need to reconstruct the raid array, and a suitable comparative disk or array, and get it to boot up from a floppy (or perhaps a CD-ROM). On a mirrored system, however, it should be simple enough to simply mount one of the mirror pair drives for this purpose.

Should the attacker have used a kernel rootkit, this approach would still find all the modified files without relying on the kernel itself (we are using a different, very small kernel). The disadvantage of this method is, obviously, that the system must be taken offline, and boot floppies prepared.

This approach turned up a few significant file differences, indicating that sshd had been replaced[29], and some of the other related activity. It also showed evidence of the initial investigation by the discovering administrator.

The general research turned up a number of rootkits that might possibly have had the sshd component of them installed, but none that matched the replacement sshd exactly. There were no other files found to have been actually replaced, and there were no kernel differences, so the team concluded that the original assumption that a rootkit was used was incorrect, and that the compromise was perhaps not as serious as first thought.

Focusing on the syslog entries and searching security-related sites, the team discovered that the entries they were seeing were characteristic of a crc32 attack. They also determined that the sshd version they had been using was vulnerable to this type of compromise.

3) The third team looked for evidence of further compromise, and began putting together a timeline for the incident.
    a. Looking for further compromise included the following

[29] For an example of what this looks like, see attachment A.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

- Checking the external firewall logs for evidence of inbound connections to other hosts from outside, and cross-checking with local syslog entries. Aside from the original compromise and some scanning activity, nothing was found. The attacker's IP address was found, and was determined to be a dial-up connection from an ISP in another city.
- Looking through the Internal firewall logs for similar evidence. Connection attempts from the External DNS server to the Internal syslog server on port 22 were seen, but all were terminated before login, according to the log. In itself this was not evidence of a compromise.
- Matching all inbound connections found with legitimate access by privileged users. This was not too hard, as there were few connections during the time investigated, and all were made by the technical team, but a lack of notes meant that most of this was done from memory.
- Coordinating with the second team to determine likely signatures of the method used. This turned up that the attacker had used a crc32 attack on sshd, and had then replaced sshd, but these symptoms were not found on any other systems.

The timeline was simply a listing of all the incident related events, and the associated times and dates. It was the high-level summation of all notes and actions, and the listing of the four W's of Incident Handling: Who, What, Where, and When. It looked like this:

Date:	Time:	Analyst	System(s) Affected	Event
11/1/2001	18:46	Jim	External dns	First evidence of unauthorized access in syslog on syslog server.
11/1/2001	18:48	Jim	External dns	Evidence of successful crc32 attack on external DNS server
11/1/2001	20:41	Jim	External dns	Evidence that sshd was modified at this time (file modified date)
11/1/2001	20:45	Phil	Syslog	Evidence that Syslog server was scanned (Internal Firewall Log)
11/2/2001	06:55	Jim	External dns	Administrator first alerted to possible compromise by reading email
11/2/2001	07:15	Jim	External dns	Administrator logged in to External DNS server, begins investigating
11/2/2001	8:05	Jim	External dns	Administrator completes investigation, calls managers
11/2/2001	11:15	Team	External dns, External SMTP, Firewalls, Syslog	Eradication teams start work
Etc...				

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Having this timeline is useful in decision-making, and in ensuring all persons on a team are working from the same assumptions. It can become a reference in determining the root cause of an incident and in gaining insight about the attacker and their methods. It is also useful as a general practice in non-security related incidents or outages. It is not of particular use in proving guilt, since this depends on specific evidence, but it helps greatly in managing a multi-person effort.

After about two hours the server replacement was done, and the group met again to decide what should be done next. The research team had found the signature from syslog listed in a description of a crc32 vulnerability, and surmised that this was what was used to compromise the system. The third team had found no evidence that other systems were compromised in the same way as the External DNS server had been, and, working with the second team, put the time of initial compromise to be the previous evening. The remaining containment and eradication action items were:

- Identify any remaining systems with vulnerable versions of sshd
- Shutdown and upgrade sshd if it is a vulnerable version and accessible from the Internet
- Have everyone change their ssh passwords

The technical teams divided up the work, taking most of the afternoon to complete it. Other longstanding unresolved technical issues complicated matters, and some users complained about the fact that computer support seemed distracted, and unable to service requests.

At this point the company gossip chain was alive with the rumor: We've been hacked! The management spent valuable time quelling this, and coming up with a sanitized version of events for office consumption. The CEO demanded a briefing early on, and was given incorrect information, which had to be corrected later, embarrassing the technical team manager. More meetings were scheduled, and management grilled all members of the technical team about the security of the network. Trust was damaged internally, and the rumor got around that the network was insecure, and that hackers had gotten in and stolen company secrets.

Recovery

The next day, a new pair of external DNS servers were implemented, replacing the temporary server put in place the day before. The extra server was added as a secondary to the Internic record. This completed the repair of the outage caused by the taking down of the compromised DNS server, and improved the reliability of the external DNS.

In order to provide some assurance that the attacker was not re-penetrating the systems, syslog entries on all external systems were checked every 2 hours for 24 hours by administrators, and the swatch utility was checked for function and accuracy.

All action items from the previous day's meeting were confirmed as completed. An incident report was created to allow the organization to learn from the experience. The incident report consisted of the timeline, the corrective actions taken and to be taken, and the root cause analysis. This is a simple summary of what caused the incident, and is useful for security-related and non security-related incidents. In some

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

cases incidents that are thought to be security-related initially turn out to be due to some other cause, or vice versa, so it is useful to have a general review procedure. The analysis in this case was as follows:

Root Cause:

There was not procedure in place to update operating systems or communications software. This led to the exposure of a critical system to a well-known vulnerability, which was exploited by an unknown attacker. Recovery was adequate, and several weaknesses in the security of the network were dealt with, but no report to the security community was made, nor was any attempt made to trap or even specifically track the attackers.

Proximate Cause:

The lack of a proper trust model (trusted hosts) and the similar lack of any obfuscation or true IDS contributed to the vulnerability becoming exploitable, and not sufficiently recorded for a legal case to be built against the attacker.

Lessons Learned:

The establishment of proactive security procedures for maintenance is necessary, as is the establishment of routine incident handling procedures. Without this basic discipline, even simple incidents can become expensive and easily get out of control.

Follow-up and Corrective Actions

The fictitious team had actually a fairly good experience considering their lack of preparation. I assumed a fairly heavy team, (at least 6 people), with involved managers and fairly high skill levels. The incident shows, however, that several commonly neglected items combined to amplify a relatively small mistake into a huge problem. The IT department had to scramble to save the situation, and management ended up losing confidence in it's technical team and IT management.

The team failed to capture and preserve enough evidence to successfully prosecute the attacker. In part this was due to mistakes made in the incident handling process, but also it was due to a conscious decision on their part not to gather more information, but to simply recover from the compromise and get back to work. In fact, the attacker had not gotten very far, and while their actions were meant to penetrate the security of the network, it would be difficult to prove that they did any significant harm. The risk management decision to rebuild the compromised server (or more accurately to down the existing server while the replacement was being built) was the single most important factor in destroying the chance of catching the attacker.

The following points summarize the problem areas and optimal corrective actions:

- Establishment of a procedure for incident handling
  The initial investigation of the alarm was too casual, as the administrator was not expecting an intrusion problem, and had never actually dealt with anything more serious than an email virus. Thus the lack of a documented procedure, and the

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

associated discipline needed to recognize when to pull out the procedure and follow the steps were critically lacking.

The lack of a procedure led to the following mistakes:

1) The investigating administrator made no attestation. This compromises the credibility of the information they uncover – it is less likely to be admissible as evidence.
2) No notes were taken as to what was done. The administrator simply poked around for a while, and was not careful enough not to disturb the system. Note that this activity showed up in forensic analysis. The investigative techniques were potentially destructive – use of the vi editor, for example, was avoidable, and had the potential for altering files and file access information such as date modified. It was also probably unwise to connect to the system remotely, as this involved execution of the compromised sshd code, which turned out not to be harmful, but might just as easily have been.
3) The administrator did not immediately call for assistance. This further degraded the evidentiary chain, as the testimony of two witnesses is much more compelling as evidence. It was also a tactical error, as a team is likely to make far fewer bad choices than an individual.

To supplement the procedure, the team could also have prepared:

- Updates to the disaster recovery plan to include incident handling. Not needed in this case, but only because the attacker was stymied early in the attack.
- Checklists and documentation for incident handling and evidence gathering. This would have given the teams a clear path, and cut down on the need for meetings and discussion – the worst time to write a policy is when you need it.
- Jump kits, containing the software, backup devices and media for evidence preservation.[30] Better forensic tools could have made the detection job easier, and while the team did a good job with what they had, it was something of a scramble to put it together.
- Related procedures for information sharing with the security community. As these were not in place, there was no report made, and no one else outside the company learned anything from the experience of the team.

- Training of employees in incident handling

The best procedure in the industry is useless if it sits in a file cabinet and is never used. Training is essential for all employees in reporting an incident, and for those individuals likely to discover one in the course of their jobs, incident handling training should be mandatory. In the same way that employees are trained in specialized skills

[30] In this case the disaster recovery equipment had to be used for incident recovery. Such dual-use is good practice, actually. The jump kit should contain at a minimum the boot floppies, documentation on their use, and CDs with known good binaries for comparison or even for boot. Forensic utilities for disk inspection to improve the improvised procedures used could optionally be added.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

and company policy, everyone needs to know how to at least recognize and report an incident. Training for managers is important as well, as I illustrate in the next bullet.

- Management must enforce the use of process and adherence to policy.

All managers of technical departments should be aware of information security, and be committed to processes that improve security over time. Quality enhancement processes and models such as ITIL or ISO 9001[31] are ideal frameworks for organizing good security discipline, and for enforcing it. These models have historically been ignored in smaller technical organizations, but as technical professionals focus on security, process models such as these are gaining more acceptance.

The ITIL in particular is germane in that it categorizes IT management into Service Delivery and Service Support. Greatly simplified, the Service Delivery portion focuses on the planning and management (proactive) tasks involved in running the business of IT, such as Service Level Management, Financial Management, etc. Service Support is generally composed of reactive and situational tasks, including the establishment of a Service Desk, and specifically Incident Management. While ITIL does not (yet) specifically detail *security* Incident Management to the degree SANS does, it's general structure affords this, and will likely be extended in later drafts. In reality, quality processes such as ITIL are intended to allow for all problems to be tracked and evaluated, as a way to improve quality, by building in the review of lessons learned. When properly applied, this process significantly enhances security. Perhaps even more important, it assigns security related infrastructure, expenditures, policies, and procedures the proper significance in the IT organization, ensuring they are not ignored.

Note: not least among the recommendations of standards organizations and security management professionals is the recommendation that management implement *measurable* performance metrics for IT and security staff, with *rewards* for good performance and *consequences* for poor performance. This point is often ignored, and leads to higher turnover than necessary in most technical shops, and resultant significant security issues. Also, a common set of rules cuts down on the destructive rivalry, political maneuvering, and grandstanding so common in corporate environments.

- Assignment of an Incident handling team.

System administrators were asked to do this work in addition to their normal tasks, and there was no leader or go-to person for this situation. This resulted in disruptive and contentious meetings that delayed work on the incident, allowed rumors to start, and unnecessarily raised the stress level of everyone involved. In addition, management was forced to make decisions in the moment that could have been left up to the team, had the team been prepared.

- Establishment of a liaison in local law enforcement

[31] See the International Organization for Standardization: http://www.iso.ch/iso/en/ISOOnline.frontpage. The "ISO Process" is a framework for quality improvement in an organization. While not specific to security processes, it is adaptable to them.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

While in this case evidence was not preserved adequately to enable prosecution, had it been, the company would have had to start it's relationship with law enforcement with an active case, not knowing who the appropriate local contact was, what procedures they would require, etc. The chance of a successful prosecution is therefore reduced.

- Establishment of a maintenance and audit schedule, and procedures to keep software up to date.

This is basic system administration, but requires a discipline rarely seen outside military organizations. It's effect on security is obvious, and in this case it would have eliminated the threat before it became a practical reality: the x2 code was not written until some months after the vulnerability it exploits was published.

- Establishment of a trust model for remote access

In this situation, all hosts were trusted to make an ssh connection from the Internet on a known port. While password security was in place, this practice violated the principle of defense in depth. Not all hosts needed to be trusted, therefore, not all hosts should have been. Also, ssh was used with no requirement for the client to have knowledge of the server's public key prior to connecting, thus negating a significant part of the security mechanism of the protocol.

- Establishment of a robust Intrusion Detection System

The gathering of evidence in this case was painful and slow. Correlating evidence was compromised, and only one set of logs really showed that the attack took place at all. Having an IDS, particularly an IDS that is configured to capture and log packet data, is essential for correlation of the time and nature of the attack.

Conclusion

This incident, though fictitious, serves to illustrate some of the common failings seen in the industry. Despite a robust architecture, excellent equipment and reliable systems and backups, the team was not prepared to really handle an actual intrusion. In the end, they did have some evidence in the backup of the compromised system and the associated log files, but because they were unaware of the proper procedures for handling the incident, this evidence was tainted, and the management was not confident enough to pursue the attacker.

The IT team was happily going about it's normal business, confident that they had paid attention to security. They had implemented encryption. They had a secure logging server, and monitored the logs. The simple omission of a process to update software on the exposed systems allowed an otherwise well-run and highly functional network to become compromised. This false sense of security is something I have observed many times, in many organizations, and usually for similar reasons.

While security is often, in the moment of compromise, a matter of technical detail, in general good security only comes with a fairly mundane focus on process, quality, and discipline. For this reason I hope to raise the awareness of my colleagues in the information security field about organizations like ITIL and ISO and encourage them to contribute to the standardization of the procedural portions of the field.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

The crc32 vulnerability was interesting to study for another reason. In the course of writing this paper, several people I mentioned it to were dismissive of the threat, saying things like “That’s been known for a year! Anyone who didn’t update their software must be living on Mars”. As the current statistics show, however, the issue is still very far from solved. I think the false sense of security issue is very real, not least because of the attitudes we take about “old” threats. Old threats are real, and the exploits based on them are widespread and various. We must look for and fix the vulnerable systems in our midst. It only takes one to ruin your whole day.

Sources List

Web sources:
www.dshield.org
http://www.citi.umich.edu/ssh/
http://staff.washington.edu/dittrich/misc/ssh-analysis.txt
http://www.vnunet.com/News/1127965
http://www.itil-itsm-world.com/
http://www.cve.mitre.org
http://razor.bindview.com/publish/advisories/adv_ssh1crc.html
http://packetstorm.widexs.nl/0102-exploits/ssh1.crc32.txt
http://www.incidents.org/papers/ssh_exploit.pdf
http://www.ssh.com/tech/archive/secsh/architecture.txt
http://www.tripwire.com/, http://www.tripwire.org/
http://www.cert.org/security-improvement/implementations/i042.01.html
http://www.chiark.greenend.org.uk/~sgtatham/putty/
http://www.iso.ch/iso/en/ISOOnline.frontpage
Print Sources:
Thomas A. Wadlow: The Process of Network Security, Addison Wesley, 2000. – For general reference.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Attachment A: Diff Listing and discussion

The process of generating detailed listings of potentially compromised disks and their supposedly clean images is discussed here in somewhat more detail than in the general text. I generated a couple of listings in a lab setting and saved the diff output.

```
root@debian:~# diff disk1.list1 disk1.list2
6,7c6,7
< drwxr-xr-x    5 root     root        20480 Feb  5 05:35 dev
< drwxr-xr-x   47 root     root         4096 Feb  5 05:35 etc
---
> drwxr-xr-x    5 root     root        20480 Feb  3 02:57 dev
> drwxr-xr-x   47 root     root         4096 Feb  3 02:57 etc
```

So, these directories were touched. This could be innocent, but is interesting.

```
407c407
< prw-------    1 root     root           0 Feb  5 05:34 initctl
---
> prw-------    1 root     root           0 Feb  3 02:57 initctl
976c976
< crw-------    1 root     tty        4,   1 Feb  5 05:23 tty1
---
> crw-------    1 root     tty        4,   1 Feb  3 02:32 tty1
987c987
< crw-------    1 root     tty        4,   2 Feb  5 05:24 tty2
---
> crw-------    1 root     tty        4,   2 Feb  3 02:42 tty2
998c998
< crw-------    1 root     root       4,   3 Feb  5 05:20 tty3
---
> crw-------    1 root     root       4,   3 Feb  3 02:32 tty3
1009c1009
< crw-------    1 root     root       4,   4 Feb  5 05:20 tty4
---
> crw-------    1 root     root       4,   4 Feb  3 02:32 tty4
1020c1020
< crw-------    1 root     root       4,   5 Feb  5 05:20 tty5
---
> crw-------    1 root     root       4,   5 Feb  3 02:32 tty5
1031c1031
< crw-------    1 root     root       4,   6 Feb  5 05:20 tty6
---
> crw-------    1 root     root       4,   6 Feb  3 02:32 tty6
1299c1299
< cr--r--r--    1 root     root       1,   9 Feb  5 05:20 urandom
---
> cr--r--r--    1 root     root       1,   9 Feb  3 02:32 urandom
1431c1431
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
< prw-r-----    1 root     adm            0 Feb  5 05:20
xconsole
---
> prw-r-----    1 root     adm            0 Feb  3 02:32
xconsole
4278c4278
< -rw-r--r--    1 root     root          45 Feb  5 05:35 adjtime
---
> -rw-r--r--    1 root     root          45 Feb  3 02:57 adjtime
```

All fairly innocent stuff – you expect these files to change...

```
4324c4324
< drwxr-xr-x    2 root     root        4096 Feb  5 05:30 init.d
---
> drwxr-xr-x    2 root     root        4096 Feb  3 00:29 init.d
4327c4327
< -rw-------    1 root     root          60 Feb  5 05:20
ioctl.save
---
> -rw-------    1 root     root          60 Feb  3 02:32
ioctl.save
4350,4351c4350,4351
< -rw-r--r--    1 root     root         365 Feb  5 05:20 motd
< -rw-r--r--    1 root     root          65 Feb  5 05:35 mtab
---
> -rw-r--r--    1 root     root         365 Feb  3 02:32 motd
> -rw-r--r--    1 root     root          65 Feb  3 02:57 mtab
4357c4357
< drwxr-xr-x    2 root     root        4096 Feb  5 05:30 pam.d
---
> drwxr-xr-x    2 root     root        4096 Feb  3 00:29 pam.d
4376c4376
< -rw-r--r--    1 root     root          55 Feb  5 05:20
resolv.conf
---
> -rw-r--r--    1 root     root          55 Feb  3 02:31
resolv.conf
4390c4390
< drwxr-xr-x    2 root     root        4096 Feb  5 05:30 ssh
---
> drwxr-xr-x    2 root     root        4096 Feb  3 00:28 ssh
```

SSH directory is modifed? Hmmm...

```
4945c4945
< -rw-r--r--    1 root     root           0 Feb  5 05:35 ifstate
---
> -rw-r--r--    1 root     root           0 Feb  3 02:57 ifstate
5241c5241
< -rw-------    1 root     root         526 Feb  5 05:30
ssh_host_key
---
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
> -rw-------    1 root     root          526 Feb  3 00:28
ssh_host_key
5397c5397
< drwxr-sr-x    2 root     staff        4096 Feb  5 05:27 tomas
---
> drwxr-sr-x    3 root     staff        4096 Feb  3 02:53 tomas
5400c5400,5405
> -rw-r--r--    1 root     staff       16514 Jan 26 22:32
net1.txt
> drwxr-sr-x    2 root     staff        4096 Jan 29 06:24 sshd
> -rw-r--r--    1 root     staff        6888 Jan 29 07:15 take2
>
> /home/tomas/sshd:
5719,5726c5724,5731
< -rw-r--r--    1 root     root         3817 Feb  5 05:20
modules.dep
< -rw-r--r--    1 root     root           31 Feb  5 05:20
modules.generic_string
< -rw-r--r--    1 root     root           73 Feb  5 05:20
modules.ieee1394map
< -rw-r--r--    1 root     root           81 Feb  5 05:20
modules.isapnpmap
< -rw-r--r--    1 root     root           29 Feb  5 05:20
modules.parportmap
< -rw-r--r--    1 root     root        15975 Feb  5 05:20
modules.pcimap
< -rw-r--r--    1 root     root           24 Feb  5 05:20
modules.pnpbiosmap
< -rw-r--r--    1 root     root          189 Feb  5 05:20
modules.usbmap
---
> -rw-r--r--    1 root     root         3817 Feb  3 02:31
modules.dep
> -rw-r--r--    1 root     root           31 Feb  3 02:31
modules.generic_string
> -rw-r--r--    1 root     root           73 Feb  3 02:31
modules.ieee1394map
> -rw-r--r--    1 root     root           81 Feb  3 02:31
modules.isapnpmap
> -rw-r--r--    1 root     root           29 Feb  3 02:31
modules.parportmap
> -rw-r--r--    1 root     root        15975 Feb  3 02:31
modules.pcimap
> -rw-r--r--    1 root     root           24 Feb  3 02:31
modules.pnpbiosmap
> -rw-r--r--    1 root     root          189 Feb  3 02:31
modules.usbmap
5974,5975c5979,5980
< drwxr-xr-x    2 root     root        12288 Feb  5 05:30 bin
< drwxr-xr-x    2 root     root         4096 Feb  5 05:30 doc
---
> drwxr-xr-x    2 root     root        12288 Feb  3 00:52 bin
> drwxr-xr-x    2 root     root         4096 Feb  3 00:52 doc
5979c5984
< drwxr-xr-x   24 root     root         8192 Feb  5 05:30 lib
---
> drwxr-xr-x   24 root     root         8192 Feb  3 02:31 lib
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
5981c5986
< drwxr-xr-x    2 root     root         4096 Feb  5 05:30 sbin
---
> drwxr-xr-x    2 root     root         4096 Feb  3 02:57 sbin
6911c6916
< lrwxrwxrwx    1 root     root            3 Feb  5 05:30 slogin
-> ssh
---
> lrwxrwxrwx    1 root     root            3 Feb  3 00:22 slogin
-> ssh
7204c7209
< lrwxrwxrwx    1 root     root           16 Feb  5 05:30 ssh ->
../share/doc/ssh
---
> lrwxrwxrwx    1 root     root           16 Feb  3 00:28 ssh ->
../share/doc/ssh
11614c11619
< -rwxr-xr-x    1 root     root       241340 Jan 26 00:36 sshd
---
> -rwxr-xr-x    1 root     root       655916 Feb  3 02:30 sshd
```

Uh, oh. SSHD has been replaced...

```
11661c11666
< drwxr-xr-x  200 root     root         4096 Feb  5 05:30 doc
---
> drwxr-xr-x  200 root     root         4096 Feb  3 00:50 doc
12312c12317
< drwxr-xr-x    2 root     root         4096 Feb  5 05:30 ssh
---
> drwxr-xr-x    2 root     root         4096 Feb  3 00:22 ssh
16654c16659
< drwxr-xr-x    2 root     root        16384 Feb  5 05:30 man1
---
> drwxr-xr-x    2 root     root        16384 Feb  3 00:52 man1
16661c16666
< drwxr-xr-x    2 root     root         8192 Feb  5 05:30 man8
---
> drwxr-xr-x    2 root     root         8192 Feb  3 00:50 man8
17324c17329
< lrwxrwxrwx    1 root     root            8 Feb  5 05:30
slogin.1.gz -> ssh.1.gz
---
> lrwxrwxrwx    1 root     root            8 Feb  3 00:22
slogin.1.gz -> ssh.1.gz
32506c32511
< -rw-r--r--    1 root     src         16384 Aug  6  2001
mpparse.c
---
> -rw-r--r--    1 root     src         16384 Aug  6 17:29
mpparse.c
35007,35008c35012,35013
< -rw-r--r--    1 root     src        247961 Aug  6  2001
DAC960.c
< -rw-r--r--    1 root     src        143612 Aug  6  2001
DAC960.h
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
---
> -rw-r--r--    1 root     src        247961 Aug  6 17:34
DAC960.c
> -rw-r--r--    1 root     src        143612 Aug  6 17:34
DAC960.h
35200c35205
< -rw-r--r--    1 root     src         63208 Aug  6  2001 mxser.c
---
> -rw-r--r--    1 root     src         63208 Aug  6 17:33 mxser.c
```

A couple of grep searches:

```
root@debian:~# diff disk1.list1 disk1.list2 | grep sshd
> drwxr-sr-x    2 root     staff        4096 Jan 29 06:24 sshd
> /target/home/tomas/sshd:
< -rwxr-xr-x    1 root     root       241340 Jan 26 00:36 sshd
> -rwxr-xr-x    1 root     root       655916 Feb  3 02:30 sshd
root@debian:~# diff disk1.list1 disk1.list2 | grep ssh
< drwxr-xr-x    2 root     root         4096 Feb  5 05:30 ssh
> drwxr-xr-x    2 root     root         4096 Feb  3 00:28 ssh
< -rw-------    1 root     root          526 Feb  5 05:30
ssh_host_key
> -rw-------    1 root     root          526 Feb  3 00:28
ssh_host_key
> -rw-r--r--    1 root     staff        2538 Jan 29 06:28 falcon-
ssh-diffs.tar.gz
> drwxr-sr-x    2 root     staff        4096 Jan 29 06:24 sshd
> /target/home/tomas/sshd:
< lrwxrwxrwx    1 root     root            3 Feb  5 05:30 slogin
-> ssh
> lrwxrwxrwx    1 root     root            3 Feb  3 00:22 slogin
-> ssh
< lrwxrwxrwx    1 root     root           16 Feb  5 05:30 ssh ->
../share/doc/ssh
> lrwxrwxrwx    1 root     root           16 Feb  3 00:28 ssh ->
../share/doc/ssh
< -rwxr-xr-x    1 root     root       241340 Jan 26 00:36 sshd
> -rwxr-xr-x    1 root     root       655916 Feb  3 02:30 sshd
< drwxr-xr-x    2 root     root         4096 Feb  5 05:30 ssh
> drwxr-xr-x    2 root     root         4096 Feb  3 00:22 ssh
< lrwxrwxrwx    1 root     root            8 Feb  5 05:30
slogin.1.gz -> ssh.1.gz
> lrwxrwxrwx    1 root     root            8 Feb  3 00:22
slogin.1.gz -> ssh.1.gz
```

Actually, I got this listing by replacing sshd with an older version, generating a listing, and then updating to the latest version, then generating another listing. I also modified some associated files. The results are similar to what you would see if someone were attempting to sabotage sshd.

Note that the first grep search turns up the biggest nugget: sshd of two different sizes. This information is a little out of context, due to grep grabbing single lines from a larger list, and that list a diff output, but it is useful nonetheless. From this simple search we can see that sshd has been altered, and can narrow our focus to this file and it's

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

associated configuration files. We can always go back to the listing files or even the backup of the disk for more details.

© SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.